

Babel

Version 3.49
2020/10/03

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

I	User guide	4
1	The user interface	4
1.1	Monolingual documents	4
1.2	Multilingual documents	6
1.3	Mostly monolingual documents	8
1.4	Modifiers	8
1.5	Troubleshooting	9
1.6	Plain	9
1.7	Basic language selectors	9
1.8	Auxiliary language selectors	10
1.9	More on selection	11
1.10	Shorthands	12
1.11	Package options	16
1.12	The base option	18
1.13	ini files	18
1.14	Selecting fonts	27
1.15	Modifying a language	29
1.16	Creating a language	30
1.17	Digits and counters	33
1.18	Dates	35
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Selection based on BCP 47 tags	39
1.22	Selecting scripts	40
1.23	Selecting directions	41
1.24	Language attributes	45
1.25	Hooks	45
1.26	Languages supported by babel with ldf files	46
1.27	Unicode character properties in luatex	47
1.28	Tweaking some features	48
1.29	Tips, workarounds, known issues and notes	48
1.30	Current and future work	49
1.31	Tentative and experimental code	50
2	Loading languages with language.dat	50
2.1	Format	51
3	The interface between the core of babel and the language definition files	51
3.1	Guidelines for contributed languages	53
3.2	Basic macros	53
3.3	Skeleton	54
3.4	Support for active characters	55
3.5	Support for saving macro definitions	56
3.6	Support for extending macros	56
3.7	Macros common to a number of languages	56
3.8	Encoding-dependent strings	57
4	Changes	60
4.1	Changes in babel version 3.9	60
II	Source code	61

5	Identification and loading of required files	61
6	locale directory	61
7	Tools	62
7.1	Multiple languages	66
7.2	The Package File (\LaTeX , babel.sty)	67
7.3	base	69
7.4	Conditional loading of shorthands	71
7.5	Cross referencing macros	73
7.6	Marks	75
7.7	Preventing clashes with other packages	76
7.7.1	ifthen	76
7.7.2	varioref	77
7.7.3	hhline	78
7.7.4	hyperref	78
7.7.5	fancyhdr	78
7.8	Encoding and fonts	79
7.9	Basic bidi support	80
7.10	Local Language Configuration	86
8	The kernel of Babel (babel.def, common)	90
8.1	Tools	90
9	Multiple languages	91
9.1	Selecting the language	93
9.2	Errors	102
9.3	Hooks	105
9.4	Setting up language files	107
9.5	Shorthands	109
9.6	Language attributes	118
9.7	Support for saving macro definitions	120
9.8	Short tags	121
9.9	Hyphens	121
9.10	Multiencoding strings	123
9.11	Macros common to a number of languages	129
9.12	Making glyphs available	129
9.12.1	Quotation marks	129
9.12.2	Letters	131
9.12.3	Shorthands for quotation marks	132
9.12.4	Umlauts and tremas	133
9.13	Layout	134
9.14	Load engine specific macros	135
9.15	Creating and modifying languages	135
10	Adjusting the Babel bahavior	154
11	Loading hyphenation patterns	156
12	Font handling with fontspec	161

13	Hooks for XeTeX and LuaTeX	165
13.1	XeTeX	165
13.2	Layout	167
13.3	LuaTeX	169
13.4	Southeast Asian scripts	175
13.5	CJK line breaking	178
13.6	Automatic fonts and ids switching	179
13.7	Layout	189
13.8	Auto bidi with basic and basic-r	192
14	Data for CJK	203
15	The ‘nil’ language	203
16	Support for Plain T_EX (plain.def)	204
16.1	Not renaming hyphen.tex	204
16.2	Emulating some L _A T _E X features	205
16.3	General tools	205
16.4	Encoding related macros	209
17	Acknowledgements	211

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format	6
You are loading directly a language style	9
Unknown language ‘LANG’	9
Argument of \language@active@arg” has an extra }	13
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’	28
Package babel Info: The following fonts are not babel standard families	28

Part I

User guide

What is this document about? This user guide focuses on internationalization and localization with \LaTeX and pdf \TeX , xetex and luatex with the babel package. There are also some notes on its use with Plain \TeX . Part II describes the code, and usually it can be ignored.

What if I'm interested only in the latest changes? Changes and new features with relation to version 3.8 are highlighted with **New X.XX**, and there are some notes for the latest versions in [the babel wiki](#). The most recent features can be still unstable.

Can I help? Sure! If you are interested in the \TeX multilingual support, please join the [kadingira mail list](#). You can follow the development of babel in [GitHub](#) and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

It doesn't work for me! You can ask for help in some forums like tex.stackexchange, but if you have found a bug, I strongly beg you to report it in [GitHub](#), which is much better than just complaining on an e-mail list or a web forum.

How can I contribute a new language? See section 3.1 for contributing a language.

I only need learn the most basic features. The first subsections (1.1-1.3) describe the traditional way of loading a language (with ldf files), which is usually all you need. The alternative way based on ini files, which complements the previous one (it does *not* replace it, although it is still necessary in some languages), is described below; go to 1.13.

I don't like manuals. I prefer sample files. This manual contains lots of examples and tips, but in GitHub there are many [sample files](#).

1 The user interface

1.1 Monolingual documents

In most cases, a single language is required, and then all you need in \LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in \LaTeX for an option – in this case a language – to be recognized by several packages.

Many languages are compatible with xetex and luatex,. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current \LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to lmrroman. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

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

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

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

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

PDFTEX

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
\usepackage[utf8]{inputenc}  
  
\usepackage[english,french]{babel}  
  
\begin{document}  
  
Plus ça change, plus c'est la même chose!  
  
\selectlanguage{english}  
  
And an English paragraph, with a short text in  
\foreignlanguage{french}{français}.  
  
\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}  
  
\usepackage[vietnamese,danish]{babel}  
  
\begin{document}  
  
\prefacename{} -- \alsoname{} -- \today  
  
\selectlanguage{vietnamese}
```



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

\end{document}
```

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

1.3 Mostly monolingual documents

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

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

EXAMPLE A trivial document is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

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

1.5 Troubleshooting

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

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

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

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

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

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` `{⟨language⟩}`

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:

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”.

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

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

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

Spaces after the environment are ignored.

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

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

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things, for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of 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:

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

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).

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

\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 discretionary (languages do not define shorthands consistently, and “-”, “-”, “=” have different meanings). You can start with, say:

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

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

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

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

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

\languageshorthands {⟨language⟩}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the

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.

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

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

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

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

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

1.11 Package options

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

- KeepShorthandsActive** Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
- activeacute** For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
- activegrave** Same for `.
- shorthands=** `<char><char>... | off`
The only language shorthands activated are those given, like, eg:

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

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

- safe=** `none | ref | bib`
Some `TeX` macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from `varioref` and `ifthen`). With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in `TeX` based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

math=	active normal
	Shorthands are mainly intended for text, not for math. By setting this option with the value <code>normal</code> they are deactivated in math mode (default is <code>active</code>) and things like $\{a'\}$ (a closing brace after a shorthand) are not a source of trouble anymore.
config=	$\langle file \rangle$
	Load $\langle file \rangle$.cfg instead of the default config file <code>bblopts.cfg</code> (the file is loaded even with <code>noconfigs</code>).
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 spoilt by an unexpected .cfg file. However, if the key <code>config</code> is set, this file is loaded.
showlanguages	Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
nocase	New 3.9l Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code>) are ignored. Use only if there are incompatibilities with other packages.
silent	New 3.9l No warnings and no <i>infos</i> are written to the log file. ⁹
strings=	generic unicode encoded $\langle label \rangle$ $\langle font encoding \rangle$
	Selects the encoding of strings in languages supporting this feature. Predefined labels are <code>generic</code> (for traditional \TeX , LICR and ASCII strings), <code>unicode</code> (for engines like <code>xetex</code> and <code>luatex</code>) and <code>encoded</code> (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in <code>\MakeUppercase</code> and the like (this feature misuses some internal \TeX tools, so use it only as a last resort).
hyphenmap=	off first select other other*
	New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it. ¹⁰ It can take the following values:
	off deactivates this feature and no case mapping is applied;
	first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at <code>\begin{document}</code>), but also the first <code>\selectlanguage</code> in the preamble), and it's the default if a single language option has been stated. ¹¹

⁹You can use alternatively the package `silence`.

¹⁰Turned off in plain.

¹¹Duplicated options count as several ones.

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

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

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

layout=

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

1.12 The base option

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

\AfterBabelLanguage `{⟨option-name⟩}{⟨code⟩}`

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}
```

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

1.13 ini files

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

¹²Providing `foreign` is pointless, because the case mapping applied is that at the end of the paragraph, but if either `xetex` or `luatex` change this behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

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

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

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

LUATEX/XETEX

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამშარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამშარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

Or also:

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

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

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

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

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

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

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

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

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{໐ ໑ ໒ ໓ ໔ ໕} % Random
```

East Asia scripts Settings for either Simplified or Traditional should work out of the box, with basic line breaking with any renderer. Although for a few words and shorts texts the ini files should be fine, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX, etc.). This is what the class `ltjbook` does with `luatex`, which can be used in conjunction with the `ldf` for `japanese`, because the following piece of code loads `luatexja`:

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

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

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

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

af	Afrikaans ^{ul}	dsb	Lower Sorbian ^{ul}
agq	Aghem	dua	Duala
ak	Akan	dyo	Jola-Fonyi
am	Amharic ^{ul}	dz	Dzongkha
ar	Arabic ^{ul}	ebu	Embu
ar-DZ	Arabic ^{ul}	ee	Ewe
ar-MA	Arabic ^{ul}	el	Greek ^{ul}
ar-SY	Arabic ^{ul}	el-polyton	Polytonic Greek ^{ul}
as	Assamese	en-AU	English ^{ul}
asa	Asu	en-CA	English ^{ul}
ast	Asturian ^{ul}	en-GB	English ^{ul}
az-Cyrl	Azerbaijani	en-NZ	English ^{ul}
az-Latn	Azerbaijani	en-US	English ^{ul}
az	Azerbaijani ^{ul}	en	English ^{ul}
bas	Basaa	eo	Esperanto ^{ul}
be	Belarusian ^{ul}	es-MX	Spanish ^{ul}
bem	Bemba	es	Spanish ^{ul}
bez	Bena	et	Estonian ^{ul}
bg	Bulgarian ^{ul}	eu	Basque ^{ul}
bm	Bambara	ewo	Ewondo
bn	Bangla ^{ul}	fa	Persian ^{ul}
bo	Tibetan ^u	ff	Fulah
brx	Bodo	fi	Finnish ^{ul}
bs-Cyrl	Bosnian	fil	Filipino
bs-Latn	Bosnian ^{ul}	fo	Faroese
bs	Bosnian ^{ul}	fr	French ^{ul}
ca	Catalan ^{ul}	fr-BE	French ^{ul}
ce	Chechen	fr-CA	French ^{ul}
cgg	Chiga	fr-CH	French ^{ul}
chr	Cherokee	fr-LU	French ^{ul}
ckb	Central Kurdish	fur	Friulian ^{ul}
cop	Coptic	fy	Western Frisian
cs	Czech ^{ul}	ga	Irish ^{ul}
cu	Church Slavic	gd	Scottish Gaelic ^{ul}
cu-Cyrs	Church Slavic	gl	Galician ^{ul}
cu-Glag	Church Slavic	grc	Ancient Greek ^{ul}
cy	Welsh ^{ul}	gsw	Swiss German
da	Danish ^{ul}	gu	Gujarati
dav	Taita	guz	Gusii
de-AT	German ^{ul}	gv	Manx
de-CH	German ^{ul}	ha-GH	Hausa
de	German ^{ul}	ha-NE	Hausa ^l
dje	Zarma	ha	Hausa

haw	Hawaiian	mgo	Meta'
he	Hebrew ^{ul}	mk	Macedonian ^{ul}
hi	Hindi ^u	ml	Malayalam ^{ul}
hr	Croatian ^{ul}	mn	Mongolian
hsb	Upper Sorbian ^{ul}	mr	Marathi ^{ul}
hu	Hungarian ^{ul}	ms-BN	Malay ^l
hy	Armenian ^u	ms-SG	Malay ^l
ia	Interlingua ^{ul}	ms	Malay ^{ul}
id	Indonesian ^{ul}	mt	Maltese
ig	Igbo	mua	Mundang
ii	Sichuan Yi	my	Burmese
is	Icelandic ^{ul}	mzn	Mazanderani
it	Italian ^{ul}	naq	Nama
ja	Japanese	nb	Norwegian Bokmål ^{ul}
jgo	Ngomba	nd	North Ndebele
jmc	Machame	ne	Nepali
ka	Georgian ^{ul}	nl	Dutch ^{ul}
kab	Kabyle	nmg	Kwasio
kam	Kamba	nn	Norwegian Nynorsk ^{ul}
kde	Makonde	nnh	Ngiemboon
kea	Kabuverdianu	nus	Nuer
khq	Koyra Chiini	nyn	Nyankole
ki	Kikuyu	om	Oromo
kk	Kazakh	or	Odia
kkj	Kako	os	Ossetic
kl	Kalaallisut	pa-Arab	Punjabi
kln	Kalenjin	pa-Guru	Punjabi
km	Khmer	pa	Punjabi
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}
mgh	Makhuwa-Meetto	seh	Sena

ses	Koyraboro Senni	twq	Tasawaq
sg	Sango	tzm	Central Atlas Tamazight
shi-Latn	Tachelhit	ug	Uyghur
shi-Tfng	Tachelhit	uk	Ukrainian ^{ul}
shi	Tachelhit	ur	Urdu ^{ul}
si	Sinhala	uz-Arab	Uzbek
sk	Slovak ^{ul}	uz-Cyrl	Uzbek
sl	Slovenian ^{ul}	uz-Latn	Uzbek
smn	Inari Sami	uz	Uzbek
sn	Shona	vai-Latn	Vai
so	Somali	vai-Vaii	Vai
sq	Albanian ^{ul}	vai	Vai
sr-Cyrl-BA	Serbian ^{ul}	vi	Vietnamese ^{ul}
sr-Cyrl-ME	Serbian ^{ul}	vun	Vunjo
sr-Cyrl-XK	Serbian ^{ul}	wae	Walser
sr-Cyrl	Serbian ^{ul}	xog	Soga
sr-Latn-BA	Serbian ^{ul}	yav	Yangben
sr-Latn-ME	Serbian ^{ul}	yi	Yiddish
sr-Latn-XK	Serbian ^{ul}	yo	Yoruba
sr-Latn	Serbian ^{ul}	yue	Cantonese
sr	Serbian ^{ul}	zgh	Standard Moroccan Tamazight
sv	Swedish ^{ul}	zh-Hans-HK	Chinese
sw	Swahili	zh-Hans-MO	Chinese
ta	Tamil ^u	zh-Hans-SG	Chinese
te	Telugu ^{ul}	zh-Hans	Chinese
teo	Teso	zh-Hant-HK	Chinese
th	Thai ^{ul}	zh-Hant-MO	Chinese
ti	Tigrinya	zh-Hant	Chinese
tk	Turkmen ^{ul}	zh	Chinese
to	Tongan	zu	Zulu
tr	Turkish ^{ul}		

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

aghem	assamese
akan	asturian
albanian	asu
american	australian
amharic	austrian
ancientgreek	azerbaijani-cyrillic
arabic	azerbaijani-cyrl
arabic-algeria	azerbaijani-latin
arabic-DZ	azerbaijani-latn
arabic-morocco	azerbaijani
arabic-MA	bafia
arabic-syria	bambara
arabic-SY	basaa
armenian	basque

belarusian	english-au
bemba	english-australia
beni	english-ca
bengali	english-canada
bodo	english-gb
bosnian-cyrillic	english-newzealand
bosnian-cyrl	english-nz
bosnian-latin	english-unitedkingdom
bosnian-latn	english-unitedstates
bosnian	english-us
brazilian	english
breton	esperanto
british	estonian
bulgarian	ewe
burmese	ewondo
canadian	faroeese
cantonese	filipino
catalan	finnish
centralatlastamazight	french-be
centralkurdish	french-belgium
chechen	french-ca
cherokee	french-canada
chiga	french-ch
chinese-hans-hk	french-lu
chinese-hans-mo	french-luxembourg
chinese-hans-sg	french-switzerland
chinese-hans	french
chinese-hant-hk	friulian
chinese-hant-mo	fulah
chinese-hant	galician
chinese-simplified-hongkongsarchina	ganda
chinese-simplified-macausarchina	georgian
chinese-simplified-singapore	german-at
chinese-simplified	german-austria
chinese-traditional-hongkongsarchina	german-ch
chinese-traditional-macausarchina	german-switzerland
chinese-traditional	german
chinese	greek
churchslavic	gujarati
churchslavic-cyrs	gusii
churchslavic-oldcyrillic ¹³	hausa-gh
churchsslavic-glag	hausa-ghana
churchsslavic-glagolitic	hausa-ne
cognian	hausa-niger
cornish	hausa
croatian	hawaiian
czech	hebrew
danish	hindi
duala	hungarian
dutch	icelandic
dzongkha	igbo
embu	inarisami

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

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

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

sanskrit-gujarati	tachelhit-latn
sanskrit-gujr	tachelhit-tfng
sanskrit-kannada	tachelhit-tifinagh
sanskrit-knda	tachelhit
sanskrit-malayalam	taita
sanskrit-mlym	tamil
sanskrit-telu	tasawaq
sanskrit-telugu	telugu
sanskrit	teso
scottishgaelic	thai
sena	tibetan
serbian-cyrillic-bosniaherzegovina	tigrinya
serbian-cyrillic-kosovo	tongan
serbian-cyrillic-montenegro	turkish
serbian-cyrillic	turkmen
serbian-cyrl-ba	ukenglish
serbian-cyrl-me	ukrainian
serbian-cyrl-xk	upporsorbian
serbian-cyrl	urdu
serbian-latin-bosniaherzegovina	usenglish
serbian-latin-kosovo	usorbian
serbian-latin-montenegro	uyghur
serbian-latin	uzbek-arab
serbian-latn-ba	uzbek-arabic
serbian-latn-me	uzbek-cyrillic
serbian-latn-xk	uzbek-cyrl
serbian-latn	uzbek-latin
serbian	uzbek-latn
shambala	uzbek
shona	vai-latin
sichuanyi	vai-latn
sinhala	vai-vai
slovak	vai-vaii
slovene	vai
slovenian	vietnam
soga	vietnamese
somali	vunjo
spanish-mexico	walser
spanish-mx	welsh
spanish	westernfrisian
standardmoroccantamazight	yangben
swahili	yiddish
swedish	yoruba
swissgerman	zarma
tachelhit-latin	zulu afrikaans

Modifying and adding values to ini files

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

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

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

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

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

EXAMPLE Here is how to do it:

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

This is *not* and error. `babel` assumes that if you are using `\babelfont` for a family, very likely you want to define the rest of them. If you don’t, you can find some

inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.

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

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

1.15 Modifying a language

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

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

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

EXAMPLE If you need a language named `arhinish`:

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

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

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

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

The main language is not changed (danish in this example). So, you must add `\selectlanguage{arhinish}` or other selectors where necessary. If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

`import=` *`<language-tag>`*

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

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

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

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

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

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

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

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

`captions=` *`<language-tag>`*

Loads only the strings. For example:

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

`hyphenrules=` *`<language-list>`*

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

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

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

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


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

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

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

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

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

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

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

script= *<script-name>*

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

language= *<language-name>*

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

alph= *<counter-name>*

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

Alph= *<counter-name>*

Same for \Alph.

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

onchar= ids | fonts

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

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

`mapfont=` `direction`

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

`intraspace=` `<base>` `<shrink>` `<stretch>`

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

`intrapenalty=` `<penalty>`

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

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

1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

New 4.41 Many ‘ini’ locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with `xetex` and `luatex` and are fully expendable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000. There are several ways to use them (for the available styles in each language, see the list below):

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

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

The styles are:

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

Russian lower, lower.full, upper, upper.full
Syriac letters
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

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

1.18 Dates

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

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

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

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

1.19 Accessing language info

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

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

\iflanguage {*<language>*}{*<true>*}{*<false>*}

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

\localeinfo {*<field>*}

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

`name.english` as provided by the Unicode CLDR.
`tag.ini` is the tag of the ini file (the way this file is identified in its name).
`tag.bcp47` is the full BCP 47 tag (see the warning below).
`language.tag.bcp47` is the BCP 47 language tag.
`tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).
`script.name`, as provided by the Unicode CLDR.
`script.tag.bcp47` is the BCP 47 tag of the script used by this locale.

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

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

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

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

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

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

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

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

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

`\babelhyphen` *****{*type*}

`\babelhyphen` *****{*text*}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account). For example, you can use the `string` replacement to replace a character (or series of them) by another character (or series of them). Thus, to enter `ž` as `zh` and `š` as `sh` in a newly created locale for transliterated Russian:

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

```

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

```

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

1.21 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}

```

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

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

`autoloading.bcp47` with values on and off.

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

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

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

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

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

1.22 Selecting scripts

Currently `babel` provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.¹⁶ Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the `babel` core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was `LY1`), and therefore it has been deprecated.¹⁷

`\ensureascii` `{⟨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 taken into account, since they are not used for “ordinary” text (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).

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

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

¹⁷But still defined for backwards compatibility.

1.23 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}
```

```
\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the `bidi` package, which

provides its own mechanism to control these elements). You may use several options with a dot-separated list (eg, `layout=counters.contents.sectioning`). This list will be expanded in future releases. Note not all options are required by all engines.

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

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

1.24 Language attributes

`\languageattribute`

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

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

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

1.25 Hooks

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

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

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

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

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

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

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

defaultcommands Used (locally) in `\StartBabelCommands`.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.26 Languages supported by babel with ldf files

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

Afrikaans afrikaans

Azerbaijani azerbaijani

Basque basque

Breton breton

Bulgarian bulgarian

Catalan catalan

Croatian croatian

Czech czech

Danish danish

Dutch dutch

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

Esperanto esperanto

Estonian estonian

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

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

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

```

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

```

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

1.27 Unicode character properties in luatex

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

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

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

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

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

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

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

1.28 Tweaking some features

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

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

1.29 Tips, workarounds, known issues and notes

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

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

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

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

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

(A recent version of inputenc is required.)

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

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

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

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

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

zhspacing Spacing for CJK documents in xetex.

1.30 Current and future work

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

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

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

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

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

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

1.31 Tentative and experimental code

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

Labels

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

`\babelprehyphenation`

New 3.44 Note it is tentative, but the current behavior for glyphs should be correct. It is similar to `\babelposthyphenation`, but (as its name implies) applied before hyphenation. There are other differences: (1) the first argument is the locale instead the name of hyphenation patterns; (2) in the search patterns = has no special meaning (| is still reserved, but currently unused); (3) in the replacement, discretionaries are not accepted, only remove, , and string = ...
Currently it handles glyphs, not discretionaries or spaces (in particular, it will not catch the hyphen and you can’t insert or remove spaces). Also, you are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg. Performance is still somewhat poor.

2 Loading languages with `language.dat`

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

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

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

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

2.1 Format

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

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

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

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

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

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

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

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

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.

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

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

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

Some recommendations:

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

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

²⁶But not removed, for backward compatibility.

3.1 Guidelines for contributed languages

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

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

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

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

The following page provides a starting point for ldf files:

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

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

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

3.2 Basic macros

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

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

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

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

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

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

\providehyphenmins The macro \providehyphenmins should be used in the language definition files to set \lefthyphenmin and \righthyphenmin. This macro will check whether these parameters

	were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do <i>not</i> set them).
<code>\captions<lang></code>	The macro <code>\captions<lang></code> defines the macros that hold the texts to replace the original hard-wired texts.
<code>\date<lang></code>	The macro <code>\date<lang></code> defines <code>\today</code> .
<code>\extras<lang></code>	The macro <code>\extras<lang></code> contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
<code>\noextras<lang></code>	Because we want to let the user switch between languages, but we do not know what state \TeX might be in after the execution of <code>\extras<lang></code> , a macro that brings \TeX into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras<lang></code> .
<code>\bbl@declareattribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \TeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \TeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions<lang></code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

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

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

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



```

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

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

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

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

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

3.4 Support for active characters

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

`\initiate@active@char`

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

`\bbl@activate`
`\bbl@deactivate`

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

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

`\bbl@add@special`
`\bbl@remove@special` The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380] 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`. \LaTeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special⟨char⟩` and `\bbl@remove@special⟨char⟩` add and remove the character `⟨char⟩` to these two sets.

3.5 Support for saving macro definitions

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

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

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

3.6 Support for extending macros

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

3.7 Macros common to a number of languages

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

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

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

`\set@low@box` For some languages, quotes need to be lowered to the baseline. For this purpose the macro

²⁷This mechanism was introduced by Bernd Raichle.

`\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.

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

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

`\bbl@nonfrenchspacing`

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

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

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

\EndBabelCommands
```

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

²⁸In future releases further categories may be added.

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

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

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands `{\langle code \rangle}`

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

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

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

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

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

\StartBabelCommands{turkish}{}[
```

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

```

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

- `\select@language` did not set `\language`name. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.

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

Part II

Source code

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

5 Identification and loading of required files

Code documentation is still under revision.

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

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.

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

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

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

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

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

6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

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

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

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

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

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

7 Tools

```
1 <<version=3.49>>
2 <<date=2020/10/03>>
```

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@c1#1{\csname bbl@#1\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
```

```

18 \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19 \fi}
20 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
\bbl@add@list This internal macro adds its second argument to a comma separated list in its first
argument. When the list is not defined yet (or empty), it will be initiated. It presumes
expandable character strings.
21 \def\bbl@add@list#1#2{%
22 \edef#1{%
23 \bbl@ifunset{\bbl@stripslash#1}%
24 }%
25 {\ifx#1\@empty\else#1,\fi}%
26 #2}}
\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead,
\bbl@afterfi we take extra care to ‘throw’ it over the \else and \fi parts of an \if-statement30. These
macros will break if another \if... \fi statement appears in one of the arguments and it
is not enclosed in braces.
27 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
28 \long\def\bbl@afterfi#1\fi{\fi#1}
\bbl@exp Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple
and readable. Here \ stands for \noexpand and \<.> for \noexpand applied to a built
macro name (the latter does not define the macro if undefined to \relax, because it is
created locally). The result may be followed by extra arguments, if necessary.
29 \def\bbl@exp#1{%
30 \begingroup
31 \let\ \noexpand
32 \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
33 \edef\bbl@exp@aux{\endgroup#1}%
34 \bbl@exp@aux}
\bbl@trim The following piece of code is stolen (with some changes) from keyval, by David Carlisle. It
defines two macros: \bbl@trim and \bbl@trim@def. The first one strips the leading and
trailing spaces from the second argument and then applies the first argument (a macro,
\toks@ and the like). The second one, as its name suggests, defines the first argument as
the stripped second argument.
35 \def\bbl@tempa#1{%
36 \long\def\bbl@trim##1#2{%
37 \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38 \def\bbl@trim@c{%
39 \ifx\bbl@trim@a\@sptoken
40 \expandafter\bbl@trim@b
41 \else
42 \expandafter\bbl@trim@b\expandafter#1%
43 \fi}%
44 \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
45 \bbl@tempa{ }
46 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}
\bbl@ifunset To check if a macro is defined, we create a new macro, which does the same as
\@ifundefined. However, in an  $\epsilon$ -tex engine, it is based on \ifcsname, which is more
efficient, and do not waste memory.

```

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


```

48 \begingroup
49   \gdef\bbl@ifunset#1{%
50     \expandafter\ifx\csname#1\endcsname\relax
51       \expandafter\@firstoftwo
52     \else
53       \expandafter\@secondoftwo
54     \fi}
55 \bbl@ifunset{ifcsname}%
56 {}%
57 {\gdef\bbl@ifunset#1{%
58   \ifcsname#1\endcsname
59     \expandafter\ifx\csname#1\endcsname\relax
60       \bbl@afterelse\expandafter\@firstoftwo
61     \else
62       \bbl@afterfi\expandafter\@secondoftwo
63     \fi
64   \else
65     \expandafter\@firstoftwo
66   \fi}}
67 \endgroup

```

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

```

68 \def\bbl@ifblank#1{%
69   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bbl@ifset#1#2#3{%
72   \bbl@ifunset{#1}{#3}{\bbl@exp{\@nil\bbl@ifblank{#1}}{#3}{#2}}}

```

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

```

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

```

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

```

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

```

\bbl@replace

```

93 \def\bb1@replace#1#2#3{% in #1 -> repl #2 by #3
94   \toks@{}}%
95   \def\bb1@replace@aux##1#2##2#2{%
96     \ifx\bb1@nil##2%
97       \toks@\expandafter{\the\toks@##1}%
98     \else
99       \toks@\expandafter{\the\toks@##1#3}%
100     \bb1@afterfi
101     \bb1@replace@aux##2#2%
102   \fi}%
103 \expandafter\bb1@replace@aux#1#2\bb1@nil#2%
104 \edef#1{\the\toks@}}

```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in \bb1@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bb1@replace; I'm not sure checking the replacement is really necessary or just paranoia).

```

105 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
106   \bb1@exp{\def\\bb1@parsedef##1\detokenize{macro:}}#2->#3\relax{%
107     \def\bb1@tempa{#1}%
108     \def\bb1@tempb{#2}%
109     \def\bb1@tempe{#3}}
110   \def\bb1@sreplace#1#2#3{%
111     \begingroup
112     \expandafter\bb1@parsedef\meaning#1\relax
113     \def\bb1@tempc{#2}%
114     \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
115     \def\bb1@tempd{#3}%
116     \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
117     \bb1@xin@{\bb1@tempc}{\bb1@tempe}% If not in macro, do nothing
118     \ifin@
119       \bb1@exp{\\bb1@replace\\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
120       \def\bb1@tempc{% Expanded an executed below as 'uplevel'
121         \\makeatletter % "internal" macros with @ are assumed
122         \\scantokens{%
123           \bb1@tempa\\@namedef{\bb1@stripslash#1}\bb1@tempb{\bb1@tempe}}%
124         \catcode64=\the\catcode64\relax}% Restore @
125     \else
126       \let\bb1@tempc\@empty % Not \relax
127     \fi
128     \bb1@exp{% For the 'uplevel' assignments
129   \endgroup
130   \bb1@tempc}} % empty or expand to set #1 with changes
131 \fi

```

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.

```

132 \def\bb1@ifsamestring#1#2{%
133   \begingroup
134   \protected@edef\bb1@tempb{#1}%
135   \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
136   \protected@edef\bb1@tempc{#2}%
137   \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
138   \ifx\bb1@tempb\bb1@tempc

```

```

139     \aftergroup\@firstoftwo
140     \else
141     \aftergroup\@secondoftwo
142     \fi
143 \endgroup}
144 \chardef\bbl@engine=%
145 \ifx\directlua\@undefined
146     \ifx\XeTeXinputencoding\@undefined
147         \z@
148     \else
149         \tw@
150     \fi
151 \else
152     \@ne
153 \fi

```

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

```

154 \def\bbl@bspack{%
155     \ifhmode
156         \hskip\z@skip
157     \def\bbl@espack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158     \else
159         \let\bbl@espack\@empty
160     \fi}

```

Another hackish tool, to apply case changes inside a protected macros. It's based on the internal `\let's` made by `\MakeUppercase` and `\MakeLowercase` between things like `\oe` and `\OE`.

```

161 \def\bbl@cased{%
162     \ifx\oe\OE
163         \expandafter\in@\expandafter
164             {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166         \bbl@afterelse\expandafter\MakeUppercase
167     \else
168         \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170 \else
171     \expandafter\@firstofone
172     \fi}
173 <</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 .

```

174 <<*Make sure ProvidesFile is defined>> ≡
175 \ifx\ProvidesFile\@undefined
176     \def\ProvidesFile#1[#2 #3 #4]{%
177         \wlog{File: #1 #4 #3 <#2>}%
178         \let\ProvidesFile\@undefined}
179 \fi
180 <</Make sure ProvidesFile is defined>>

```

7.1 Multiple languages

`\language` Plain \TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in `switch.def` and `hyphen.cfg`; the latter

may seem redundant, but remember babel doesn't require loading `switch.def` in the format.

```
181 <<*Define core switching macros>> ≡
182 \ifx\language\undefined
183   \csname newcount\endcsname\language
184 \fi
185 <</Define core switching macros>>
```

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

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

```
186 <<*Define core switching macros>> ≡
187 <<*Define core switching macros>> ≡
188 \countdef\last@language=19 % TODO. why? remove?
189 \def\addlanguage{\csname newlanguage\endcsname}
190 <</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 \TeX 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

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

7.2 The Package File (\TeX , `babel.sty`)

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

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

The first two options are for debugging.

```
191 <*package>
192 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
193 \ProvidesPackage{babel}[\<date> \<version>] The Babel package]
194 \@ifpackagewith{babel}{debug}
195   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
196    \let\bbl@debug@firstofone}
197   {\providecommand\bbl@trace[1]{}%
198    \let\bbl@debug@gobble}
199 <<Basic macros>>
200 % Temporarily repeat here the code for errors
201 \def\bbl@error#1#2{%
202   \begingroup
203     \def\{\MessageBreak}%
204     \PackageError{babel}{#1}{#2}%
205   \endgroup}
206 \def\bbl@warning#1{%
207   \begingroup
208     \def\{\MessageBreak}%
209     \PackageWarning{babel}{#1}%
210 }
```

```

210 \endgroup}
211 \def\bbl@infowarn#1{%
212 \begingroup
213 \def\{\MessageBreak}%
214 \GenericWarning
215 {(babel) \@spaces\@spaces\@spaces}%
216 {Package babel Info: #1}%
217 \endgroup}
218 \def\bbl@info#1{%
219 \begingroup
220 \def\{\MessageBreak}%
221 \PackageInfo{babel}{#1}%
222 \endgroup}
223 \def\bbl@nocaption{\protect\bbl@nocaption@i}
224 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
225 \global\@namedef{#2}{\textbf{?#1?}}%
226 \@nameuse{#2}%
227 \bbl@warning{%
228 \@backslashchar#2 not set. Please, define it\\%
229 after the language has been loaded (typically\\%
230 in the preamble) with something like:\\%
231 \string\renewcommand\@backslashchar#2{..}\\%
232 Reported}}
233 \def\bbl@tentative{\protect\bbl@tentative@i}
234 \def\bbl@tentative@i#1{%
235 \bbl@warning{%
236 Some functions for '#1' are tentative.\\%
237 They might not work as expected and their behavior\\%
238 may change in the future.\\%
239 Reported}}
240 \def\@nolanerr#1{%
241 \bbl@error
242 {You haven't defined the language #1\space yet.\\%
243 Perhaps you misspelled it or your installation\\%
244 is not complete}%
245 {Your command will be ignored, type <return> to proceed}}
246 \def\@nopatterns#1{%
247 \bbl@warning
248 {No hyphenation patterns were preloaded for\\%
249 the language '#1' into the format.\\%
250 Please, configure your TeX system to add them and\\%
251 rebuild the format. Now I will use the patterns\\%
252 preloaded for \bbl@nulllanguage\space instead}}
253 % End of errors
254 \@ifpackagewith{babel}{silent}
255 {\let\bbl@info\@gobble
256 \let\bbl@infowarn\@gobble
257 \let\bbl@warning\@gobble}
258 {}
259 %
260 \def\AfterBabelLanguage#1{%
261 \global\expandafter\bbl@add\csname#1.ldf-ho@k\endcsname}%

If the format created a list of loaded languages (in \bbl@languages), get the name of the
0-th to show the actual language used. Also available with base, because it just shows info.

262 \ifx\bbl@languages\undefined\else
263 \begingroup
264 \catcode\^^I=12
265 \@ifpackagewith{babel}{showlanguages}{%

```

```

266 \begingroup
267 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}}%
268 \wlog{<*languages>}%
269 \bbl@languages
270 \wlog{</languages>}%
271 \endgroup}{%
272 \endgroup
273 \def\bbl@elt#1#2#3#4{%
274 \ifnum#2=\z@
275 \gdef\bbl@nulllanguage{#1}%
276 \def\bbl@elt##1##2##3##4{%
277 \fi}%
278 \bbl@languages
279 \fi%

```

7.3 base

The first ‘real’ option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that \LaTeX forgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.

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

```

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

```

```

315 #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
316 \def\bbl@tempd#1.#2@nnil{% TODO. Refactor lists?
317 \ifx\@empty#2%
318 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
319 \else
320 \in@{,provide,}{, #1,}%
321 \ifin@
322 \edef\bbl@tempc{%
323 \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
324 \else
325 \in@{=}{#1}%
326 \ifin@
327 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
328 \else
329 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
330 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
331 \fi
332 \fi
333 \fi}
334 \let\bbl@tempc\@empty
335 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty@nnil}
336 \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.

```

337 \DeclareOption{KeepShorthandsActive}{}
338 \DeclareOption{activeacute}{}
339 \DeclareOption{activegrave}{}
340 \DeclareOption{debug}{}
341 \DeclareOption{noconfigs}{}
342 \DeclareOption{showlanguages}{}
343 \DeclareOption{silent}{}
344 \DeclareOption{mono}{}
345 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
346 \chardef\bbl@iniflag\z@
347 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
348 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
349 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
350 % Don't use. Experimental. TODO.
351 \newif\ifbbl@single
352 \DeclareOption{selectors=off}{\bbl@singletrue}
353 \DeclareOption{provide@=*}{% % autoload with cat @=letter
354 \makeatother
355 \DeclareOption{provide@=*}{% % autoload with cat @=other
356 \makeatletter
357 <<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.

```

358 \let\bbl@opt@shorthands\@nnil
359 \let\bbl@opt@config\@nnil
360 \let\bbl@opt@main\@nnil
361 \let\bbl@opt@headfoot\@nnil
362 \let\bbl@opt@layout\@nnil

```

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

```

363 \def\bbl@tempa#1=#2\bbl@tempa{%
364   \bbl@csarg\ifx{opt@#1}\@nnil
365     \bbl@csarg\edef{opt@#1}{#2}%
366   \else
367     \bbl@error
368     {Bad option `#1=#2'. Either you have misspelled the\\%
369     key or there is a previous setting of `#1'. Valid\\%
370     keys are, among others, `shorthands', `main', `bidi',\\%
371     `strings', `config', `headfoot', `safe', `math'.}%
372     {See the manual for further details.}
373   \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.

```

374 \let\bbl@language@opts\@empty
375 \DeclareOption*{%
376   \bbl@xin@{\string=}{\CurrentOption}%
377   \ifin@
378     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
379   \else
380     \bbl@add@list\bbl@language@opts{\CurrentOption}%
381   \fi}

```

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

```

382 \ProcessOptions*

```

7.4 Conditional loading of shorthands

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

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

```

383 \bbl@trace{Conditional loading of shorthands}
384 \def\bbl@sh@string#1{%
385   \ifx#1\@empty\else
386     \ifx#1t\string~%
387     \else\ifx#1c\string,%
388     \else\string#1%
389   \fi\fi
390   \expandafter\bbl@sh@string
391 \fi}
392 \ifx\bbl@opt@shorthands\@nnil
393   \def\bbl@ifshorthand#1#2#3{#2}%
394 \else\ifx\bbl@opt@shorthands\@empty
395   \def\bbl@ifshorthand#1#2#3{#3}%
396 \else

```

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

```

397 \def\bbl@ifshorthand#1{%
398   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
399   \ifin@
400     \expandafter\@firstoftwo
401   \else

```



```

402     \expandafter\@secondoftwo
403     \fi}

```

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

```

404 \edef\bbl@opt@shorthands{%
405     \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.

```

406 \bbl@ifshorthand{'}%
407     {\PassOptionsToPackage{activeacute}{babel}}{}
408 \bbl@ifshorthand{`}%
409     {\PassOptionsToPackage{activegrave}{babel}}{}
410 \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.

```

411 \ifx\bbl@opt@headfoot\@nnil\else
412     \g@addto@macro\@resetactivechars{%
413         \set@typeset@protect
414         \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
415         \let\protect\noexpand}
416 \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.

```

417 \ifx\bbl@opt@safe\@undefined
418     \def\bbl@opt@safe{BR}
419 \fi
420 \ifx\bbl@opt@main\@nnil\else
421     \edef\bbl@language@opts{%
422         \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
423         \bbl@opt@main}
424 \fi

```

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

```

425 \bbl@trace{Defining IfBabelLayout}
426 \ifx\bbl@opt@layout\@nnil
427     \newcommand\IfBabelLayout[3]{#3}%
428 \else
429     \newcommand\IfBabelLayout[1]{%
430         \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
431         \ifin@
432             \expandafter\@firstoftwo
433         \else
434             \expandafter\@secondoftwo
435         \fi}
436 \fi

```

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

```

437 \input babel.def

```

7.5 Cross referencing macros

The \LaTeX book states:

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

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

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

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

```
438 <<*More package options>> ≡
439 \DeclareOption{safe=none}{\let\bbl@opt@safe\empty}
440 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
441 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
442 <</More package options>>
```

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

```
443 \bbl@trace{Cross referencing macros}
444 \ifx\bbl@opt@safe\empty\else
445   \def\@newl@bel#1#2#3{%
446     {\@safe@activetrue
447       \bbl@ifunset{#1@#2}%
448         \relax
449         {\gdef\@multiplelabels{%
450           \latex@warning@no@line{There were multiply-defined labels}}%
451           \latex@warning@no@line{Label `#2' multiply defined}}%
452       \global\@namedef{#1@#2}{#3}}}
```

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

```
453 \CheckCommand*\@testdef[3]{%
454   \def\reserved@a{#3}%
455   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
456   \else
457     \@tempwattrue
458     \fi}
```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’. Then we use `\bbl@tempa` as an ‘alias’ for the macro that contains the label which is being checked. Then we define `\bbl@tempb` just as `\@newl@bel` does it. When the label is defined we replace the definition of `\bbl@tempa` by its meaning. If the label didn’t change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

```
459 \def\@testdef#1#2#3{% TODO. With @samestring?
460   \@safe@activetrue
461   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
462   \def\bbl@tempb{#3}%
463   \@safe@activesfalse
464   \ifx\bbl@tempa\relax
465   \else
466     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
467     \fi
468   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

```

469 \ifx\bbl@tempa\bbl@tempb
470 \else
471 \@tempswatrue
472 \fi}
473 \fi

```

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

```

474 \bbl@xin@{R}\bbl@opt@safe
475 \ifin@
476 \bbl@redefineroobust\ref#1{%
477 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
478 \bbl@redefineroobust\pageref#1{%
479 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
480 \else
481 \let\org@ref\ref
482 \let\org@pageref\pageref
483 \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.

```

484 \bbl@xin@{B}\bbl@opt@safe
485 \ifin@
486 \bbl@redefine\@citex[#1]#2{%
487 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
488 \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.

```

489 \AtBeginDocument{%
490 \@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.)

```

491 \def\@citex[#1][#2]#3{%
492 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
493 \org@@citex[#1][#2]{\@tempa}}%
494 }{}}

```

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

```

495 \AtBeginDocument{%
496 \@ifpackageloaded{cite}{%
497 \def\@citex[#1]#2{%
498 \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
499 }{}}

```

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

```

500 \bbl@redefine\nocite#1{%
501   \@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.

502 \bbl@redefine\bibcite{%
503   \bbl@cite@choice
504   \bibcite}

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

505 \def\bbl@bibcite#1#2{%
506   \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.

507 \def\bbl@cite@choice{%
508   \global\let\bibcite\bbl@bibcite
509   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
510   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
511   \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.

512 \AtBeginDocument{\bbl@cite@choice}

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

513 \bbl@redefine\@bibitem#1{%
514   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
515 \else
516   \let\org@nocite\nocite
517   \let\org@@citex\@citex
518   \let\org@bibcite\bibcite
519   \let\org@bibitem\@bibitem
520 \fi

```

7.6 Marks

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

```

521 \bbl@trace{Marks}
522 \IfBabelLayout{sectioning}
523   {\ifx\bbl@opt@headfoot\@nnil
524     \g@addto@macro\@resetactivechars{%

```

```

525     \set@typeset@protect
526     \expandafter\select@language@x\expandafter{\bbl@main@language}%
527     \let\protect\noexpand
528     \ifcase\bbl@bidimode\else % Only with bidi. See also above
529         \edef\thepage{%
530             \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
531     \fi}%
532 \fi}
533 {\ifbbl@single\else
534     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
535     \markright#1{%
536         \bbl@ifblank{#1}%
537         {\org@markright{}}}%
538         {\toks@{#1}%
539         \bbl@exp{%
540             \\org@markright{\\protect\\foreignlanguage{\language}%
541                 {\\protect\\bbl@restore@actives\the\toks@}}}}}%
\markboth The definition of \markboth is equivalent to that of \markright, except that we need two
\@mkboth token registers. The documentclasses report and book define and set the headings for the
page. While doing so they also store a copy of \markboth in \@mkboth. Therefore we need
to check whether \@mkboth has already been set. If so we need to do that again with the
new definition of \markboth. (As of Oct 2019, LATEX stores the definition in an intermediate
macro, so it's not necessary anymore, but it's preserved for older versions.)
542     \ifx\@mkboth\markboth
543         \def\bbl@tempc{\let\@mkboth\markboth}
544     \else
545         \def\bbl@tempc{}
546     \fi
547     \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
548     \markboth#1#2{%
549         \protected@edef\bbl@tempb##1{%
550             \protect\foreignlanguage
551             {\language}{\protect\bbl@restore@actives##1}}%
552         \bbl@ifblank{#1}%
553         {\toks@{}}%
554         {\toks@\expandafter{\bbl@tempb{#1}}}%
555         \bbl@ifblank{#2}%
556         {\@temptokena{}}%
557         {\@temptokena\expandafter{\bbl@tempb{#2}}}%
558         \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}
559     \bbl@tempc
560 \fi} % end ifbbl@single, end \IfBabelLayout

```

7.7 Preventing clashes with other packages

7.7.1 ifthen

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

```

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

```

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

We want to revert the definition of `\pageref` and `\ref` to their original definition for the first argument of `\ifthenelse`, so we first need to store their current meanings. Then we can set the `\@safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments.

```

561 \bbl@trace{Preventing clashes with other packages}
562 \bbl@xin@{R}\bbl@opt@safe
563 \ifin@
564 \AtBeginDocument{%
565   \@ifpackageloaded{ifthen}{%
566     \bbl@redefine@long\ifthenelse#1#2#3{%
567       \let\bbl@temp@pref\pageref
568       \let\pageref\org@pageref
569       \let\bbl@temp@ref\ref
570       \let\ref\org@ref
571       \@safe@activestrue
572       \org@ifthenelse{#1}%
573       {\let\pageref\bbl@temp@pref
574        \let\ref\bbl@temp@ref
575        \@safe@activesfalse
576        #2}%
577       {\let\pageref\bbl@temp@pref
578        \let\ref\bbl@temp@ref
579        \@safe@activesfalse
580        #3}%
581     }%
582   }{}%
583 }

```

7.7.2 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`.
`\vrefpagenum` The same needs to happen for `\vrefpagenum`.
`\Ref`

```

584 \AtBeginDocument{%
585   \@ifpackageloaded{varioref}{%
586     \bbl@redefine\@@vpageref#1[#2]#3{%
587       \@safe@activestrue
588       \org@@@vpageref{#1}[#2]{#3}%
589       \@safe@activesfalse}%
590     \bbl@redefine\vrefpagenum#1#2{%
591       \@safe@activestrue
592       \org@vrefpagenum{#1}{#2}%
593       \@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.

```

594   \expandafter\def\csname Ref \endcsname#1{%
595     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
596   }{}%
597 }
598 \fi

```

7.7.3 hline

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

```
599 \AtEndOfPackage{%
600   \AtBeginDocument{%
601     \ifpackageloaded{hline}%
602       {\expandafter\ifx\csname normal@char\string\endcsname\relax
603         \else
604           \makeatletter
605           \def\@currname{hline}\input{hline.sty}\makeatother
606         \fi}%
607       {}}}
```

7.7.4 hyperref

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

```
608 % \AtBeginDocument{%
609 %   \ifx\pdfstringdefDisableCommands\undefined\else
610 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
611 %   \fi}
```

7.7.5 fancyhdr

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

```
612 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
613   \lowercase{\foreignlanguage{#1}}}
```

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

```
614 \def\substitutefontfamily#1#2#3{%
615   \lowercase{\immediate\openout15=#1#2.fd\relax}%
616   \immediate\write15{%
617     \string\ProvidesFile{#1#2.fd}%
618     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
619     \space generated font description file]^^J
620     \string\DeclareFontFamily{#1}{#2}{^^J
621     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
622     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
623     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
624     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
625     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
626     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
627     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
628     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
```

```

629 }%
630 \closeout15
631 }
632 \@onlypreamble\substitutefontfamily

```

7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing `\@filelist` to search for `\encenc.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`

```

633 \bbl@trace{Encoding and fonts}
634 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
635 \newcommand\BabelNonText{TS1,T3,TS3}
636 \let\org@TeX\TeX
637 \let\org@LaTeX\LaTeX
638 \let\ensureascii\@firstofone
639 \AtBeginDocument{%
640   \in@false
641   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
642     \ifin@ \else
643       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
644       \fi}%
645   \ifin@ % if a text non-ascii has been loaded
646     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
647     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
648     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
649     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
650     \def\bbl@tempc#1ENC.DEF#2\@@{\%
651       \ifx\@empty#2\else
652         \bbl@ifunset{T#1}%
653         {}%
654         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
655         \ifin@
656           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
657           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
658         \else
659           \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
660           \fi}%
661     \fi}%
662   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
663   \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
664   \ifin@ \else
665     \edef\ensureascii#1{%
666       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
667   \fi
668 \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.

```
669 \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.

```
670 \AtBeginDocument{%
671   \ifpackageloaded{fontspec}%
672     {\xdef\latinencoding{%
673       \ifx\UTFencname\@undefined
674         EU\ifcase\bb1@engine\or2\or1\fi
675       \else
676         \UTFencname
677       \fi}}%
678   {\gdef\latinencoding{OT1}%
679     \ifx\cf@encoding\bb1@t@one
680       \xdef\latinencoding{\bb1@t@one}%
681     \else
682       \ifx\@fontenc@load@list\@undefined
683         \ifl@aded{def}{t1enc}{\xdef\latinencoding{\bb1@t@one}}}%
684       \else
685         \def\@elt#1{, #1,}%
686         \edef\bb1@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
687         \let\@elt\relax
688         \bb1@xin@{, T1, }\bb1@tempa
689         \ifin@
690           \xdef\latinencoding{\bb1@t@one}%
691         \fi
692       \fi
693     \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.

```
694 \DeclareRobustCommand{\latintext}{%
695   \fontencoding{\latinencoding}\selectfont
696   \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.

```
697 \ifx\@undefined\DeclareTextFontCommand
698   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
699 \else
700   \DeclareTextFontCommand{\textlatin}{\latintext}
701 \fi
```

7.9 Basic bidi support

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

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

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

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

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

```

702 \ifodd\bbl@engine
703   \def\bbl@activate@preotf{%
704     \let\bbl@activate@preotf\relax % only once
705     \directlua{
706       Babel = Babel or {}
707       %
708       function Babel.pre_otfload_v(head)
709         if Babel.numbers and Babel.digits_mapped then
710           head = Babel.numbers(head)
711         end
712         if Babel.bidi_enabled then
713           head = Babel.bidi(head, false, dir)
714         end
715         return head
716       end
717       %
718       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
719         if Babel.numbers and Babel.digits_mapped then
720           head = Babel.numbers(head)
721         end
722         if Babel.bidi_enabled then
723           head = Babel.bidi(head, false, dir)
724         end
725         return head
726       end
727       %
728       luatexbase.add_to_callback('pre_linebreak_filter',
729         Babel.pre_otfload_v,
730         'Babel.pre_otfload_v',
731         luatexbase.priority_in_callback('pre_linebreak_filter',
732           'luaotfload.node_processor') or nil)
733       %
734       luatexbase.add_to_callback('hpack_filter',
735         Babel.pre_otfload_h,
736         'Babel.pre_otfload_h',
737         luatexbase.priority_in_callback('hpack_filter',
738           'luaotfload.node_processor') or nil)
739     }}
740 \fi

```

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

```

741 \bbl@trace{Loading basic (internal) bidi support}
742 \ifodd\bbl@engine
743   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
744     \let\bbl@beforeforeign\leavevmode
745     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
746     \RequirePackage{luatexbase}
747     \bbl@activate@preotf
748     \directlua{
749       require('babel-data-bidi.lua')
750       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
751         require('babel-bidi-basic.lua')
752       \or
753         require('babel-bidi-basic-r.lua')
754       \fi}
755     % TODO - to locale_props, not as separate attribute
756     \newattribute\bbl@attr@dir
757     % TODO. I don't like it, hackish:
758     \bbl@exp{\output{\bodydir\pagedir\the\output}}
759     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
760   \fi\fi
761 \else
762   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
763     \bbl@error
764     {The bidi method 'basic' is available only in\\%
765      luatex. I'll continue with 'bidi=default', so\\%
766      expect wrong results}%
767     {See the manual for further details.}%
768     \let\bbl@beforeforeign\leavevmode
769     \AtEndOfPackage{%
770       \EnableBabelHook{babel-bidi}%
771       \bbl@xebidipar}
772   \fi\fi
773   \def\bbl@loadxebidi#1{%
774     \ifx\RTLfootnotetext\@undefined
775       \AtEndOfPackage{%
776         \EnableBabelHook{babel-bidi}%
777         \ifx\fontspec\@undefined
778           \bbl@loadfontspec % bidi needs fontspec
779         \fi
780         \usepackage#1{bidi}}%
781     \fi}
782   \ifnum\bbl@bidimode>200
783     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
784       \bbl@tentative{bidi=bidi}
785       \bbl@loadxebidi{}
786     \or
787       \bbl@loadxebidi{[rldocument]}
788     \or
789       \bbl@loadxebidi{}
790     \fi
791   \fi
792 \fi
793 \ifnum\bbl@bidimode=\@ne
794   \let\bbl@beforeforeign\leavevmode
795   \ifodd\bbl@engine
796     \newattribute\bbl@attr@dir

```

```

797 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
798 \fi
799 \AtEndOfPackage{%
800 \EnableBabelHook{babel-bidi}%
801 \ifodd\bbl@engine\else
802 \bbl@xebidipar
803 \fi}
804 \fi

```

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

```

805 \bbl@trace{Macros to switch the text direction}
806 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
807 \def\bbl@rscripts{% TODO. Base on codes ??
808 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
809 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
810 Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
811 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
812 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
813 Old South Arabian,}%
814 \def\bbl@provide@dirs#1{%
815 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
816 \ifin@
817 \global\bbl@csarg\chardef{wdir@#1}\@ne
818 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
819 \ifin@
820 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
821 \fi
822 \else
823 \global\bbl@csarg\chardef{wdir@#1}\z@
824 \fi
825 \ifodd\bbl@engine
826 \bbl@csarg\ifcase{wdir@#1}%
827 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
828 \or
829 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
830 \or
831 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
832 \fi
833 \fi}
834 \def\bbl@switchdir{%
835 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
836 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
837 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
838 \def\bbl@setdirs#1{% TODO - math
839 \ifcase\bbl@select@type % TODO - strictly, not the right test
840 \bbl@bodydir{#1}%
841 \bbl@pardir{#1}%
842 \fi
843 \bbl@texmdir{#1}}
844 % TODO. Only if \bbl@bidimode > 0?:
845 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
846 \DisableBabelHook{babel-bidi}

```

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

```

847 \ifodd\bbl@engine % luatex=1
848 \chardef\bbl@thetexmdir\z@
849 \chardef\bbl@thepardir\z@
850 \def\bbl@getluadir#1{%

```

```

851 \directlua{
852   if tex.#1dir == 'TLT' then
853     tex.sprint('0')
854   elseif tex.#1dir == 'TRT' then
855     tex.sprint('1')
856   end}}
857 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
858   \ifcase#3\relax
859     \ifcase\bbl@getluadir{#1}\relax\else
860       #2 TLT\relax
861     \fi
862   \else
863     \ifcase\bbl@getluadir{#1}\relax
864       #2 TRT\relax
865     \fi
866   \fi}
867 \def\bbl@textdir#1{%
868   \bbl@setluadir{text}\textdir{#1}%
869   \chardef\bbl@thetextdir#1\relax
870   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
871 \def\bbl@pardir#1{%
872   \bbl@setluadir{par}\pardir{#1}%
873   \chardef\bbl@thepardir#1\relax}
874 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
875 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
876 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
877 % Sadly, we have to deal with boxes in math with basic.
878 % Activated every math with the package option bidi=:
879 \def\bbl@mathboxdir{%
880   \ifcase\bbl@thetextdir\relax
881     \everyhbox{\textdir TLT\relax}%
882   \else
883     \everyhbox{\textdir TRT\relax}%
884   \fi}
885 \frozen@everymath\expandafter{%
886   \expandafter\bbl@mathboxdir\the\frozen@everymath}
887 \frozen@everydisplay\expandafter{%
888   \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
889 \else % pdftex=0, xetex=2
890   \newcount\bbl@dirlevel
891   \chardef\bbl@thetextdir\z@
892   \chardef\bbl@thepardir\z@
893   \def\bbl@textdir#1{%
894     \ifcase#1\relax
895       \chardef\bbl@thetextdir\z@
896       \bbl@textdir@i\beginL\endL
897     \else
898       \chardef\bbl@thetextdir\@ne
899       \bbl@textdir@i\beginR\endR
900     \fi}
901   \def\bbl@textdir@i#1#2{%
902     \ifhmode
903       \ifnum\currentgrouplevel>\z@
904         \ifnum\currentgrouplevel=\bbl@dirlevel
905           \bbl@error{Multiple bidi settings inside a group}%
906           {I'll insert a new group, but expect wrong results.}%
907           \bgroup\aftergroup#2\aftergroup\egroup
908         \else
909           \ifcase\currentgrouptype\or % 0 bottom

```

```

910         \aftergroup#2% 1 simple {}
911     \or
912         \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
913     \or
914         \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
915     \or\or\or % vbox vtop align
916     \or
917         \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
918     \or\or\or\or\or\or\or % output math disc insert vcent mathchoice
919     \or
920         \aftergroup#2% 14 \begingroup
921     \else
922         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
923     \fi
924 \fi
925 \bbl@dirlevel\currentgrouplevel
926 \fi
927 #1%
928 \fi}
929 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
930 \let\bbl@bodydir\@gobble
931 \let\bbl@pagedir\@gobble
932 \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).

```

933 \def\bbl@xebidipar{%
934     \let\bbl@xebidipar\relax
935     \TeXeTstate\@ne
936     \def\bbl@xeverypar{%
937         \ifcase\bbl@thepardir
938             \ifcase\bbl@thetextdir\else\beginR\fi
939         \else
940             {\setbox\z@\lastbox\beginR\box\z@}%
941         \fi}%
942     \let\bbl@severypar\everypar
943     \newtoks\everypar
944     \everypar=\bbl@severypar
945     \bbl@severypar{\bbl@xeverypar\the\everypar}}
946 \ifnum\bbl@bidimode>200
947     \let\bbl@textdir\i\@gobbletwo
948     \let\bbl@xebidipar\@empty
949     \AddBabelHook{bidi}{foreign}{%
950         \def\bbl@tempa{\def\BabelText####1}%
951         \ifcase\bbl@thetextdir
952             \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
953         \else
954             \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
955         \fi}
956     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
957 \fi
958 \fi

```

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

```

959 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}
960 \AtBeginDocument{%
961     \ifx\pdfstringdefDisableCommands\@undefined\else
962         \ifx\pdfstringdefDisableCommands\relax\else

```

```

963 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
964 \fi
965 \fi}

```

7.10 Local Language Configuration

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

```

966 \bbl@trace{Local Language Configuration}
967 \ifx\loadlocalcfg\@undefined
968 \ifpackagewith{babel}{noconfigs}%
969 {\let\loadlocalcfg\@gobble}%
970 {\def\loadlocalcfg#1{%
971 \InputIfFileExists{#1.cfg}%
972 {\typeout{*****^J%
973 * Local config file #1.cfg used^^J%
974 *}}}%
975 \@empty}}
976 \fi

```

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

```

977 \ifx\@unexpandable@protect\@undefined
978 \def\@unexpandable@protect{\noexpand\protect\noexpand}
979 \long\def\protected@write#1#2#3{%
980 \begingroup
981 \let\thepage\relax
982 #2%
983 \let\protect\@unexpandable@protect
984 \edef\reserved@a{\write#1{#3}}%
985 \reserved@a
986 \endgroup
987 \if@nobreak\ifvmode\nobreak\fi\fi}
988 \fi
989 %
990 % \subsection{Language options}
991 %
992 % Languages are loaded when processing the corresponding option
993 % \textit{except} if a |main| language has been set. In such a
994 % case, it is not loaded until all options has been processed.
995 % The following macro inputs the ldf file and does some additional
996 % checks (|\input| works, too, but possible errors are not caught).
997 %
998 % \begin{macrocode}
999 \bbl@trace{Language options}
1000 \let\bbl@afterlang\relax
1001 \let\BabelModifiers\relax
1002 \let\bbl@loaded\@empty
1003 \def\bbl@load@language#1{%
1004 \InputIfFileExists{#1.ldf}%
1005 {\edef\bbl@loaded{\CurrentOption
1006 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1007 \expandafter\let\expandafter\bbl@afterlang

```

```

1008 \csname\CurrentOption.ldf-h@@k\endcsname
1009 \expandafter\let\expandafter\BabelModifiers
1010 \csname bbl@mod@\CurrentOption\endcsname}%
1011 {\bbl@error{%
1012   Unknown option '\CurrentOption'. Either you misspelled it\\%
1013   or the language definition file \CurrentOption.ldf was not found}{%
1014   Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1015   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1016   headfoot=, strings=, config=, hyphenmap=, or a language name.}}

```

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

```

1017 \def\bbl@try@load@lang#1#2#3{%
1018   \IfFileExists{\CurrentOption.ldf}%
1019   {\bbl@load@language{\CurrentOption}}%
1020   {#1\bbl@load@language{#2}#3}}
1021 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
1022 \DeclareOption{hebrew}{%
1023   \input{rlbabel.def}%
1024   \bbl@load@language{hebrew}}
1025 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1026 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1027 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1028 \DeclareOption{polutonikogreek}{%
1029   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1030 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1031 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1032 \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.

```

1033 \ifx\bbl@opt@config\@nnil
1034   \@ifpackagewith{babel}{noconfigs}{}%
1035   {\InputIfFileExists{bblopts.cfg}%
1036     {\typeout{*****^J%
1037               * Local config file bblopts.cfg used^^J%
1038               *}}}%
1039   }{}%
1040 \else
1041   \InputIfFileExists{\bbl@opt@config.cfg}%
1042   {\typeout{*****^J%
1043             * Local config file \bbl@opt@config.cfg used^^J%
1044             *}}%
1045   {\bbl@error{%
1046     Local config file '\bbl@opt@config.cfg' not found}{%
1047     Perhaps you misspelled it.}}%
1048 \fi

```

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

```

1049 \let\bbl@tempc\relax

```



```

1050 \bbl@foreach\bbl@language@opts{%
1051   \ifcase\bbl@iniflag
1052     \bbl@ifunset{ds@#1}%
1053     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1054     {}%
1055   \or
1056     \@gobble % case 2 same as 1
1057   \or
1058     \bbl@ifunset{ds@#1}%
1059     {\IfFileExists{#1.ldf}{}%
1060      {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}}{}}}%
1061     {}%
1062     \bbl@ifunset{ds@#1}%
1063     {\def\bbl@tempc{#1}%
1064      \DeclareOption{#1}{%
1065        \ifnum\bbl@iniflag>\@ne
1066          \bbl@ldfinit
1067          \babelprovide[import]{#1}%
1068          \bbl@afterldf}%
1069        \else
1070          \bbl@load@language{#1}%
1071        \fi}}%
1072     {}%
1073   \or
1074     \def\bbl@tempc{#1}%
1075     \bbl@ifunset{ds@#1}%
1076     {\DeclareOption{#1}{%
1077       \bbl@ldfinit
1078       \babelprovide[import]{#1}%
1079       \bbl@afterldf}}}%
1080     {}%
1081   \fi}

```

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.

```

1082 \let\bbl@tempb\@nnil
1083 \bbl@foreach\@classoptionslist{%
1084   \bbl@ifunset{ds@#1}%
1085   {\IfFileExists{#1.ldf}{}%
1086    {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}}{}}}%
1087   {}%
1088   \bbl@ifunset{ds@#1}%
1089   {\def\bbl@tempb{#1}%
1090    \DeclareOption{#1}{%
1091      \ifnum\bbl@iniflag>\@ne
1092        \bbl@ldfinit
1093        \babelprovide[import]{#1}%
1094        \bbl@afterldf}%
1095      \else
1096        \bbl@load@language{#1}%
1097      \fi}}%
1098   {}%

```

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

```

1099 \ifnum\bbl@iniflag=\z@\else
1100   \ifx\bbl@opt@main\@nnil
1101     \ifx\bbl@tempc\relax
1102       \let\bbl@opt@main\bbl@tempb

```

```

1103 \else
1104 \let\bbl@opt@main\bbl@tempc
1105 \fi
1106 \fi
1107 \fi
1108 \ifx\bbl@opt@main\@nnil\else
1109 \expandafter
1110 \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1111 \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1112 \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):

```

1113 \def\AfterBabelLanguage#1{%
1114 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1115 \DeclareOption*{}
1116 \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.

```

1117 \bbl@trace{Option 'main'}
1118 \ifx\bbl@opt@main\@nnil
1119 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1120 \let\bbl@tempc\@empty
1121 \bbl@for\bbl@tempb\bbl@tempa{%
1122 \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
1123 \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1124 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1125 \expandafter\bbl@tempa\bbl@loaded,\@nnil
1126 \ifx\bbl@tempb\bbl@tempc\else
1127 \bbl@warning{%
1128 Last declared language option is '\bbl@tempc',\%
1129 but the last processed one was '\bbl@tempb'.\%
1130 The main language cannot be set as both a global\%
1131 and a package option. Use 'main=\bbl@tempc' as\%
1132 option. Reported}%
1133 \fi
1134 \else
1135 \ifodd\bbl@iniflag % case 1,3
1136 \bbl@ldfinit
1137 \let\CurrentOption\bbl@opt@main
1138 \bbl@exp{\@babelprovide[import,main]{\bbl@opt@main}}
1139 \bbl@afterldf}%
1140 \else % case 0,2
1141 \chardef\bbl@iniflag\z@ % Force ldf
1142 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1143 \DeclareOption*{}%
1144 \ProcessOptions*
1145 \fi
1146 \fi
1147 \def\AfterBabelLanguage{%
1148 \bbl@error
1149 {Too late for \string\AfterBabelLanguage}%
1150 {Languages have been loaded, so I can do nothing}}

```

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

```

1151 \ifx\bbl@main@language\@undefined
1152   \bbl@info{%
1153     You haven't specified a language. I'll use 'nil'\%
1154     as the main language. Reported}
1155   \bbl@load@language{nil}
1156 \fi
1157 </package>
1158 <*core>

```

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

The kernel of the babel system is currently stored in `babel.def`. The file `babel.def` contains most of the code. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns. Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only. Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don't load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

8.1 Tools

```

1159 \ifx\ldf@quit\@undefined\else
1160 \endinput\fi % Same line!
1161 <<Make sure ProvidesFile is defined>>
1162 \ProvidesFile{babel.def}[\<date>] \<version>] Babel common definitions]

```

The file `babel.def` expects some definitions made in the $\LaTeX 2_{\epsilon}$ style file. So, In $\LaTeX 2.09$ and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore 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).

```

1163 \ifx\AtBeginDocument\@undefined % TODO. change test.
1164   <<Emulate LaTeX>>
1165   \def\language{english}%
1166   \let\bbl@opt@shorthands\@nnil
1167   \def\bbl@ifshorthand#1#2#3{#2}%
1168   \let\bbl@language@opts\@empty
1169   \ifx\babeloptionstrings\@undefined
1170     \let\bbl@opt@strings\@nnil
1171   \else
1172     \let\bbl@opt@strings\babeloptionstrings
1173   \fi
1174   \def\BabelStringsDefault{generic}
1175   \def\bbl@tempa{normal}
1176   \ifx\babeloptionmath\bbl@tempa
1177     \def\bbl@mathnormal{\noexpand\textormath}
1178   \fi
1179   \def\AfterBabelLanguage#1#2{}
1180   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi

```

```

1181 \let\bbl@afterlang\relax
1182 \def\bbl@opt@safe{BR}
1183 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
1184 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1185 \expandafter\newif\csname ifbbl@single\endcsname
1186 \chardef\bbl@bidimode\z@
1187 \fi

```

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

```

1188 \ifx\bbl@trace\@undefined
1189 \let\LdfInit\endinput
1190 \def\ProvidesLanguage#1{\endinput}
1191 \endinput\fi % Same line!

```

And continue.

9 Multiple languages

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

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.

```

1192 <<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.

```

1193 \def\bbl@version{<<version>>}}
1194 \def\bbl@date{<<date>>}}
1195 \def\adddialect#1#2{%
1196   \global\chardef#1#2\relax
1197   \bbl@usehooks{adddialect}{#1}{#2}}%
1198 \begingroup
1199   \count@#1\relax
1200   \def\bbl@elt##1##2##3###4{%
1201     \ifnum\count@=##2\relax
1202       \bbl@info{\string#1 = using hyphenrules for ##1\\%
1203         (\string\language\the\count@)}}%
1204       \def\bbl@elt####1####2####3####4{%
1205         \fi}%
1206       \bbl@cs{languages}%
1207     \endgroup}

```

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

```

1208 \def\bbl@fixname#1{%
1209   \begingroup
1210     \def\bbl@tempe{l@}%
1211     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1212     \bbl@tempd
1213     {\lowercase\expandafter{\bbl@tempd}}%
1214     {\uppercase\expandafter{\bbl@tempd}}%
1215     \@empty

```

```

1216      {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1217      \uppercase\expandafter{\bbl@tempd}}}%
1218      {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1219      \lowercase\expandafter{\bbl@tempd}}}%
1220      \@empty
1221      \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}}%
1222      \bbl@tempd
1223      \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1224      \def\bbl@iflanguage#1{%
1225      \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

1226 \def\bbl@bcpcase#1#2#3#4\@#5{%
1227   \ifx\@empty#3%
1228     \uppercase{\def#5{#1#2}}%
1229   \else
1230     \uppercase{\def#5{#1}}%
1231     \lowercase{\edef#5{#5#2#3#4}}%
1232   \fi}
1233 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1234   \let\bbl@bcp\relax
1235   \lowercase{\def\bbl@tempa{#1}}%
1236   \ifx\@empty#2%
1237     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1238   \else\ifx\@empty#3%
1239     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1240     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1241     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1242     {}%
1243     \ifx\bbl@bcp\relax
1244       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1245     \fi
1246   \else
1247     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1248     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1249     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1250     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1251     {}%
1252     \ifx\bbl@bcp\relax
1253       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1254       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1255       {}%
1256     \fi
1257     \ifx\bbl@bcp\relax
1258       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1259       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1260       {}%
1261     \fi
1262     \ifx\bbl@bcp\relax
1263       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1264     \fi
1265   \fi\fi}

```

```

1266 \let\bbl@autoload@options\@empty
1267 \let\bbl@initoload\relax
1268 \def\bbl@provide@locale{%
1269   \ifx\babelprovide\undefined
1270     \bbl@error{For a language to be defined on the fly 'base'\\%
1271               is not enough, and the whole package must be\\%
1272               loaded. Either delete the 'base' option or\\%
1273               request the languages explicitly}%
1274     {See the manual for further details.}%
1275   \fi
1276 % TODO. Option to search if loaded, with \LocaleForEach
1277 \let\bbl@auxname\language % Still necessary. TODO
1278 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1279   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1280 \ifbbl@bcp@allowed
1281   \expandafter\ifx\csname date\language\endcsname\relax
1282     \expandafter
1283     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1284     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1285       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1286       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1287       \expandafter\ifx\csname date\language\endcsname\relax
1288         \let\bbl@initoload\bbl@bcp
1289         \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1290         \let\bbl@initoload\relax
1291       \fi
1292       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1293     \fi
1294   \fi
1295 \fi
1296 \expandafter\ifx\csname date\language\endcsname\relax
1297   \IfFileExists{babel-\language.tex}%
1298   {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
1299   {}%
1300 \fi}

```

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

```

1301 \def\iflanguage#1{%
1302   \bbl@iflanguage{#1}%
1303   \ifnum\csname l@#1\endcsname=\language
1304     \expandafter\@firstoftwo
1305   \else
1306     \expandafter\@secondoftwo
1307   \fi}}

```

9.1 Selecting the language

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

```

1308 \let\bbl@select@type\z@
1309 \edef\selectlanguage{%
1310   \noexpand\protect

```

```
1311 \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`.

```
1312 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1313 \let\xstring\string
```

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

`\bbl@pop@language` But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's `aftergroup` mechanism to help us. The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

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

```
1314 \def\bbl@language@stack{}
```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

`\bbl@push@language` The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:

`\bbl@pop@language`

```
1315 \def\bbl@push@language{%
1316   \ifx\language\@undefined\else
1317     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1318   \fi}
```

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

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

```
1319 \def\bbl@pop@lang#1+#2\@{
1320   \edef\language{#1}%
1321   \xdef\bbl@language@stack{#2}}
```

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

```
1322 \let\bbl@ifrestoring\@secondoftwo
1323 \def\bbl@pop@language{%
1324   \expandafter\bbl@pop@lang\bbl@language@stack\@
1325   \let\bbl@ifrestoring\@firstoftwo
1326   \expandafter\bbl@set@language\expandafter{\language}%
1327   \let\bbl@ifrestoring\@secondoftwo}
```

Once the name of the previous language is retrieved from the stack, it is fed to `\bbl@set@language` to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of `\localeid`. This means `\l@...` will be reserved for hyphenation patterns (so that two locales can share the same rules).

```

1328 \chardef\localeid\z@
1329 \def\bbl@id@last{0} % No real need for a new counter
1330 \def\bbl@id@assign{%
1331   \bbl@ifunset{bbl@id@@\language}%
1332   {\count@bbl@id@last\relax
1333    \advance\count@ne
1334    \bbl@csarg\chardef{id@@\language}\count@
1335    \edef\bbl@id@last{\the\count@}%
1336    \ifcase\bbl@engine\or
1337      \directlua{
1338        Babel = Babel or {}
1339        Babel.locale_props = Babel.locale_props or {}
1340        Babel.locale_props[\bbl@id@last] = {}
1341        Babel.locale_props[\bbl@id@last].name = '\language'
1342      }%
1343    \fi}%
1344  }%
1345  \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

1346 \expandafter\def\csname selectlanguage \endcsname#1{%
1347   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
1348   \bbl@push@language
1349   \aftergroup\bbl@pop@language
1350   \bbl@set@language{#1}}

```

`\bbl@set@language` The macro `\bbl@set@language` takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of `\language`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\language` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards. We also write a command to change the current language in the auxiliary files.

```

1351 \def\BabelContentsFiles{toc,lof,lot}
1352 \def\bbl@set@language#1{% from selectlanguage, pop@
1353   % The old buggy way. Preserved for compatibility.
1354   \edef\language{%
1355     \ifnum\escapechar=\expandafter`\string#1\@empty
1356     \else\string#1\@empty\fi}%
1357   \ifcat\relax\noexpand#1%
1358     \expandafter\ifx\csname date\language\endcsname\relax
1359       \edef\language{#1}%
1360       \let\localename\language
1361     \else
1362       \bbl@info{Using '\string\language' instead of 'language' is%%
1363         deprecated. If what you want is to use a%%
1364         macro containing the actual locale, make%%
1365         sure it does not not match any language.%%
1366         Reported}%

```



```

1367 %                I'll\\%
1368 %                try to fix '\string\localename', but I cannot promise\\%
1369 %                anything. Reported}%
1370 \ifx\scantokens\undefined
1371 \def\localename{??}%
1372 \else
1373 \scantokens\expandafter{\expandafter
1374 \def\expandafter\localename\expandafter{\language}%
1375 \fi
1376 \fi
1377 \else
1378 \def\localename{#1}% This one has the correct catcodes
1379 \fi
1380 \select@language{\language}%
1381 % write to auxs
1382 \expandafter\ifx\csname date\language\endcsname\relax\else
1383 \if@files
1384 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1385 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1386 \fi
1387 \bbl@usehooks{write}{}}%
1388 \fi
1389 \fi}
1390 %
1391 \newif\ifbbl@bcpallowed
1392 \bbl@bcpallowedfalse
1393 \def\select@language#1{% from set@, babel@aux
1394 % set hymap
1395 \ifnum\bbl@hymapset=\@ccclv\chardef\bbl@hymapset4\relax\fi
1396 % set name
1397 \edef\language{#1}%
1398 \bbl@fixname\language
1399 % TODO. name@map must be here?
1400 \bbl@provide@locale
1401 \bbl@iflanguage\language{%
1402 \expandafter\ifx\csname date\language\endcsname\relax
1403 \bbl@error
1404 {Unknown language '\language'. Either you have\\%
1405 misspelled its name, it has not been installed,\\%
1406 or you requested it in a previous run. Fix its name,\\%
1407 install it or just rerun the file, respectively. In\\%
1408 some cases, you may need to remove the aux file}%
1409 {You may proceed, but expect wrong results}%
1410 \else
1411 % set type
1412 \let\bbl@select@type\z@
1413 \expandafter\bbl@switch\expandafter{\language}%
1414 \fi}}
1415 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1416 \select@language{#1}%
1417 \bbl@foreach\BabelContentsFiles{%
1418 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1419 \def\babel@toc#1#2{%
1420 \select@language{#1}}

```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring \TeX in a certain pre-defined state.

The name of the language is stored in the control sequence `\language`.

Then we have to *redefine* `\originalTeX` to compensate for the things that have been

activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\<lang>hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\<lang>hyphenmins` will be used.

```

1421 \newif\ifbbl@usedategroup
1422 \def\bbl@switch#1{% from select@, foreign@
1423 % make sure there is info for the language if so requested
1424 \bbl@ensureinfo{#1}%
1425 % restore
1426 \originalTeX
1427 \expandafter\def\expandafter\originalTeX\expandafter{%
1428 \csname noextras#1\endcsname
1429 \let\originalTeX\empty
1430 \babel@beginsave}%
1431 \bbl@usehooks{afterreset}{}%
1432 \languageshorthands{none}%
1433 % set the locale id
1434 \bbl@id@assign
1435 % switch captions, date
1436 % No text is supposed to be added here, so we remove any
1437 % spurious spaces.
1438 \bbl@bsphack
1439 \ifcase\bbl@select@type
1440 \csname captions#1\endcsname\relax
1441 \csname date#1\endcsname\relax
1442 \else
1443 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1444 \ifin@
1445 \csname captions#1\endcsname\relax
1446 \fi
1447 \bbl@xin@{,date,}{, \bbl@select@opts,}%
1448 \ifin@ % if \foreign... within \<lang>date
1449 \csname date#1\endcsname\relax
1450 \fi
1451 \fi
1452 \bbl@esphack
1453 % switch extras
1454 \bbl@usehooks{beforeextras}{}%
1455 \csname extras#1\endcsname\relax
1456 \bbl@usehooks{afterextras}{}%
1457 % > babel-ensure
1458 % > babel-sh-<short>
1459 % > babel-bidi
1460 % > babel-fontspec
1461 % hyphenation - case mapping
1462 \ifcase\bbl@opt@hyphenmap\or
1463 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1464 \ifnum\bbl@hymapsel>4\else
1465 \csname\language @bbl@hyphenmap\endcsname
1466 \fi
1467 \chardef\bbl@opt@hyphenmap\z@
1468 \else

```

```

1469 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1470 \csname\language @bbl@hyphenmap\endcsname
1471 \fi
1472 \fi
1473 \global\let\bbl@hymapsel\@cclv
1474 % hyphenation - patterns
1475 \bbl@patterns{#1}%
1476 % hyphenation - mins
1477 \babel@savevariable\lefthyphenmin
1478 \babel@savevariable\righthyphenmin
1479 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1480 \set@hyphenmins\tw@\thr@\relax
1481 \else
1482 \expandafter\expandafter\expandafter\set@hyphenmins
1483 \csname #1hyphenmins\endcsname\relax
1484 \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.

```

1485 \long\def\otherlanguage#1{%
1486 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
1487 \csname selectlanguage \endcsname{#1}%
1488 \ignorespaces}

```

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

```

1489 \long\def\endotherlanguage{%
1490 \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`.

```

1491 \expandafter\def\csname otherlanguage*\endcsname{%
1492 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1493 \def\bbl@otherlanguage@s[#1]#2{%
1494 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1495 \def\bbl@select@opts{#1}%
1496 \foreign@language{#2}}

```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```

1497 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras<lang>` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`. `\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and

therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op. (3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in vmode and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into hmode with the surrounding lang, and with `\foreignlanguage*` with the new lang.

```

1498 \providecommand\bbl@beforeforeign{}
1499 \edef\foreignlanguage{%
1500   \noexpand\protect
1501   \expandafter\noexpand\csname foreignlanguage \endcsname}
1502 \expandafter\def\csname foreignlanguage \endcsname{%
1503   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1504 \providecommand\bbl@foreign@x[3][{}]{%
1505   \begingroup
1506     \def\bbl@select@opts{#1}%
1507     \let\BabelText\@firstofone
1508     \bbl@beforeforeign
1509     \foreign@language{#2}%
1510     \bbl@usehooks{foreign}{}%
1511     \BabelText{#3}% Now in horizontal mode!
1512   \endgroup}
1513 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1514   \begingroup
1515     {\par}%
1516     \let\BabelText\@firstofone
1517     \foreign@language{#1}%
1518     \bbl@usehooks{foreign*}{}%
1519     \bbl@dirparastext
1520     \BabelText{#2}% Still in vertical mode!
1521     {\par}%
1522   \endgroup}

```

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

```

1523 \def\foreign@language#1{%
1524   % set name
1525   \edef\language#1}%
1526   \ifbbl@usedategroup
1527     \bbl@add\bbl@select@opts{,date,}%
1528     \bbl@usedategroupfalse
1529   \fi
1530   \bbl@fixname\language
1531   % TODO. name@map here?
1532   \bbl@provide@locale
1533   \bbl@iflanguage\language%
1534     \expandafter\ifx\csname date\language\endcsname\relax
1535       \bbl@warning % TODO - why a warning, not an error?
1536       {Unknown language `#1'. Either you have\\%
1537         misspelled its name, it has not been installed,\\%

```

```

1538         or you requested it in a previous run. Fix its name,\%
1539         install it or just rerun the file, respectively. In\%
1540         some cases, you may need to remove the aux file.\%
1541         I'll proceed, but expect wrong results.\%
1542     Reported}%
1543 \fi
1544 % set type
1545 \let\bbl@select@type\@ne
1546 \expandafter\bbl@switch\expandafter{\language\name}}

```

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

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

```

1547 \let\bbl@hyphlist\@empty
1548 \let\bbl@hyphenation@\relax
1549 \let\bbl@pttnlist\@empty
1550 \let\bbl@patterns@\relax
1551 \let\bbl@hymapsel=\@ccclv
1552 \def\bbl@patterns#1{%
1553   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1554     \csname l@#1\endcsname
1555     \edef\bbl@tempa{#1}%
1556   \else
1557     \csname l@#1:\f@encoding\endcsname
1558     \edef\bbl@tempa{#1:\f@encoding}%
1559   \fi
1560   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1561   % > luatex
1562   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
1563     \begingroup
1564       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1565       \ifin@else
1566         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1567         \hyphenation{%
1568           \bbl@hyphenation@
1569           \@ifundefined{bbl@hyphenation@#1}%
1570             \@empty
1571             {\space\csname bbl@hyphenation@#1\endcsname}}%
1572         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1573       \fi
1574     \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*.

```

1575 \def\hyphenrules#1{%
1576   \edef\bbl@tempf{#1}%
1577   \bbl@fixname\bbl@tempf
1578   \bbl@iflanguage\bbl@tempf{%
1579     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%

```

```

1580 \languageshortands{none}%
1581 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1582 \set@hyphenmins\tw@\thr@@\relax
1583 \else
1584 \expandafter\expandafter\expandafter\set@hyphenmins
1585 \csname\bbl@tempf hyphenmins\endcsname\relax
1586 \fi}}
1587 \let\endhyphenrules\@empty

```

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

```

1588 \def\providehyphenmins#1#2{%
1589 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1590 \@namedef{#1hyphenmins}{#2}%
1591 \fi}

```

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

```

1592 \def\set@hyphenmins#1#2{%
1593 \lefthyphenmin#1\relax
1594 \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.

```

1595 \ifx\ProvidesFile\@undefined
1596 \def\ProvidesLanguage#1[#2 #3 #4]{%
1597 \wlog{Language: #1 #4 #3 <#2>}%
1598 }
1599 \else
1600 \def\ProvidesLanguage#1{%
1601 \begingroup
1602 \catcode\ 10 %
1603 \@makeother\/%
1604 \@ifnextchar[%]
1605 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1606 \def\@provideslanguage#1[#2]{%
1607 \wlog{Language: #1 #2}%
1608 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1609 \endgroup}
1610 \fi

```

`\originalTeX` The macro `\originalTeX` should be known to \TeX at this moment. As it has to be expandable we `\let` it to `\@empty` instead of `\relax`.

```

1611 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi

```

Because this part of the code can be included in a format, we make sure that the macro which initializes the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

1612 \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’:

```

1613 \providecommand\setlocale{%

```

```

1614 \bbl@error
1615 {Not yet available}%
1616 {Find an armchair, sit down and wait}}
1617 \let\uselocale\setlocale
1618 \let\locale\setlocale
1619 \let\selectlocale\setlocale
1620 \let\localename\setlocale
1621 \let\textlocale\setlocale
1622 \let\textlanguage\setlocale
1623 \let\language\setlocale

```

9.2 Errors

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

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

```

1624 \edef\bbl@nulllanguage{\string\language=0}
1625 \ifx\PackageError\@undefined % TODO. Move to Plain
1626 \def\bbl@error#1#2{%
1627 \begingroup
1628 \newlinechar=`^^J
1629 \def\{^^J(babel) }%
1630 \errhelp{#2}\errmessage{\{#1}%
1631 \endgroup}
1632 \def\bbl@warning#1{%
1633 \begingroup
1634 \newlinechar=`^^J
1635 \def\{^^J(babel) }%
1636 \message{\{#1}%
1637 \endgroup}
1638 \let\bbl@infowarn\bbl@warning
1639 \def\bbl@info#1{%
1640 \begingroup
1641 \newlinechar=`^^J
1642 \def\{^^J}%
1643 \wlog{#1}%
1644 \endgroup}
1645 \fi
1646 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1647 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1648 \global\@namedef{#2}{\textbf{?#1?}}%
1649 \@nameuse{#2}%
1650 \bbl@warning{%
1651 \@backslashchar#2 not set. Please, define it\\%
1652 after the language has been loaded (typically\\%
1653 in the preamble) with something like:\\%
1654 \string\renewcommand\@backslashchar#2{..}\\%
1655 Reported}}

```

```

1656 \def\bbl@tentative{\protect\bbl@tentative@i}
1657 \def\bbl@tentative@i#1{%
1658   \bbl@warning{%
1659     Some functions for '#1' are tentative.\\%
1660     They might not work as expected and their behavior\\%
1661     could change in the future.\\%
1662     Reported}}
1663 \def\@nolanerr#1{%
1664   \bbl@error
1665   {You haven't defined the language #1\space yet.\\%
1666     Perhaps you misspelled it or your installation\\%
1667     is not complete}%
1668   {Your command will be ignored, type <return> to proceed}}
1669 \def\@nopatterns#1{%
1670   \bbl@warning
1671   {No hyphenation patterns were preloaded for\\%
1672     the language '#1' into the format.\\%
1673     Please, configure your TeX system to add them and\\%
1674     rebuild the format. Now I will use the patterns\\%
1675     preloaded for \bbl@nulllanguage\space instead}}
1676 \let\bbl@usehooks\@gobbletwo
1677 \ifx\bbl@onlyswitch\@empty\endinput\fi
1678 % Here ended switch.def

Here ended switch.def.

1679 \ifx\directlua\@undefined\else
1680   \ifx\bbl@luapatterns\@undefined
1681     \input luababel.def
1682   \fi
1683 \fi
1684 <<Basic macros>>
1685 \bbl@trace{Compatibility with language.def}
1686 \ifx\bbl@languages\@undefined
1687   \ifx\directlua\@undefined
1688     \openin1 = language.def % TODO. Remove hardcoded number
1689     \ifeof1
1690       \closein1
1691       \message{I couldn't find the file language.def}
1692     \else
1693       \closein1
1694       \begingroup
1695         \def\addlanguage#1#2#3#4#5{%
1696           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1697             \global\expandafter\let\csname l@#1\endcsname
1698             \csname lang@#1\endcsname
1699           \fi}%
1700         \def\uselanguage#1{%
1701           \input language.def
1702         \endgroup
1703       \fi
1704     \fi
1705   \chardef\l@english\z@
1706 \fi

```

\addto It takes two arguments, a *<control sequence>* and TeX-code to be added to the *<control sequence>*.
If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to \relax, in which case a circular definition results. The net

result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

1707 \def\addto#1#2{%
1708   \ifx#1\undefined
1709     \def#1{#2}%
1710   \else
1711     \ifx#1\relax
1712       \def#1{#2}%
1713     \else
1714       {\toks@\expandafter{#1#2}%
1715        \xdef#1{the\toks@}}%
1716   \fi
1717 \fi}

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. `TODO`. Always used with additional expansions. Move them here? Move the macro to basic?

```

1718 \def\bbl@withactive#1#2{%
1719   \begingroup
1720   \lccode`~=`#2\relax
1721   \lowercase{\endgroup#1~}}

```

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

```

1722 \def\bbl@redefine#1{%
1723   \edef\bbl@tempa{\bbl@stripslash#1}%
1724   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1725   \expandafter\def\csname\bbl@tempa\endcsname{
1726   \@onlypreamble\bbl@redefine

```

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

```

1727 \def\bbl@redefine@long#1{%
1728   \edef\bbl@tempa{\bbl@stripslash#1}%
1729   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1730   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1731   \@onlypreamble\bbl@redefine@long

```

`\bbl@redefineroobust` For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo_`. So it is necessary to check whether `\foo_` exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define `\foo_`.

```

1732 \def\bbl@redefineroobust#1{%
1733   \edef\bbl@tempa{\bbl@stripslash#1}%
1734   \bbl@ifunset{\bbl@tempa\space}%
1735   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1736    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1737   {\bbl@exp{\let\<org@\bbl@tempa\<\bbl@tempa\space>}}}%
1738   \@namedef{\bbl@tempa\space}}
1739 \@onlypreamble\bbl@redefineroobust

```

9.3 Hooks

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

```

1740 \bbl@trace{Hooks}
1741 \newcommand\AddBabelHook[3][]{%
1742   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1743   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1744   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1745   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1746     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1747     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1748   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1749 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1750 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1751 \def\bbl@usehooks#1#2{%
1752   \def\bbl@elt##1{%
1753     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1754     \bbl@cs{ev@#1@}%
1755     \ifx\language@undefined\else % Test required for Plain (?)
1756       \def\bbl@elt##1{%
1757         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1758         \bbl@cl{ev@#1}%
1759       \fi}

```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```

1760 \def\bbl@evargs{,% <- don't delete this comma
1761   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1762   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1763   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1764   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1765   beforestart=0,language=2}

```

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

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

```

1766 \bbl@trace{Defining babelensure}
1767 \newcommand\babelensure[2][]{% TODO - revise test files
1768   \AddBabelHook{babel-ensure}{afterextras}{%
1769     \ifcase\bbl@select@type
1770       \bbl@cl{e}%
1771     \fi}%
1772   \beginngroup
1773     \let\bbl@ens@include\@empty
1774     \let\bbl@ens@exclude\@empty
1775     \def\bbl@ens@fontenc{\relax}%

```

```

1776 \def\bbl@tempb##1{%
1777   \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1778 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1779 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1780 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1781 \def\bbl@tempc{\bbl@ensure}%
1782 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1783   \expandafter{\bbl@ens@include}}%
1784 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1785   \expandafter{\bbl@ens@exclude}}%
1786 \toks@\expandafter{\bbl@tempc}%
1787 \bbl@exp{%
1788 \endgroup
1789 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1790 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1791 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1792   \ifx##1\undefined % 3.32 - Don't assume the macro exists
1793     \edef##1{\noexpand\bbl@nocaption
1794       {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}%
1795   \fi
1796   \ifx##1\@empty\else
1797     \in@{##1}{#2}%
1798     \ifin@ \else
1799       \bbl@ifunset{\bbl@ensure@\language\language}%
1800       {\bbl@exp{%
1801         \\DeclareRobustCommand\bbl@ensure@<\language>[1]{%
1802           \\foreignlanguage{\language}%
1803           {\ifx\relax#3\else
1804             \\fontencoding{#3}\\selectfont
1805             \fi
1806             #####1}}}%
1807         }%
1808         \toks@\expandafter{##1}%
1809         \edef##1{%
1810           \bbl@csarg\noexpand{ensure@\language}%
1811           {\the\toks@}}%
1812       \fi
1813       \expandafter\bbl@tempb
1814     \fi}%
1815 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1816 \def\bbl@tempa##1{% elt for include list
1817   \ifx##1\@empty\else
1818     \bbl@csarg\in@{ensure@\language\language}\expandafter{\bbl@tempb}
1819     \ifin@ \else
1820       \bbl@tempb##1\@empty
1821     \fi
1822     \expandafter\bbl@tempa
1823   \fi}%
1824 \bbl@tempa#1\@empty}
1825 \def\bbl@captionslist{%
1826 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1827 \contentsname\listfigurename\listtablename\indexname\figurename
1828 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1829 \alsoname\proofname\glossaryname}

```

9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```
1830 \bbl@trace{Macros for setting language files up}
1831 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1832   \let\bbl@screset@empty
1833   \let\BabelStrings\bbl@opt@string
1834   \let\BabelOptions@empty
1835   \let\BabelLanguages\relax
1836   \ifx\originalTeX\@undefined
1837     \let\originalTeX@empty
1838   \else
1839     \originalTeX
1840   \fi}
1841 \def\LdfInit#1#2{%
1842   \chardef\atcatcode=\catcode`\@
1843   \catcode`\@=11\relax
1844   \chardef\eqcatcode=\catcode`\=
1845   \catcode`\==12\relax
1846   \expandafter\if\expandafter\@backslashchar
1847     \expandafter\@car\string#2\@nil
1848   \ifx#2\@undefined\else
1849     \ldf@quit{#1}%
1850   \fi
1851 \else
1852   \expandafter\ifx\csname#2\endcsname\relax\else
1853     \ldf@quit{#1}%
1854   \fi
1855 \fi
1856 \bbl@ldfinit}
```

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

```
1857 \def\ldf@quit#1{%
1858   \expandafter\main@language\expandafter{#1}%
1859   \catcode`\@=\atcatcode \let\atcatcode\relax
1860   \catcode`\==\eqcatcode \let\eqcatcode\relax
1861   \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.

```
1862 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1863   \bbl@afterlang
1864   \let\bbl@afterlang\relax
1865   \let\BabelModifiers\relax
1866   \let\bbl@screset\relax}%
1867 \def\ldf@finish#1{%
1868   \ifx\loadlocalcfg\undefined\else % For LaTeX 209
1869     \loadlocalcfg{#1}%
1870   \fi
1871   \bbl@afterldf{#1}%
1872   \expandafter\main@language\expandafter{#1}%
1873   \catcode`\@=\atcatcode \let\atcatcode\relax
1874   \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 .

```
1875 \@onlypreamble\LdfInit
1876 \@onlypreamble\ldf@quit
1877 \@onlypreamble\ldf@finish
```

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

```
1878 \def\main@language#1{%
1879   \def\bbl@main@language{#1}%
1880   \let\languagename\bbl@main@language % TODO. Set localename
1881   \bbl@id@assign
1882   \bbl@patterns{\languagename}}
```

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

```
1883 \def\bbl@beforestart{%
1884   \bbl@usehooks{beforestart}{}%
1885   \global\let\bbl@beforestart\relax}
1886 \AtBeginDocument{%
1887   \@nameuse{bbl@beforestart}%
1888   \if@filesw
1889     \providecommand\babel@aux[2]{}%
1890     \immediate\write\@mainaux{%
1891       \string\providecommand\string\babel@aux[2]{}%
1892       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1893   \fi
1894   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1895   \ifbbl@single % must go after the line above.
1896     \renewcommand\selectlanguage[1]{}%
1897     \renewcommand\foreignlanguage[2]{#2}%
1898     \global\let\babel@aux\@gobbletwo % Also as flag
1899   \fi
1900   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```

1901 \def\select@language@x#1{%
1902   \ifcase\bbbl@select@type
1903     \bbbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1904   \else
1905     \select@language{#1}%
1906   \fi}

```

9.5 Shorthands

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

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

```

1907 \bbbl@trace{Shorhands}
1908 \def\bbbl@add@special#1{% 1:a macro like \", \?, etc.
1909   \bbbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1910   \bbbl@ifunset{@sanitize}{\bbbl@add\@sanitize{\@makeother#1}}%
1911   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1912     \begingroup
1913       \catcode`#1\active
1914       \nfss@catcodes
1915       \ifnum\catcode`#1=\active
1916         \endgroup
1917         \bbbl@add\nfss@catcodes{\@makeother#1}%
1918       \else
1919         \endgroup
1920       \fi
1921   \fi}

```

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

```

1922 \def\bbbl@remove@special#1{%
1923   \begingroup
1924     \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1925       \else\noexpand##1\noexpand##2\fi}%
1926     \def\do{\x\do}%
1927     \def\@makeother{\x\@makeother}%
1928     \edef\x{\endgroup
1929       \def\noexpand\dospecials{\dospecials}%
1930       \expandafter\ifx\cscname @sanitize\endcscname\relax\else
1931         \def\noexpand\@sanitize{\@sanitize}%
1932       \fi}%
1933     \x}

```

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

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

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

```

1934 \def\bbl@active@def#1#2#3#4{%
1935   \@namedef{#3#1}{%
1936     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1937       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1938     \else
1939       \bbl@afterfi\csname#2@sh@#1\endcsname
1940     \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.

```

1941 \long\@namedef{#3@arg#1}##1{%
1942   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1943     \bbl@afterelse\csname#4#1\endcsname##1%
1944   \else
1945     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1946   \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.

```

1947 \def\initiate@active@char#1#2#3{%
1948   \bbl@ifunset{active@char\string#1}%
1949   {\bbl@withactive
1950     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1951   {}}

```

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`).

```

1952 \def\@initiate@active@char#1#2#3{%
1953   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1954   \ifx#1\@undefined
1955     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1956   \else
1957     \bbl@csarg\let{oridef@#2}#1%
1958     \bbl@csarg\edef{oridef@#2}{%
1959       \let\noexpand#1%
1960       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1961   \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*").

```

1962 \ifx#1#3\relax
1963   \expandafter\let\csname normal@char#2\endcsname#3%
1964 \else
1965   \bbl@info{Making #2 an active character}%
1966   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1967   \@namedef{normal@char#2}{%
1968     \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1969   \else
1970     \@namedef{normal@char#2}{#3}%
1971   \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).

```

1972   \bbl@restoreactive{#2}%
1973   \AtBeginDocument{%
1974     \catcode`#2\active
1975     \if@filesw
1976       \immediate\write\@mainaux{\catcode`\string#2\active}%
1977     \fi}%
1978   \expandafter\bbl@add@special\csname#2\endcsname
1979   \catcode`#2\active
1980 \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⟩`).

```

1981 \let\bbl@tempa\@firstoftwo
1982 \if\string^#2%
1983   \def\bbl@tempa{\noexpand\textormath}%
1984 \else
1985   \ifx\bbl@mathnormal\@undefined\else
1986     \let\bbl@tempa\bbl@mathnormal
1987   \fi
1988 \fi
1989 \expandafter\edef\csname active@char#2\endcsname{%
1990   \bbl@tempa
1991     {\noexpand\if@safe@actives
1992       \noexpand\expandafter
1993       \expandafter\noexpand\csname normal@char#2\endcsname
1994     \noexpand\else
1995       \noexpand\expandafter
1996       \expandafter\noexpand\csname bbl@doactive#2\endcsname
1997     \noexpand\fi}%
1998   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1999 \bbl@csarg\edef{doactive#2}{%
2000   \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!).

```
2001 \bbl@csarg\edef{active@#2}{%
2002   \noexpand\active@prefix\noexpand#1%
2003   \expandafter\noexpand\csname active@char#2\endcsname}%
2004 \bbl@csarg\edef{normal@#2}{%
2005   \noexpand\active@prefix\noexpand#1%
2006   \expandafter\noexpand\csname normal@char#2\endcsname}%
2007 \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.

```
2008 \bbl@active@def#2\user@group{user@active}{language@active}%
2009 \bbl@active@def#2\language@group{language@active}{system@active}%
2010 \bbl@active@def#2\system@group{system@active}{normal@char}%
```

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

```
2011 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2012   {\expandafter\noexpand\csname normal@char#2\endcsname}%
2013 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
2014   {\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.

```
2015 \if\string'#2%
2016   \let\prim@s\bbl@prim@s
2017   \let\active@math@prime#1%
2018 \fi
2019 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
2020 <<(*More package options)>> ≡
2021 \DeclareOption{math=active}{}
2022 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2023 <</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*.

```
2024 \@ifpackagewith{babel}{KeepShorthandsActive}%
2025   {\let\bbl@restoreactive\@gobble}%
2026   {\def\bbl@restoreactive#1{%
2027     \bbl@exp{%
2028       \\\AfterBabelLanguage\\CurrentOption
2029       {\catcode`#1=\the\catcode`#1\relax}%
2030       \\\AtEndOfPackage
2031       {\catcode`#1=\the\catcode`#1\relax}}}%
2032   \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.

```
2033 \def\bbl@sh@select#1#2{%
2034   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2035     \bbl@afterelse\bbl@scndcs
2036   \else
2037     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2038   \fi}
```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protects` the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar`: (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```
2039 \begingroup
2040 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2041   {\gdef\active@prefix#1{%
2042     \ifx\protect\@typeset@protect
2043       \else
2044         \ifx\protect\@unexpandable@protect
2045           \noexpand#1%
2046         \else
2047           \protect#1%
2048         \fi
2049       \expandafter\@gobble
2050     \fi}}
2051   {\gdef\active@prefix#1{%
2052     \ifincsname
2053       \string#1%
2054       \expandafter\@gobble
2055     \else
2056       \ifx\protect\@typeset@protect
2057         \else
2058           \ifx\protect\@unexpandable@protect
2059             \noexpand#1%
2060           \else
2061             \protect#1%
2062           \fi
2063         \expandafter\expandafter\expandafter\@gobble
2064       \fi
2065     \fi}}
2066 \endgroup
```

`\if@safe@actives` In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char` (*char*).

```
2067 \newif\if@safe@actives
2068 \@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.

```

2069 \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
\bbl@deactivate 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.

2070 \def\bbl@activate#1{%
2071   \bbl@withactive{\expandafter\let\expandafter}#1%
2072   \csname bbl@active@\string#1\endcsname}
2073 \def\bbl@deactivate#1{%
2074   \bbl@withactive{\expandafter\let\expandafter}#1%
2075   \csname bbl@normal@\string#1\endcsname}

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
2076 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2077 \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.

2078 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2079 \def\@decl@short#1#2#3\@nil#4{%
2080   \def\bbl@tempa{#3}%
2081   \ifx\bbl@tempa\@empty
2082     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2083     \bbl@ifunset{#1@sh@\string#2@}{}%
2084     {\def\bbl@tempa{#4}%
2085       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2086       \else
2087         \bbl@info
2088         {Redefining #1 shorthand \string#2\%
2089         in language \CurrentOption}%
2090       \fi}%
2091     \@namedef{#1@sh@\string#2@}{#4}%
2092   \else
2093     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2094     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2095     {\def\bbl@tempa{#4}%
2096       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2097       \else
2098         \bbl@info
2099         {Redefining #1 shorthand \string#2\string#3\%
2100         in language \CurrentOption}%
2101       \fi}%
2102     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2103   \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.

2104 \def\textormath{%
2105   \ifmmode
2106     \expandafter\@secondoftwo

```

```

2107 \else
2108 \expandafter\@firstoftwo
2109 \fi}

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

2110 \def\user@group{user}
2111 \def\language@group{english} % TODO. I don't like defaults
2112 \def\system@group{system}

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

2113 \def\useshorthands{%
2114 \ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2115 \def\bbl@usesh@s#1{%
2116 \bbl@usesh@x
2117 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2118 {#1}}
2119 \def\bbl@usesh@x#1#2{%
2120 \bbl@ifshorthand{#2}%
2121 {\def\user@group{user}%
2122 \initiate@active@char{#2}%
2123 #1%
2124 \bbl@activate{#2}}%
2125 {\bbl@error
2126 {Cannot declare a shorthand turned off (\string#2)}
2127 {Sorry, but you cannot use shorthands which have been\\
2128 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.

2129 \def\user@language@group{user@\language@group}
2130 \def\bbl@set@user@generic#1#2{%
2131 \bbl@ifunset{user@generic@active#1}%
2132 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2133 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2134 \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2135 \expandafter\noexpand\csname normal@char#1\endcsname}%
2136 \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2137 \expandafter\noexpand\csname user@active#1\endcsname}}%
2138 \@empty}
2139 \newcommand\defineshorthand[3][user]{%
2140 \edef\bbl@tempa{\zap@space#1 \@empty}%
2141 \bbl@for\bbl@tempb\bbl@tempa{%
2142 \if*\expandafter\@car\bbl@tempb\@nil
2143 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2144 \@expandtwoargs
2145 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2146 \fi
2147 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```
2148 \def\languageshorthands#1{\def\language@group{#1}}
```

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

```
2149 \def\aliasshorthand#1#2{%
2150   \bbl@ifshorthand{#2}%
2151   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2152     \ifx\document\@notprerr
2153       \@notshorthand{#2}%
2154     \else
2155       \initiate@active@char{#2}%
2156       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2157         \csname active@char\string#1\endcsname
2158       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2159         \csname normal@char\string#1\endcsname
2160       \bbl@activate{#2}%
2161     \fi
2162   \fi}%
2163   {\bbl@error
2164     {Cannot declare a shorthand turned off (\string#2)}
2165     {Sorry, but you cannot use shorthands which have been\\%
2166       turned off in the package options}}}
```

`\@notshorthand`

```
2167 \def\@notshorthand#1{%
2168   \bbl@error{%
2169     The character '\string #1' should be made a shorthand character;\\%
2170     add the command \string\usesshorthands\string{#1\string} to
2171     the preamble.\\%
2172     I will ignore your instruction}%
2173   {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.

```
2174 \newcommand*\shorthandon[1]{\bbl@switch@sh@ne#1\@nnil}
2175 \DeclareRobustCommand*\shorthandoff{%
2176   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2177 \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.

```
2178 \def\bbl@switch@sh#1#2{%
2179   \ifx#2\@nnil\else
2180     \bbl@ifunset{\bbl@active@\string#2}%
```

```

2181      {\bbl@error
2182        {I cannot switch '\string#2' on or off--not a shorthand}%
2183        {This character is not a shorthand. Maybe you made\\
2184          a typing mistake? I will ignore your instruction}}}%
2185      {\ifcase#1%
2186        \catcode`#212\relax
2187        \or
2188        \catcode`#2\active
2189        \or
2190        \csname bbl@oricat@\string#2\endcsname
2191        \csname bbl@oridef@\string#2\endcsname
2192        \fi}%
2193      \bbl@afterfi\bbl@switch@sh#1%
2194    \fi}

```

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

```

2195 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2196 \def\bbl@putsh#1{%
2197   \bbl@ifunset{\bbl@active@\string#1}%
2198   {\bbl@putsh@i#1\@empty\@nnil}%
2199   {\csname bbl@active@\string#1\endcsname}}
2200 \def\bbl@putsh@i#1#2\@nnil{%
2201   \csname\language@group @sh@\string#1@%
2202     \ifx\@empty#2\else\string#2\fi\endcsname}
2203 \ifx\bbl@opt@shorthands\@nnil\else
2204   \let\bbl@s@initiate@active@char\initiate@active@char
2205   \def\initiate@active@char#1{%
2206     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2207   \let\bbl@s@switch@sh\bbl@switch@sh
2208   \def\bbl@switch@sh#1#2{%
2209     \ifx#2\@nnil\else
2210       \bbl@afterfi
2211       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2212     \fi}
2213   \let\bbl@s@activate\bbl@activate
2214   \def\bbl@activate#1{%
2215     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2216   \let\bbl@s@deactivate\bbl@deactivate
2217   \def\bbl@deactivate#1{%
2218     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2219 \fi

```

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

```

2220 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

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

```

2221 \def\bbl@prim@s{%
2222   \prime\futurelet\@let@token\bbl@pr@m@s}
2223 \def\bbl@if@primes#1#2{%
2224   \ifx#1\@let@token
2225     \expandafter\@firstoftwo
2226   \else\ifx#2\@let@token
2227     \bbl@afterelse\expandafter\@firstoftwo

```

```

2228 \else
2229 \bbl@afterfi\expandafter\@secondoftwo
2230 \fi\fi}
2231 \begingroup
2232 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2233 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\'
2234 \lowercase{%
2235 \gdef\bbl@pr@m@s{%
2236 \bbl@if@primes"%
2237 \pr@@s
2238 {\bbl@if@primes*\pr@@t\egroup}}
2239 \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).

```

2240 \initiate@active@char{~}
2241 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2242 \bbl@activate{~}

```

\OT1dpos The position of the double quote character is different for the OT1 and T1 encodings. It will
\T1dpos later be selected using the `\f@encoding` macro. Therefore we define two macros here to store the position of the character in these encodings.

```

2243 \expandafter\def\csname OT1dpos\endcsname{127}
2244 \expandafter\def\csname T1dpos\endcsname{4}

```

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

```

2245 \ifx\f@encoding\@undefined
2246 \def\f@encoding{OT1}
2247 \fi

```

9.6 Language attributes

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

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

```

2248 \bbl@trace{Language attributes}
2249 \newcommand\languageattribute[2]{%
2250 \def\bbl@tempc{#1}%
2251 \bbl@fixname\bbl@tempc
2252 \bbl@iflanguage\bbl@tempc{%
2253 \bbl@vforeach{#2}{%

```

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

```

2254 \ifx\bbl@known@attrs\@undefined
2255 \in@false
2256 \else
2257 \bbl@xin@{\bbl@tempc-##1,}{,\bbl@known@attrs,}%

```

```

2258     \fi
2259     \ifin@
2260         \bbl@warning{%
2261             You have more than once selected the attribute '##1'\%
2262             for language #1. Reported}%
2263     \else

```

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

```

2264         \bbl@exp{%
2265             \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2266         \edef\bbl@tempa{\bbl@tempc-##1}%
2267         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2268         {\csname\bbl@tempc @attr##1\endcsname}%
2269         {\@attrerr{\bbl@tempc}{##1}}%
2270     \fi}}
2271 \@onlypreamble\languageattribute

```

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

```

2272 \newcommand*{\@attrerr}[2]{%
2273     \bbl@error
2274     {The attribute #2 is unknown for language #1.}%
2275     {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}`.

```

2276 \def\bbl@declare@ttribute#1#2#3{%
2277     \bbl@xin@{, #2, }{, \BabelModifiers,}%
2278     \ifin@
2279         \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2280     \fi
2281     \bbl@add@list\bbl@attributes{#1-#2}%
2282     \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.

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```

2283 \def\bbl@ifattributeset#1#2#3#4{%
2284     \ifx\bbl@known@attribs\undefined
2285         \in@false
2286     \else
2287         \bbl@xin@{, #1-#2, }{, \bbl@known@attribs,}%
2288     \fi
2289     \ifin@
2290         \bbl@afterelse#3%
2291     \else

```



```

2292 \bbl@afterfi#4%
2293 \fi
2294 }

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

2295 \def\bbl@ifknown@ttrib#1#2{%
2296 \let\bbl@tempa\@secondoftwo
2297 \bbl@loopx\bbl@tempb{#2}{%
2298 \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
2299 \ifin@
2300 \let\bbl@tempa\@firstoftwo
2301 \else
2302 \fi}%
2303 \bbl@tempa
2304 }

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

2305 \def\bbl@clear@ttribs{%
2306 \ifx\bbl@attributes\undefined\else
2307 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2308 \expandafter\bbl@clear@ttrib\bbl@tempa.
2309 }%
2310 \let\bbl@attributes\undefined
2311 \fi}
2312 \def\bbl@clear@ttrib#1-#2.{%
2313 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2314 \AtBeginDocument{\bbl@clear@ttribs}

```

9.7 Support for saving macro definitions

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

```

\babel@savecnt The initialization of a new save cycle: reset the counter to zero.
\babel@beginsave
2315 \bbl@trace{Macros for saving definitions}
2316 \def\babel@beginsave{\babel@savecnt\z@}

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

2317 \newcount\babel@savecnt
2318 \babel@beginsave

\babel@save The macro \babel@save<cname> saves the current meaning of the control sequence
\babel@savevariable <cname> to \originalTeX31. To do this, we let the current meaning to a temporary control
sequence, the restore commands are appended to \originalTeX and the counter is

```

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

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

```

2319 \def\babel@save#1{%
2320   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2321   \toks@\expandafter{\originalTeX\let#1=}%
2322   \bbl@exp{%
2323     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2324   \advance\babel@savecnt@ne}
2325 \def\babel@savevariable#1{%
2326   \toks@\expandafter{\originalTeX #1=}%
2327   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```

2328 \def\bbl@frenchspacing{%
2329   \ifnum\the\sfcodes\.=\@m
2330     \let\bbl@nonfrenchspacing\relax
2331   \else
2332     \frenchspacing
2333     \let\bbl@nonfrenchspacing\nonfrenchspacing
2334   \fi}
2335 \let\bbl@nonfrenchspacing\nonfrenchspacing
2336 %
2337 \let\bbl@elt\relax
2338 \edef\bbl@fs@chars{%
2339   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2340   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2341   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
```

9.8 Short tags

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

```

2342 \bbl@trace{Short tags}
2343 \def\babeltags#1{%
2344   \edef\bbl@tempa{\zap@space#1 \@empty}%
2345   \def\bbl@tempb##1=##2\@{ }%
2346   \edef\bbl@tempc{%
2347     \noexpand\newcommand
2348     \expandafter\noexpand\csname ##1\endcsname{%
2349       \noexpand\protect
2350       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2351     \noexpand\newcommand
2352     \expandafter\noexpand\csname text##1\endcsname{%
2353       \noexpand\foreignlanguage{##2}}
2354   \bbl@tempc}%
2355   \bbl@for\bbl@tempa\bbl@tempa{%
2356     \expandafter\bbl@tempb\bbl@tempa\@{ }}
```

9.9 Hyphens

`\babelhyphenation` This macro saves hyphenation exceptions. Two macros are used to store them:
`\bbl@hyphenation@` for the global ones and `\bbl@hyphenation<lang>` for language ones.

See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```

2357 \bbl@trace{Hyphens}
2358 \@onlypreamble\babelhyphenation
2359 \AtEndOfPackage{%
2360   \newcommand\babelhyphenation[2][\@empty]{%
2361     \ifx\bbl@hyphenation@relax
2362       \let\bbl@hyphenation@\@empty
2363     \fi
2364     \ifx\bbl@hyphlist\@empty\else
2365       \bbl@warning{%
2366         You must not intermingle \string\selectlanguage\space and\%
2367         \string\babelhyphenation\space or some exceptions will not\%
2368         be taken into account. Reported}%
2369     \fi
2370     \ifx\@empty#1%
2371       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2372     \else
2373       \bbl@vforeach{#1}{%
2374         \def\bbl@tempa{##1}%
2375         \bbl@fixname\bbl@tempa
2376         \bbl@iflanguage\bbl@tempa{%
2377           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2378             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2379             \@empty
2380             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2381             #2}}}%
2382       \fi}}

```

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

```

2383 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
2384 \def\bbl@t@one{T1}
2385 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

`\babelhyphen` Macros to insert common hyphens. Note the space before `@` in `\babelhyphen`. Instead of protecting it with `\DeclareRobustCommand`, which could insert a `\relax`, we use the same procedure as shorthands, with `\active@prefix`.

```

2386 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2387 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2388 \def\bbl@hyphen{%
2389   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2390 \def\bbl@hyphen@i#1#2{%
2391   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2392   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2393   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

```

2394 \def\bbl@usehyphen#1{%

```

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

```

2395 \leavevmode
2396 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2397 \nobreak\hskip\z@skip}
2398 \def\bbl@usehyphen#1{%
2399 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2400 \def\bbl@hyphenchar{%
2401 \ifnum\hyphenchar\font=\m@ne
2402 \babeinullhyphen
2403 \else
2404 \char\hyphenchar\font
2405 \fi}

```

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

```

2406 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2407 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2408 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2409 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2410 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2411 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2412 \def\bbl@hy@repeat{%
2413 \bbl@usehyphen{%
2414 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2415 \def\bbl@hy@repeat{%
2416 \bbl@usehyphen{%
2417 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2418 \def\bbl@hy@empty{\hskip\z@skip}
2419 \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.

```

2420 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

9.10 Multiencoding strings

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

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

```

2421 \bbl@trace{Multiencoding strings}
2422 \def\bbl@tglobal#1{\global\let#1#1}
2423 \def\bbl@recatcode#1{% TODO. Used only once?
2424 \@tempcnta="7F
2425 \def\bbl@tempa{%
2426 \ifnum\@tempcnta>"FF\else
2427 \catcode\@tempcnta=#1\relax
2428 \advance\@tempcnta\@ne
2429 \expandafter\bbl@tempa
2430 \fi}%
2431 \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).

```
2432 \@ifpackagewith{babel}{nocase}%
2433   {\let\bbl@patchuclc\relax}%
2434   {\def\bbl@patchuclc{%
2435     \global\let\bbl@patchuclc\relax
2436     \gaddto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2437     \gdef\bbl@uclc##1{%
2438       \let\bbl@encoded\bbl@encoded@uclc
2439       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2440       {##1}%
2441       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2442        \csname\language @bbl@uclc\endcsname}%
2443       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2444   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2445   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%

2446 <<More package options>> ≡
2447 \DeclareOption{nocase}{}
2448 <</More package options>>
```

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

```
2449 <<More package options>> ≡
2450 \let\bbl@opt@strings\@nnil % accept strings=value
2451 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2452 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2453 \def\BabelStringsDefault{generic}
2454 <</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.

```
2455 \@onlypreamble\StartBabelCommands
2456 \def\StartBabelCommands{%
2457   \begingroup
2458   \bbl@recatcode{11}%
2459   <<Macros local to BabelCommands>>
2460   \def\bbl@provstring##1##2{%
2461     \providecommand##1{##2}%
2462     \bbl@tglobal##1}%
2463   \global\let\bbl@scafter\@empty
2464   \let\StartBabelCommands\bbl@startcmds
2465   \ifx\BabelLanguages\relax
2466     \let\BabelLanguages\CurrentOption
2467   \fi
2468   \begingroup
2469   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2470   \StartBabelCommands}
2471 \def\bbl@startcmds{%
2472   \ifx\bbl@screset\@nnil\else
```

```

2473 \bbl@usehooks{stopcommands}{}%
2474 \fi
2475 \endgroup
2476 \begingroup
2477 \@ifstar
2478 {\ifx\bbl@opt@strings\@nnil
2479 \let\bbl@opt@strings\BabelStringsDefault
2480 \fi
2481 \bbl@startcmds@i}%
2482 \bbl@startcmds@i}
2483 \def\bbl@startcmds@i#1#2{%
2484 \edef\bbl@L{\zap@space#1 \@empty}%
2485 \edef\bbl@G{\zap@space#2 \@empty}%
2486 \bbl@startcmds@ii}
2487 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

2488 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2489 \let\SetString@gobbletwo
2490 \let\bbl@stringdef@gobbletwo
2491 \let\AfterBabelCommands@gobble
2492 \ifx\@empty#1%
2493 \def\bbl@sc@label{generic}%
2494 \def\bbl@encstring##1##2{%
2495 \ProvideTextCommandDefault##1{##2}%
2496 \bbl@tglobal##1%
2497 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2498 \let\bbl@sctest\in@true
2499 \else
2500 \let\bbl@sc@charset\space % <- zapped below
2501 \let\bbl@sc@fontenc\space % <- " "
2502 \def\bbl@tempa##1=##2\@nil{%
2503 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2504 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2505 \def\bbl@tempa##1 ##2{% space -> comma
2506 ##1%
2507 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2508 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2509 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2510 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2511 \def\bbl@encstring##1##2{%
2512 \bbl@foreach\bbl@sc@fontenc{%
2513 \bbl@ifunset{T@###1}%
2514 {}%
2515 {\ProvideTextCommand##1{###1}{##2}%
2516 \bbl@tglobal##1%
2517 \expandafter
2518 \bbl@tglobal\csname###1\string##1\endcsname}}}%
2519 \def\bbl@sctest{%

```

```

2520 \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
2521 \fi
2522 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2523 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2524 \let\AfterBabelCommands\bbl@aftercmds
2525 \let\SetString\bbl@setstring
2526 \let\bbl@stringdef\bbl@encstring
2527 \else % ie, strings=value
2528 \bbl@sctest
2529 \ifin@
2530 \let\AfterBabelCommands\bbl@aftercmds
2531 \let\SetString\bbl@setstring
2532 \let\bbl@stringdef\bbl@provstring
2533 \fi\fi\fi
2534 \bbl@scswitch
2535 \ifx\bbl@G\@empty
2536 \def\SetString##1##2{%
2537 \bbl@error{Missing group for string \string##1}%
2538 {You must assign strings to some category, typically\\%
2539 captions or extras, but you set none}}%
2540 \fi
2541 \ifx\@empty#1%
2542 \bbl@usehooks{defaultcommands}{}%
2543 \else
2544 \@expandtwoargs
2545 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2546 \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).

```

2547 \def\bbl@forlang#1#2{%
2548 \bbl@for#1\bbl@L{%
2549 \bbl@xin@{, #1, }{\BabelLanguages,}%
2550 \ifin@#2\relax\fi}}
2551 \def\bbl@scswitch{%
2552 \bbl@forlang\bbl@tempa{%
2553 \ifx\bbl@G\@empty\else
2554 \ifx\SetString\@gobbletwo\else
2555 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2556 \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
2557 \ifin@\else
2558 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2559 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2560 \fi
2561 \fi
2562 \fi}}
2563 \AtEndOfPackage{%
2564 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2565 \let\bbl@scswitch\relax}
2566 \@onlypreamble\EndBabelCommands
2567 \def\EndBabelCommands{%

```

```

2568 \bbl@usehooks{stopcommands}{}%
2569 \endgroup
2570 \endgroup
2571 \bbl@scafter}
2572 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

Strings The following macro is the actual definition of \SetString when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like \providescommand). With the event stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

2573 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2574 \bbl@forlang\bbl@tempa{%
2575 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2576 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2577 {\bbl@exp{%
2578 \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2579 }%
2580 \def\BabelString{#2}%
2581 \bbl@usehooks{stringprocess}{}%
2582 \expandafter\bbl@stringdef
2583 \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.

```

2584 \ifx\bbl@opt@strings\relax
2585 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2586 \bbl@patchuclc
2587 \let\bbl@encoded\relax
2588 \def\bbl@encoded@uclc#1{%
2589 \@inmathwarn#1%
2590 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2591 \expandafter\ifx\csname ?\string#1\endcsname\relax
2592 \TextSymbolUnavailable#1%
2593 \else
2594 \csname ?\string#1\endcsname
2595 \fi
2596 \else
2597 \csname\cf@encoding\string#1\endcsname
2598 \fi}
2599 \else
2600 \def\bbl@scset#1#2{\def#1{#2}}
2601 \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.

```

2602 <<(*Macros local to BabelCommands)>> ≡
2603 \def\SetStringLoop##1##2{%
2604 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2605 \count@\z@
2606 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2607 \advance\count@\@ne

```



```

2608      \toks@expandafter{\bbl@tempa}%
2609      \bbl@exp{%
2610        \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2611        \count@=\the\count@relax}}}%
2612 <</Macros local to BabelCommands>>

```

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

```

2613 \def\bbl@aftercmds#1{%
2614   \toks@expandafter{\bbl@scafter#1}%
2615   \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.

```

2616 <<*Macros local to BabelCommands>> ≡
2617   \newcommand\SetCase[3][]{%
2618     \bbl@patchuclc
2619     \bbl@forlang\bbl@tempa{%
2620       \expandafter\bbl@encstring
2621       \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2622       \expandafter\bbl@encstring
2623       \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2624       \expandafter\bbl@encstring
2625       \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2626 <</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.

```

2627 <<*Macros local to BabelCommands>> ≡
2628   \newcommand\SetHyphenMap[1]{%
2629     \bbl@forlang\bbl@tempa{%
2630       \expandafter\bbl@stringdef
2631       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2632 <</Macros local to BabelCommands>>

```

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

```

2633 \newcommand\BabelLower[2]{% one to one.
2634   \ifnum\lccode#1=#2\else
2635     \babel@savevariable{\lccode#1}%
2636     \lccode#1=#2relax
2637   \fi}
2638 \newcommand\BabelLowerMM[4]{% many-to-many
2639   \@tempcnta=#1relax
2640   \@tempcntb=#4relax
2641   \def\bbl@tempa{%
2642     \ifnum\@tempcnta>#2\else
2643       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2644       \advance\@tempcnta#3relax
2645       \advance\@tempcntb#3relax
2646       \expandafter\bbl@tempa
2647     \fi}%
2648   \bbl@tempa}
2649 \newcommand\BabelLowerMO[4]{% many-to-one
2650   \@tempcnta=#1relax
2651   \def\bbl@tempa{%
2652     \ifnum\@tempcnta>#2\else

```

```

2653 \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2654 \advance\@tempcnta#3
2655 \expandafter\bb1@tempa
2656 \fi}%
2657 \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2658 <<(*More package options)>> ≡
2659 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
2660 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
2661 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
2662 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@}
2663 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2664 <</More package options>>

```

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

```

2665 \AtEndOfPackage{%
2666 \ifx\bb1@opt@hyphenmap\undefined
2667 \bb1@xin{,}{\bb1@language@opts}%
2668 \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
2669 \fi}

```

9.11 Macros common to a number of languages

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

```

2670 \bb1@trace{Macros related to glyphs}
2671 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2672 \dimen\z@\ht\z@\advance\dimen\z@-\ht\tw@%
2673 \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.

```

2674 \def\save@sf@q#1{\leavevmode
2675 \begingroup
2676 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2677 \endgroup}

```

9.12 Making glyphs available

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

9.12.1 Quotation marks

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

```

2678 \ProvideTextCommand{\quotedblbase}{OT1}{%
2679 \save@sf@q{\set@low@box{\textquotedblright\}}%
2680 \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

2681 \ProvideTextCommandDefault{\quotedblbase}{%
2682 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```
2683 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2684   \save@sf@q{\set@low@box{\textquoteright\}%
2685     \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
2686 \ProvideTextCommandDefault{\quotesinglbase}{%
2687   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2688 \ProvideTextCommand{\guillemetleft}{OT1}{%
2689   \ifmmode
2690     \ll
2691   \else
2692     \save@sf@q{\nobreak
2693       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2694   \fi}
2695 \ProvideTextCommand{\guillemetright}{OT1}{%
2696   \ifmmode
2697     \gg
2698   \else
2699     \save@sf@q{\nobreak
2700       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2701   \fi}
2702 \ProvideTextCommand{\guillemotleft}{OT1}{%
2703   \ifmmode
2704     \ll
2705   \else
2706     \save@sf@q{\nobreak
2707       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2708   \fi}
2709 \ProvideTextCommand{\guillemotright}{OT1}{%
2710   \ifmmode
2711     \gg
2712   \else
2713     \save@sf@q{\nobreak
2714       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2715   \fi}
```

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

```
2716 \ProvideTextCommandDefault{\guillemetleft}{%
2717   \UseTextSymbol{OT1}{\guillemetleft}}
2718 \ProvideTextCommandDefault{\guillemetright}{%
2719   \UseTextSymbol{OT1}{\guillemetright}}
2720 \ProvideTextCommandDefault{\guillemotleft}{%
2721   \UseTextSymbol{OT1}{\guillemotleft}}
2722 \ProvideTextCommandDefault{\guillemotright}{%
2723   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2724 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2725   \ifmmode
2726     <%
2727   \else
2728     \save@sf@q{\nobreak
```

```

2729 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2730 \fi}
2731 \ProvideTextCommand{\guilsinglright}{OT1}{%
2732 \ifmmode
2733 >%
2734 \else
2735 \save@sf@q{\nobreak
2736 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2737 \fi}

```

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

```

2738 \ProvideTextCommandDefault{\guilsinglleft}{%
2739 \UseTextSymbol{OT1}{\guilsinglleft}}
2740 \ProvideTextCommandDefault{\guilsinglright}{%
2741 \UseTextSymbol{OT1}{\guilsinglright}}

```

9.12.2 Letters

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

```

2742 \DeclareTextCommand{\ij}{OT1}{%
2743 i\kern-0.02em\bbl@allowhyphens j}
2744 \DeclareTextCommand{\IJ}{OT1}{%
2745 I\kern-0.02em\bbl@allowhyphens J}
2746 \DeclareTextCommand{\ij}{T1}{\char188}
2747 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2748 \ProvideTextCommandDefault{\ij}{%
2749 \UseTextSymbol{OT1}{\ij}}
2750 \ProvideTextCommandDefault{\IJ}{%
2751 \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

2752 \def\crrtic@{\hrule height0.1ex width0.3em}
2753 \def\crttic@{\hrule height0.1ex width0.33em}
2754 \def\ddj@{%
2755 \setbox0\hbox{d}\dimen@=\ht0
2756 \advance\dimen@1ex
2757 \dimen@.45\dimen@
2758 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2759 \advance\dimen@ii.5ex
2760 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2761 \def\DDJ@{%
2762 \setbox0\hbox{D}\dimen@=.55\ht0
2763 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2764 \advance\dimen@ii.15ex % correction for the dash position
2765 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2766 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2767 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2768 %
2769 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2770 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```
2771 \ProvideTextCommandDefault{\dj}{%
2772   \UseTextSymbol{OT1}{\dj}}
2773 \ProvideTextCommandDefault{\DJ}{%
2774   \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.

```
2775 \DeclareTextCommand{\SS}{OT1}{SS}
2776 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.12.3 Shorthands for quotation marks

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

\glq The ‘german’ single quotes.

```
\grq 2777 \ProvideTextCommandDefault{\glq}{%
2778   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2779 \ProvideTextCommand{\grq}{T1}{%
2780   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2781 \ProvideTextCommand{\grq}{TU}{%
2782   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2783 \ProvideTextCommand{\grq}{OT1}{%
2784   \save@sf@q{\kern-.0125em
2785     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2786     \kern.07em\relax}}
2787 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The ‘german’ double quotes.

```
\grqq 2788 \ProvideTextCommandDefault{\glqq}{%
2789   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2790 \ProvideTextCommand{\grqq}{T1}{%
2791   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2792 \ProvideTextCommand{\grqq}{TU}{%
2793   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2794 \ProvideTextCommand{\grqq}{OT1}{%
2795   \save@sf@q{\kern-.07em
2796     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2797     \kern.07em\relax}}
2798 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq The ‘french’ single guillemets.

```
\frq 2799 \ProvideTextCommandDefault{\flq}{%
2800   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2801 \ProvideTextCommandDefault{\frq}{%
2802   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```

\frqq 2803 \ProvideTextCommandDefault{\flqq}{%
2804   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2805 \ProvideTextCommandDefault{\frqq}{%
2806   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

9.12.4 Umlauts and tremas

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

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

```

2807 \def\umlauthigh{%
2808   \def\bbl@umlauta##1{\leavevmode\bgroup%
2809     \expandafter\accent\csname\fontencoding dqpos\endcsname
2810     ##1\bbl@allowhyphens\egroup}%
2811   \let\bbl@umlaute\bbl@umlauta}
2812 \def\umlautlow{%
2813   \def\bbl@umlauta{\protect\lower@umlaut}}
2814 \def\umlautelower{%
2815   \def\bbl@umlaute{\protect\lower@umlaut}}
2816 \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.

```

2817 \expandafter\ifx\csname U@D\endcsname\relax
2818   \csname newdimen\endcsname\U@D
2819 \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.

```

2820 \def\lower@umlaut#1{%
2821   \leavevmode\bgroup
2822     \U@D 1ex%
2823     {\setbox\z@\hbox{%
2824       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2825       \dimen@ -.45ex\advance\dimen@\ht\z@
2826       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2827     \expandafter\accent\csname\fontencoding dqpos\endcsname
2828     \fontdimen5\font\U@D #1%
2829   \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).

```

2830 \AtBeginDocument{%
2831   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2832   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2833   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2834   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
2835   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2836   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2837   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2838   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2839   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2840   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2841   \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}

```

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

```

2842 \ifx\l@english\@undefined
2843   \chardef\l@english\z@
2844 \fi
2845 % The following is used to cancel rules in ini files (see Amharic).
2846 \ifx\l@babelnohyphens\@undefined
2847   \newlanguage\l@babelnohyphens
2848 \fi

```

9.13 Layout

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

```

2849 \bbl@trace{Bidi layout}
2850 \providecommand\IfBabelLayout[3]{#3}%
2851 \newcommand\BabelPatchSection[1]{%
2852   \@ifundefined{#1}{}{%
2853     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2854     \@namedef{#1}{%
2855       \@ifstar{\bbl@presec@s{#1}}%
2856       {\@dblarg{\bbl@presec@x{#1}}}}%
2857 \def\bbl@presec@x#1[#2]#3{%
2858   \bbl@exp{%
2859     \\\select@language@x{\bbl@main@language}%
2860     \\\bbl@cs{sspre@#1}%
2861     \\\bbl@cs{ss@#1}%
2862     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2863     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2864     \\\select@language@x{\language}}%
2865 \def\bbl@presec@s#1#2{%
2866   \bbl@exp{%
2867     \\\select@language@x{\bbl@main@language}%
2868     \\\bbl@cs{sspre@#1}%
2869     \\\bbl@cs{ss@#1}*%
2870     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2871     \\\select@language@x{\language}}%
2872 \IfBabelLayout{sectioning}%
2873   {\BabelPatchSection{part}%
2874     \BabelPatchSection{chapter}%

```

```

2875 \BabelPatchSection{section}%
2876 \BabelPatchSection{subsection}%
2877 \BabelPatchSection{subsubsection}%
2878 \BabelPatchSection{paragraph}%
2879 \BabelPatchSection{subparagraph}%
2880 \def\babel@toc#1{%
2881     \select@language@x{\bbl@main@language}}{}
2882 \IfBabelLayout{captions}%
2883 {\BabelPatchSection{caption}}{}

```

9.14 Load engine specific macros

```

2884 \bbl@trace{Input engine specific macros}
2885 \ifcase\bbl@engine
2886 \input txtbabel.def
2887 \or
2888 \input luababel.def
2889 \or
2890 \input xebabel.def
2891 \fi

```

9.15 Creating and modifying languages

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

```

2892 \bbl@trace{Creating languages and reading ini files}
2893 \newcommand\babelprovide[2][]{%
2894     \let\bbl@savelangname\language
2895     \edef\bbl@savelocaleid{\the\localeid}%
2896     % Set name and locale id
2897     \edef\language{#2}%
2898     % \global\@namedef{\bbl@lcname@#2}{#2}%
2899     \bbl@id@assign
2900     \let\bbl@KVP@captions\@nil
2901     \let\bbl@KVP@date\@nil
2902     \let\bbl@KVP@import\@nil
2903     \let\bbl@KVP@main\@nil
2904     \let\bbl@KVP@script\@nil
2905     \let\bbl@KVP@language\@nil
2906     \let\bbl@KVP@hyphenrules\@nil % only for provide@new
2907     \let\bbl@KVP@mapfont\@nil
2908     \let\bbl@KVP@maparabic\@nil
2909     \let\bbl@KVP@mapdigits\@nil
2910     \let\bbl@KVP@intraspace\@nil
2911     \let\bbl@KVP@intrapenalty\@nil
2912     \let\bbl@KVP@onchar\@nil
2913     \let\bbl@KVP@alph\@nil
2914     \let\bbl@KVP@Alph\@nil
2915     \let\bbl@KVP@labels\@nil
2916     \bbl@csarg\let{KVP@labels*}\@nil
2917     \bbl@forkv{#1}{% TODO - error handling
2918         \in@{/{}}{##1}%
2919         \ifin@
2920             \bbl@renewinikey##1\@{##2}%
2921         \else
2922             \bbl@csarg\def{KVP@##1}{##2}%
2923         \fi}%

```



```

2924 % == import, captions ==
2925 \ifx\bb1@KVP@import\@nil\else
2926   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
2927   {\ifx\bb1@initload\relax
2928     \begingroup
2929       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
2930       \bb1@input@texini{#2}%
2931     \endgroup
2932   \else
2933     \xdef\bb1@KVP@import{\bb1@initload}%
2934   \fi}%
2935 {}%
2936 \fi
2937 \ifx\bb1@KVP@captions\@nil
2938   \let\bb1@KVP@captions\bb1@KVP@import
2939 \fi
2940 % Load ini
2941 \bb1@ifunset{date#2}%
2942   {\bb1@provide@new{#2}}%
2943   {\bb1@ifblank{#1}%
2944     {\bb1@error
2945       {If you want to modify `#2' you must tell how in\\
2946       the optional argument. See the manual for the\\
2947       available options.}%
2948       {Use this macro as documented}}%
2949     {\bb1@provide@renew{#2}}}%
2950 % Post tasks
2951 \bb1@ifunset{\bb1@extracaps@#2}%
2952   {\bb1@exp{\bb1@babelensure[exclude=\\today]{#2}}}%
2953   {\toks@%
2954     \expandafter\expandafter\expandafter
2955     {\csname \bb1@extracaps@#2\endcsname}%
2956     \bb1@exp{\bb1@babelensure[exclude=\\today,include=\the\toks@]{#2}}}%
2957 \bb1@ifunset{\bb1@ensure@language}%
2958   {\bb1@exp{%
2959     \\DeclareRobustCommand\<\bb1@ensure@language>[1]{%
2960       \\foreignlanguage{\language}%
2961       {###1}}}%
2962   \bb1@exp{%
2963     \\bb1@toglobal\<\bb1@ensure@language>%
2964     \\bb1@toglobal\<\bb1@ensure@language\space>%
2965 % At this point all parameters are defined if 'import'. Now we
2966 % execute some code depending on them. But what about if nothing was
2967 % imported? We just load the very basic parameters.
2968 \bb1@load@basic{#2}%
2969 % == script, language ==
2970 % Override the values from ini or defines them
2971 \ifx\bb1@KVP@script\@nil\else
2972   \bb1@csarg\edef{sname@#2}{\bb1@KVP@script}%
2973 \fi
2974 \ifx\bb1@KVP@language\@nil\else
2975   \bb1@csarg\edef{lname@#2}{\bb1@KVP@language}%
2976 \fi
2977 % == onchar ==
2978 \ifx\bb1@KVP@onchar\@nil\else
2979   \bb1@luahyphenate
2980   \directlua{
2981     if Babel.locale_mapped == nil then
2982       Babel.locale_mapped = true

```

```

2983     Babel.linebreaking.add_before(Babel.locale_map)
2984     Babel.loc_to_scr = {}
2985     Babel.chr_to_loc = Babel.chr_to_loc or {}
2986   end}%
2987   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2988   \ifin@
2989     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2990       \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2991     \fi
2992     \bbl@exp{\\bbl@add\\bbl@starthyphens
2993       {\bbl@patterns@lua{\language}}}%
2994     % TODO - error/warning if no script
2995     \directlua{
2996       if Babel.script_blocks['\bbl@cl{sbc}'] then
2997         Babel.loc_to_scr[\the\localeid] =
2998           Babel.script_blocks['\bbl@cl{sbc}']
2999         Babel.locale_props[\the\localeid].lc = \the\localeid\space
3000         Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
3001       end
3002     }%
3003   \fi
3004   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3005   \ifin@
3006     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3007     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3008     \directlua{
3009       if Babel.script_blocks['\bbl@cl{sbc}'] then
3010         Babel.loc_to_scr[\the\localeid] =
3011           Babel.script_blocks['\bbl@cl{sbc}']
3012       end}%
3013     \ifx\bbl@mapselect\undefined
3014       \AtBeginDocument{%
3015         \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3016         {\selectfont}}%
3017       \def\bbl@mapselect{%
3018         \let\bbl@mapselect\relax
3019         \edef\bbl@prefontid{\fontid\font}}%
3020       \def\bbl@mapdir##1{%
3021         {\def\language{##1}%
3022          \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3023          \bbl@switchfont
3024          \directlua{
3025            Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
3026              [\bbl@prefontid] = \fontid\font\space}}}%
3027       \fi
3028       \bbl@exp{\\bbl@add\\bbl@mapselect{\bbl@mapdir{\language}}}%
3029     \fi
3030     % TODO - catch non-valid values
3031   \fi
3032   % == mapfont ==
3033   % For bidi texts, to switch the font based on direction
3034   \ifx\bbl@KVP@mapfont\@nil\else
3035     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
3036     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\
3037       mapfont. Use 'direction'.%
3038       {See the manual for details.}}}%
3039     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3040     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3041     \ifx\bbl@mapselect\undefined

```

```

3042 \AtBeginDocument{%
3043   \expandafter\bbbl@add\csname selectfont \endcsname{\bbbl@mapselect}}%
3044   {\selectfont}}%
3045 \def\bbbl@mapselect{%
3046   \let\bbbl@mapselect\relax
3047   \edef\bbbl@prefontid{\fontid\font}}%
3048 \def\bbbl@mapdir##1{%
3049   {\def\language##1}%
3050   \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
3051   \bbbl@switchfont
3052   \directlua{Babel.fontmap
3053     [\the\csname bbl@wdir@##1\endcsname]%
3054     [\bbbl@prefontid]=\fontid\font}}}%
3055 \fi
3056 \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
3057 \fi
3058 % == intraspace, intrapenalty ==
3059 % For CJK, East Asian, Southeast Asian, if interspace in ini
3060 \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
3061   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
3062 \fi
3063 \bbbl@provide@intraspace
3064 % == hyphenate.other.locale ==
3065 \bbbl@ifunset{bbl@hyotl@language}{}%
3066   {\bbbl@csarg\bbbl@replace{hyotl@language}{ }{,}}%
3067   \bbbl@startcommands*{language}{}%
3068   \bbbl@csarg\bbbl@foreach{hyotl@language}{%
3069     \ifcase\bbbl@engine
3070       \ifnum##1<257
3071         \SetHyphenMap{\BabelLower{##1}{##1}}%
3072       \fi
3073       \else
3074         \SetHyphenMap{\BabelLower{##1}{##1}}%
3075       \fi}%
3076   \bbbl@endcommands}%
3077 % == hyphenate.other.script ==
3078 \bbbl@ifunset{bbl@hyots@language}{}%
3079   {\bbbl@csarg\bbbl@replace{hyots@language}{ }{,}}%
3080   \bbbl@csarg\bbbl@foreach{hyots@language}{%
3081     \ifcase\bbbl@engine
3082       \ifnum##1<257
3083         \global\lccode##1=##1\relax
3084       \fi
3085       \else
3086         \global\lccode##1=##1\relax
3087       \fi}}%
3088 % == maparabic ==
3089 % Native digits, if provided in ini (TeX level, xe and lua)
3090 \ifcase\bbbl@engine\else
3091   \bbbl@ifunset{bbl@dgnat@language}{}%
3092   {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
3093     \expandafter\expandafter\expandafter
3094     \bbbl@setdigits\csname bbl@dgnat@language\endcsname
3095     \ifx\bbbl@KVP@maparabic\@nil\else
3096       \ifx\bbbl@latinarabic\@undefined
3097         \expandafter\let\expandafter\@arabic
3098         \csname bbl@counter@language\endcsname
3099       \else % ie, if layout=counters, which redefines \@arabic
3100         \expandafter\let\expandafter\bbbl@latinarabic

```

```

3101         \csname bbl@counter@\language\endcsname
3102     \fi
3103     \fi
3104     \fi}%
3105 \fi
3106 % == mapdigits ==
3107 % Native digits (lua level).
3108 \ifodd\bbl@engine
3109     \ifx\bbl@KVP@mapdigits\@nil\else
3110         \bbl@ifunset{bbl@dgnat\language}{}%
3111         {\RequirePackage{luatexbase}%
3112         \bbl@activate@preotf
3113         \directlua{
3114             Babel = Babel or {} %%% -> presets in luababel
3115             Babel.digits_mapped = true
3116             Babel.digits = Babel.digits or {}
3117             Babel.digits[\the\localeid] =
3118                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3119             if not Babel.numbers then
3120                 function Babel.numbers(head)
3121                     local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3122                     local GLYPH = node.id'glyph'
3123                     local inmath = false
3124                     for item in node.traverse(head) do
3125                         if not inmath and item.id == GLYPH then
3126                             local temp = node.get_attribute(item, LOCALE)
3127                             if Babel.digits[temp] then
3128                                 local chr = item.char
3129                                 if chr > 47 and chr < 58 then
3130                                     item.char = Babel.digits[temp][chr-47]
3131                                 end
3132                             end
3133                             elseif item.id == node.id'math' then
3134                                 inmath = (item.subtype == 0)
3135                             end
3136                         end
3137                     return head
3138                 end
3139             end
3140         }}%
3141     \fi
3142 \fi
3143 % == alph, Alph ==
3144 % What if extras<lang> contains a \babel@save\@alph? It won't be
3145 % restored correctly when exiting the language, so we ignore
3146 % this change with the \bbl@alph@saved trick.
3147 \ifx\bbl@KVP@alph\@nil\else
3148     \toks@\expandafter\expandafter\expandafter{%
3149         \csname extras\language\endcsname}%
3150     \bbl@exp{%
3151         \def\<extras\language>{%
3152             \let\\bbl@alph@saved\\@alph
3153             \the\toks@
3154             \let\\@alph\\bbl@alph@saved
3155             \\babel@save\\@alph
3156             \let\\@alph\<bbl@cntr\bbl@KVP@alph @\language>}}%
3157 \fi
3158 \ifx\bbl@KVP@Alph\@nil\else
3159     \toks@\expandafter\expandafter\expandafter{%

```

```

3160 \csname extras\language\endcsname}%
3161 \bbl@exp{%
3162 \def\<extras\language>{%
3163 \let\\bbl@Alph@savd\\@Alph
3164 \the\toks@
3165 \let\\@Alph\\bbl@Alph@savd
3166 \\babel@save\\@Alph
3167 \let\\@Alph<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3168 \fi
3169 % == require.babel in ini ==
3170 % To load or reload the babel-*.tex, if require.babel in ini
3171 \bbl@ifunset{bbl@rqtex\language}{}%
3172 {\expandafter\ifx\csname bbl@rqtex\language\endcsname\@empty\else
3173 \let\BabelBeforeIni@gobbles
3174 \chardef\atcatcode=\catcode\@
3175 \catcode\@=11\relax
3176 \bbl@input@texini{\bbl@cs{rqtex\language}}%
3177 \catcode\@=\atcatcode
3178 \let\atcatcode\relax
3179 \fi}%
3180 % == main ==
3181 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3182 \let\language\bbl@savelangname
3183 \chardef\localeid\bbl@savelocaleid\relax
3184 \fi}

```

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

```

3185 \def\bbl@provide@new#1{%
3186 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3187 \@namedef{extras#1}{}%
3188 \@namedef{noextras#1}{}%
3189 \bbl@startcommands*{#1}{captions}%
3190 \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3191 \def\bbl@tempb##1{% elt for \bbl@captionslist
3192 \ifx##1\@empty\else
3193 \bbl@exp{%
3194 \\SetString\\##1{%
3195 \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}%
3196 \expandafter\bbl@tempb
3197 \fi}%
3198 \expandafter\bbl@tempb\bbl@captionslist\@empty
3199 \else
3200 \ifx\bbl@initoload\relax
3201 \bbl@read@ini{\bbl@KVP@captions}0% Here letters cat = 11
3202 \else
3203 \bbl@read@ini{\bbl@initoload}0% Here all letters cat = 11
3204 \fi
3205 \bbl@after@ini
3206 \bbl@savestrings
3207 \fi
3208 \StartBabelCommands*{#1}{date}%
3209 \ifx\bbl@KVP@import\@nil
3210 \bbl@exp{%
3211 \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
3212 \else
3213 \bbl@savetoday
3214 \bbl@savedate
3215 \fi
3216 \bbl@endcommands

```

```

3217 \bbl@load@basic{#1}%
3218 % == hyphenmins == (only if new)
3219 \bbl@exp{%
3220   \gdef\<#1hyphenmins>{%
3221     {\bbl@ifunset{\bbl@lftm@#1}{2}{\bbl@cs{lftm@#1}}}%
3222     {\bbl@ifunset{\bbl@rgtm@#1}{3}{\bbl@cs{rgtm@#1}}}}}%
3223 % == hyphenrules ==
3224 \bbl@provide@hyphens{#1}%
3225 % == frenchspacing == (only if new)
3226 \bbl@ifunset{\bbl@frspc@#1}{}%
3227 {\edef\bbl@tempa{\bbl@cl{frspc}}}%
3228 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
3229 \if u\bbl@tempa      % do nothing
3230 \else\if n\bbl@tempa % non french
3231   \expandafter\bbl@add\csname extras#1\endcsname{%
3232     \let\bbl@elt\bbl@fs@elt@i
3233     \bbl@fs@chars}%
3234 \else\if y\bbl@tempa % french
3235   \expandafter\bbl@add\csname extras#1\endcsname{%
3236     \let\bbl@elt\bbl@fs@elt@ii
3237     \bbl@fs@chars}%
3238   \fi\fi\fi}%
3239 %
3240 \ifx\bbl@KVP@main\@nil\else
3241   \expandafter\main@language\expandafter{#1}%
3242   \fi}
3243 % A couple of macros used above, to avoid hashes #####...
3244 \def\bbl@fs@elt@i#1#2#3{%
3245   \ifnum\sfcode`#1=#2\relax
3246     \babel@savevariable{\sfcode`#1}%
3247     \sfcode`#1=#3\relax
3248   \fi}%
3249 \def\bbl@fs@elt@ii#1#2#3{%
3250   \ifnum\sfcode`#1=#3\relax
3251     \babel@savevariable{\sfcode`#1}%
3252     \sfcode`#1=#2\relax
3253   \fi}%
3254 %
3255 \def\bbl@provide@renew#1{%
3256   \ifx\bbl@KVP@captions\@nil\else
3257     \StartBabelCommands*{#1}{captions}%
3258     \bbl@read@ini{\bbl@KVP@captions}0%   Here all letters cat = 11
3259     \bbl@after@ini
3260     \bbl@savestrings
3261     \EndBabelCommands
3262   \fi
3263   \ifx\bbl@KVP@import\@nil\else
3264     \StartBabelCommands*{#1}{date}%
3265     \bbl@savetoday
3266     \bbl@savedate
3267     \EndBabelCommands
3268   \fi
3269   % == hyphenrules ==
3270   \bbl@provide@hyphens{#1}}
3271 % Load the basic parameters (ids, typography, counters, and a few
3272 % more), while captions and dates are left out. But it may happen some
3273 % data has been loaded before automatically, so we first discard the
3274 % saved values.
3275 \def\bbl@linebreak@export{%

```

```

3276 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3277 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3278 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3279 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3280 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3281 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3282 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3283 \bbl@exportkey{intsp}{typography.intraspace}{}%
3284 \bbl@exportkey{chrng}{characters.ranges}{}%
3285 \def\bbl@load@basic#1{%
3286 \bbl@ifunset{bbl@inidata@\language}{}%
3287 {\getlocaleproperty\bbl@tempa{\language}{identification/load.level}%
3288 \ifcase\bbl@tempa\else
3289 \bbl@csarg\let{lname@\language}\relax
3290 \fi}%
3291 \bbl@ifunset{bbl@lname@#1}%
3292 {\def\BabelBeforeIni##1##2{%
3293 \begingroup
3294 \let\bbl@ini@captions@aux@gobbletwo
3295 \def\bbl@inidata####1.####2.####3.####4\relax####5####6}%
3296 \bbl@read@ini{##1}0%
3297 \bbl@linebreak@export
3298 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3299 \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3300 \ifx\bbl@initoload\relax\endinput\fi
3301 \endgroup}%
3302 \begingroup % boxed, to avoid extra spaces:
3303 \ifx\bbl@initoload\relax
3304 \bbl@input@texini{##1}%
3305 \else
3306 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3307 \fi
3308 \endgroup}%
3309 {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3310 \def\bbl@provide@hyphens#1{%
3311 \let\bbl@tempa\relax
3312 \ifx\bbl@KVP@hyphenrules\@nil\else
3313 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3314 \bbl@foreach\bbl@KVP@hyphenrules{%
3315 \ifx\bbl@tempa\relax % if not yet found
3316 \bbl@ifsamestring{##1}{+}%
3317 {\bbl@exp{\addlanguage\<l@##1>}}%
3318 {}%
3319 \bbl@ifunset{l@##1}%
3320 {}%
3321 {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3322 \fi}%
3323 \fi
3324 \ifx\bbl@tempa\relax % if no opt or no language in opt found
3325 \ifx\bbl@KVP@import\@nil
3326 \ifx\bbl@initoload\relax\else
3327 \bbl@exp{ % and hyphenrules is not empty
3328 \bbl@ifblank{\bbl@cs{hyphr@#1}}%
3329 {}%
3330 {\let\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3331 \fi
3332 \else % if importing

```

```

3333 \bbl@exp{% and hyphenrules is not empty
3334 \\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
3335 {}%
3336 {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}}%
3337 \fi
3338 \fi
3339 \bbl@ifunset{\bbl@tempa}% ie, relax or undefined
3340 {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
3341 {\bbl@exp{\\adddialect\<l@#1>\language}}}%
3342 {}}% so, l@<lang> is ok - nothing to do
3343 {\bbl@exp{\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
3344

```

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.

```

3345 \ifx\bbl@readstream\undefined
3346 \csname newread\endcsname\bbl@readstream
3347 \fi
3348 \def\bbl@input@texini#1{%
3349 \bbl@bsphack
3350 \bbl@exp{%
3351 \catcode\==14
3352 \lowercase{\InputIfFileExists{babel-#1.tex}}}%
3353 \catcode\==\the\catcode\relax}%
3354 \bbl@esphack}
3355 \def\bbl@inipreread#1=#2\@{%
3356 \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3357 \bbl@trim\toks@{#2}%
3358 % Move trims here ??
3359 \bbl@ifunset{\bbl@KVP@\bbl@section/\bbl@tempa}%
3360 {\bbl@exp{%
3361 \\\g@addto@macro\\bbl@inidata{%
3362 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3363 \expandafter\bbl@inireader\bbl@tempa=#2\@}%
3364 {}}%
3365 \def\bbl@fetch@ini#1#2{%
3366 \bbl@exp{\def\\bbl@inidata{%
3367 \\\bbl@elt{identification}{tag.ini}{#1}%
3368 \\\bbl@elt{identification}{load.level}{#2}}}%
3369 \openin\bbl@readstream=babel-#1.ini
3370 \ifeof\bbl@readstream
3371 \bbl@error
3372 {There is no ini file for the requested language\\%
3373 (#1). Perhaps you misspelled it or your installation\\%
3374 is not complete.}%
3375 {Fix the name or reinstall babel.}%
3376 \else
3377 \catcode\=[12 \catcode\]=12 \catcode\==12
3378 \catcode\;=12 \catcode\|=12 \catcode\%=14
3379 \bbl@info{Importing
3380 \ifcase#2 \or font and identification \or basic \fi
3381 data for \language\\%
3382 from babel-#1.ini. Reported}%
3383 \loop
3384 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3385 \endlinechar\m@ne
3386 \read\bbl@readstream to \bbl@line
3387 \endlinechar\^^M
3388 \ifx\bbl@line\empty\else

```



```

3389     \expandafter\bbbl@inline\bbbl@line\bbbl@inline
3390     \fi
3391     \repeat
3392     \fi}
3393 \def\bbbl@read@ini#1#2{%
3394     \bbbl@csarg\edef\lini@{\language\language}\bbbl@ini@{#1}%
3395     \let\bbbl@section\@empty
3396     \let\bbbl@savestrings\@empty
3397     \let\bbbl@savetoday\@empty
3398     \let\bbbl@savestate\@empty
3399     \let\bbbl@inireader\bbbl@iniskip
3400     \bbbl@fetch@ini{#1}{#2}%
3401     \bbbl@foreach\bbbl@renewlist{%
3402         \bbbl@ifunset\bbbl@renew@##1}{\bbbl@inisec[##1]\@{}}%
3403     \global\let\bbbl@renewlist\@empty
3404     % Ends last section. See \bbbl@inisec
3405     \def\bbbl@elt##1##2{\bbbl@inireader##1=##2\@{}}%
3406     \bbbl@cs{renew}\bbbl@section}%
3407     \global\bbbl@csarg\let{renew@\bbbl@section}\relax
3408     \bbbl@cs{secpost@\bbbl@section}%
3409     \bbbl@csarg{\global\expandafter\let}{inidata@\language}\bbbl@inidata
3410     \bbbl@exp{\bbbl@add@list\bbbl@ini@loaded{\language}}%
3411     \bbbl@toglobal\bbbl@ini@loaded}
3412 \def\bbbl@inline#1\bbbl@inline{%
3413     \@ifnextchar{\bbbl@inisec{\@ifnextchar;\bbbl@iniskip\bbbl@inipreread}#1\@{}}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start. By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

3414 \def\bbbl@iniskip#1\@{}%      if starts with ;
3415 \def\bbbl@inisec[##1]##2\@{}% if starts with opening bracket
3416 \def\bbbl@elt##1##2{%
3417     \expandafter\toks@\expandafter{%
3418         \expandafter{\bbbl@section}{##1}{##2}}%
3419     \bbbl@exp{%
3420         \g@addto@macro\bbbl@inidata{\bbbl@elt\the\toks@}%
3421         \bbbl@inireader##1=##2\@{}}%
3422     \bbbl@cs{renew}\bbbl@section}%
3423     \global\bbbl@csarg\let{renew@\bbbl@section}\relax
3424     \bbbl@cs{secpost@\bbbl@section}%
3425     % The previous code belongs to the previous section.
3426     % -----
3427     % Now start the current one.
3428     \in@{=date.}{#1}%
3429     \ifin@
3430         \lowercase{\def\bbbl@tempa{#1}}%
3431         \bbbl@replace\bbbl@tempa{=date.gregorian}{}%
3432         \bbbl@replace\bbbl@tempa{=date.}{}%
3433         \in@{.licr=}{#1}%
3434         \ifin@
3435             \ifcase\bbbl@engine
3436                 \bbbl@replace\bbbl@tempa{.licr=}{}%
3437             \else
3438                 \let\bbbl@tempa\relax
3439             \fi
3440         \fi
3441         \ifx\bbbl@tempa\relax\else
3442             \bbbl@replace\bbbl@tempa{=}{}%

```

```

3443 \bbl@exp{%
3444 \def\<bbl@inikv@#1>####1=####2\\@{%
3445 \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3446 \fi
3447 \fi
3448 \def\bbl@section{#1}%
3449 \def\bbl@elt##1##2{%
3450 \@namedef{bbl@KVP@#1/##1}{}}%
3451 \bbl@cs{renew@#1}%
3452 \bbl@cs{secpre@#1}% pre-section 'hook'
3453 \bbl@ifunset{bbl@inikv@#1}%
3454 {\let\bbl@inireader\bbl@iniskip}%
3455 {\bbl@exp{\let\\bbl@inireader\<bbl@inikv@#1>}}
3456 \let\bbl@renewlist\@empty
3457 \def\bbl@renewinikey#1/#2\@#3{%
3458 \bbl@ifunset{bbl@renew@#1}%
3459 {\bbl@add@list\bbl@renewlist{#1}}%
3460 {}}%
3461 \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}

```

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

```

3462 \def\bbl@inikv#1=#2\@{% key=value
3463 \bbl@trim@def\bbl@tempa{#1}%
3464 \bbl@trim\toks@{#2}%
3465 \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.

```

3466 \def\bbl@exportkey#1#2#3{%
3467 \bbl@ifunset{bbl@@kv@#2}%
3468 {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3469 {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3470 \bbl@csarg\gdef{#1@\languagename}{#3}}%
3471 \else
3472 \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
3473 \fi}}

```

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

```

3474 \def\bbl@iniwarning#1{%
3475 \bbl@ifunset{bbl@@kv@identification.warning#1}{}}%
3476 {\bbl@warning{%
3477 From babel-\bbl@cs{lini@\languagename}.ini:\\%
3478 \bbl@cs{@kv@identification.warning#1}\\%
3479 Reported }}}
3480 \let\bbl@inikv@identification\bbl@inikv
3481 \def\bbl@secpost@identification{%
3482 \bbl@iniwarning}%
3483 \ifcase\bbl@engine
3484 \bbl@iniwarning{.pdflatex}%
3485 \or
3486 \bbl@iniwarning{.lualatex}%
3487 \or
3488 \bbl@iniwarning{.xelatex}%
3489 \fi%
3490 \bbl@exportkey{elname}{identification.name.english}{}}%
3491 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%

```

```

3492     {\csname bbl@elname@language\endcsname}}%
3493 \bbl@exportkey{lbcpl}{identification.tag.bcp47}{}% TODO
3494 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3495 \bbl@exportkey{esname}{identification.script.name}{}%
3496 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3497     {\csname bbl@esname@language\endcsname}}%
3498 \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3499 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3500 \ifbbl@bcptoname
3501     \bbl@csarg\xdef{bcp@map@bbl@cl{lbcpl}}{\language}%
3502 \fi}
3503 \let\bbl@inikv@typography\bbl@inikv
3504 \let\bbl@inikv@characters\bbl@inikv
3505 \let\bbl@inikv@numbers\bbl@inikv
3506 \def\bbl@inikv@counters#1=#2\@@{%
3507     \bbl@ifsamestring{#1}{digits}%
3508     {\bbl@error{The counter name 'digits' is reserved for mapping\%
3509         decimal digits}%
3510         {Use another name.}}%
3511     }%
3512 \def\bbl@tempc{#1}%
3513 \bbl@trim@def{\bbl@tempb*}{#2}%
3514 \in@{.1$}{#1$}%
3515 \ifin@
3516     \bbl@replace\bbl@tempc{.1}{}%
3517     \bbl@csarg\protected@xdef{cntr@bbl@tempc @language}{%
3518         \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3519 \fi
3520 \in@{.F.}{#1}%
3521 \ifin@ \else \in@{.S.}{#1} \fi
3522 \ifin@
3523     \bbl@csarg\protected@xdef{cntr@#1@language}{\bbl@tempb*}%
3524 \else
3525     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3526     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3527     \bbl@csarg{\global\expandafter\let}{cntr@#1@language}\bbl@tempa
3528 \fi}
3529 \def\bbl@after@ini{%
3530     \bbl@linebreak@export
3531     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3532     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3533     \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3534     \bbl@toggle\bbl@savetoday
3535     \bbl@toggle\bbl@savestate}

```

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

```

3536 \ifcase\bbl@engine
3537     \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3538         \bbl@ini@captions@aux{#1}{#2}}
3539 \else
3540     \def\bbl@inikv@captions#1=#2\@@{%
3541         \bbl@ini@captions@aux{#1}{#2}}
3542 \fi

```

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

```

3543 \def\bbl@ini@captions@aux#1#2{%
3544     \bbl@trim@def\bbl@tempa{#1}%

```

```

3545 \bbl@xin@{.template}\bbl@tempa}%
3546 \ifin@
3547 \bbl@replace\bbl@tempa{.template}{}%
3548 \def\bbl@toreplace{#2}%
3549 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3550 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3551 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3552 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}%
3553 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3554 \bbl@xin@{,\bbl@tempa,}{,chapter,}%
3555 \ifin@
3556 \bbl@patchchapter
3557 \global\bbl@csarg\let{chapfmt@\language}\bbl@toreplace
3558 \fi
3559 \bbl@xin@{,\bbl@tempa,}{,appendix,}%
3560 \ifin@
3561 \bbl@patchchapter
3562 \global\bbl@csarg\let{appxfmt@\language}\bbl@toreplace
3563 \fi
3564 \bbl@xin@{,\bbl@tempa,}{,part,}%
3565 \ifin@
3566 \bbl@patchpart
3567 \global\bbl@csarg\let{partfmt@\language}\bbl@toreplace
3568 \fi
3569 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3570 \ifin@
3571 \toks@{\expandafter\bbl@toreplace}%
3572 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3573 \fi
3574 \else
3575 \bbl@ifblank{#2}%
3576 {\bbl@exp{%
3577 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3578 {\bbl@trim\toks@{#2}}%
3579 \bbl@exp{%
3580 \bbl@add\bbl@savestrings{%
3581 \SetString\<\bbl@tempa name>{\the\toks@}}%
3582 \toks@{\expandafter\bbl@captionslist}%
3583 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3584 \ifin@else
3585 \bbl@exp{%
3586 \bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3587 \bbl@to\global\<\bbl@extracaps@\language>}%
3588 \fi
3589 \fi}

```

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

```

3590 \def\bbl@list@the{%
3591 part,chapter,section,subsection,subsubsection,paragraph,%
3592 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3593 table,page,footnote,mpfootnote,mpfn}
3594 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3595 \bbl@ifunset\bbl@map@#1@\language}%
3596 {\@nameuse{#1}}%
3597 {\@nameuse{\bbl@map@#1@\language}}%
3598 \def\bbl@inikv@labels#1=#2\@@{%
3599 \in@{.map}{#1}%
3600 \ifin@

```

```

3601 \ifx\bb1@KVP@labels\@nil\else
3602 \bb1@xin@{ map }{ \bb1@KVP@labels\space}%
3603 \ifin@
3604 \def\bb1@tempc{#1}%
3605 \bb1@replace\bb1@tempc{.map}{}%
3606 \in@{,#2,}{,arabic,roman,Roman,alpha,Alph,fnsymbol,}%
3607 \bb1@exp{%
3608 \gdef\<bb1@map@\bb1@tempc @\language\name>%
3609 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3610 \bb1@foreach\bb1@list@the{%
3611 \bb1@ifunset{the##1}{}%
3612 {\bb1@exp{\let\\bb1@tempd\<the##1>}%
3613 \bb1@exp{%
3614 \\bb1@sreplace\<the##1>%
3615 {\<\bb1@tempc>{##1}}{\bb1@map@cnt{\bb1@tempc}{##1}}%
3616 \\bb1@sreplace\<the##1>%
3617 {\<\@empty @\bb1@tempc>\<c@##1>}{\bb1@map@cnt{\bb1@tempc}{##1}}}%
3618 \expandafter\ifx\csname the##1\endcsname\bb1@tempd\else
3619 \toks@ \expandafter\expandafter\expandafter{%
3620 \csname the##1\endcsname}%
3621 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3622 \fi}}%
3623 \fi
3624 \fi
3625 %
3626 \else
3627 %
3628 % The following code is still under study. You can test it and make
3629 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3630 % language dependent.
3631 \in@{enumerate.}{#1}%
3632 \ifin@
3633 \def\bb1@tempa{#1}%
3634 \bb1@replace\bb1@tempa{enumerate.}{}%
3635 \def\bb1@toreplace{#2}%
3636 \bb1@replace\bb1@toreplace{[ ]}{\nobreakspace{}}%
3637 \bb1@replace\bb1@toreplace{[]}{\csname the}%
3638 \bb1@replace\bb1@toreplace{[]}{\endcsname{}}%
3639 \toks@ \expandafter{\bb1@toreplace}%
3640 \bb1@exp{%
3641 \\bb1@add\<extras\language\name>{%
3642 \\babel@save\<labelenum\romannumeral\bb1@tempa>%
3643 \def\<labelenum\romannumeral\bb1@tempa>{\the\toks@}}%
3644 \\bb1@tglobal\<extras\language\name>}%
3645 \fi
3646 \fi}

```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```

3647 \def\bb1@chapttype{chap}
3648 \ifx\@makechapterhead\@undefined
3649 \let\bb1@patchchapter\relax
3650 \else\ifx\thechapter\@undefined
3651 \let\bb1@patchchapter\relax
3652 \else\ifx\ps@headings\@undefined
3653 \let\bb1@patchchapter\relax
3654 \else

```

```

3655 \def\bbl@patchchapter{%
3656   \global\let\bbl@patchchapter\relax
3657   \bbl@add\appendix{\def\bbl@chapttype{appx}}% Not harmful, I hope
3658   \bbl@tglobal\appendix
3659   \bbl@sreplace\ps@headings
3660   {\@chapapp\ \thechapter}%
3661   {\bbl@chapterformat}%
3662   \bbl@tglobal\ps@headings
3663   \bbl@sreplace\chaptermark
3664   {\@chapapp\ \thechapter}%
3665   {\bbl@chapterformat}%
3666   \bbl@tglobal\chaptermark
3667   \bbl@sreplace\@makechapterhead
3668   {\@chapapp\space\thechapter}%
3669   {\bbl@chapterformat}%
3670   \bbl@tglobal\@makechapterhead
3671   \gdef\bbl@chapterformat{%
3672     \bbl@ifunset{\bbl@bbl@chapttype fmt@\languagename}%
3673     {\@chapapp\space\thechapter}
3674     {\@nameuse{\bbl@bbl@chapttype fmt@\languagename}}}}
3675 \fi\fi\fi
3676 \ifx\@part\@undefined
3677   \let\bbl@patchpart\relax
3678 \else
3679   \def\bbl@patchpart{%
3680     \global\let\bbl@patchpart\relax
3681     \bbl@sreplace\@part
3682     {\partname\nobreakspace\thepart}%
3683     {\bbl@partformat}%
3684     \bbl@tglobal\@part
3685     \gdef\bbl@partformat{%
3686       \bbl@ifunset{\bbl@partfmt@\languagename}%
3687       {\partname\nobreakspace\thepart}
3688       {\@nameuse{\bbl@partfmt@\languagename}}}}
3689 \fi

```

Date. TODO. Document

```

3690 % Arguments are _not_ protected.
3691 \let\bbl@calendar\@empty
3692 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3693 \def\bbl@localedate#1#2#3#4{%
3694   \begingroup
3695     \ifx\@empty#1\@empty\else
3696       \let\bbl@ld@calendar\@empty
3697       \let\bbl@ld@variant\@empty
3698       \edef\bbl@tempa{\zap@space#1 \@empty}%
3699       \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3700       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3701       \edef\bbl@calendar{%
3702         \bbl@ld@calendar
3703         \ifx\bbl@ld@variant\@empty\else
3704           .\bbl@ld@variant
3705         \fi}%
3706       \bbl@replace\bbl@calendar{gregorian}{}%
3707     \fi
3708     \bbl@cased
3709     {\@nameuse{\bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3710   \endgroup}
3711 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar

```

```

3712 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3713 \bbl@trim@def\bbl@tempa{#1.#2}%
3714 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3715 {\bbl@trim@def\bbl@tempa{#3}%
3716 \bbl@trim\toks@{#5}%
3717 \@temptokena\expandafter{\bbl@savedate}%
3718 \bbl@exp{% Reverse order - in ini last wins
3719 \def\\bbl@savedate{%
3720 \\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}%
3721 \the\@temptokena}}%
3722 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3723 {\lowercase{\def\bbl@tempb{#6}}%
3724 \bbl@trim@def\bbl@toreplace{#5}%
3725 \bbl@TG@date
3726 \bbl@ifunset\bbl@date@\language @}%
3727 {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3728 % TODO. Move to a better place.
3729 \bbl@exp{%
3730 \gdef\<\language date>\protect\<\language date >}%
3731 \gdef\<\language date >###1###2###3{%
3732 \\bbl@usedategroupttrue
3733 \<bbl@ensure@\language >{%
3734 \\localedate{###1}{###2}{###3}}%
3735 \\bbl@add\\bbl@savetoday{%
3736 \\SetString\\today{%
3737 \<\language date>%
3738 {\the\year}{\the\month}{\the\day}}}%
3739 }%
3740 \ifx\bbl@tempb\empty\else
3741 \global\bbl@csarg\let{date@\language @}\bbl@tempb\bbl@toreplace
3742 \fi}%
3743 {}%

```

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.

```

3744 \let\bbl@calendar\@empty
3745 \newcommand\BabelDateSpace{\nobreakspace}
3746 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3747 \newcommand\BabelDated[1]{\number#1}
3748 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3749 \newcommand\BabelDateM[1]{\number#1}
3750 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3751 \newcommand\BabelDateMMMM[1]{%
3752 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3753 \newcommand\BabelDatey[1]{\number#1}%
3754 \newcommand\BabelDateyy[1]{%
3755 \ifnum#1<10 0\number#1 %
3756 \else\ifnum#1<100 \number#1 %
3757 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3758 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3759 \else
3760 \bbl@error
3761 {Currently two-digit years are restricted to the\
3762 range 0-9999.}%
3763 {There is little you can do. Sorry.}%
3764 \fi\fi\fi\fi}%
3765 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3766 \def\bbl@replace@finish@iii#1{%

```

```

3767 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3768 \def\bbl@TG@date{%
3769 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3770 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3771 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3772 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3773 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3774 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3775 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3776 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3777 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3778 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3779 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3780 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3781 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3782 % Note after \bbl@replace \toks@ contains the resulting string.
3783 % TODO - Using this implicit behavior doesn't seem a good idea.
3784 \bbl@replace@finish@iii\bbl@toreplace}
3785 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3786 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

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

```

3787 \def\bbl@provide@lsys#1{%
3788 \bbl@ifunset{bbl@lname@#1}%
3789 {\bbl@ini@basic{#1}}%
3790 {}%
3791 \bbl@csarg\let{lsys@#1}\@empty
3792 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3793 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3794 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3795 \bbl@ifunset{bbl@lname@#1}{%
3796 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3797 \ifcase\bbl@engine\or\or
3798 \bbl@ifunset{bbl@prehc@#1}{}%
3799 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3800 {}%
3801 {\ifx\bbl@xenohyph\@undefined
3802 \let\bbl@xenohyph\bbl@xenohyph@d
3803 \ifx\AtBeginDocument\@notprerr
3804 \expandafter\@secondoftwo % to execute right now
3805 \fi
3806 \AtBeginDocument{%
3807 \expandafter\bbl@add
3808 \csname selectfont \endcsname{\bbl@xenohyph}%
3809 \expandafter\selectlanguage\expandafter{\language}%
3810 \expandafter\bbl@tglobal\csname selectfont \endcsname}%
3811 \fi}}%
3812 \fi
3813 \bbl@csarg\bbl@tglobal{lsys@#1}}
3814 \def\bbl@xenohyph@d{%
3815 \bbl@ifset{bbl@prehc@language}%
3816 {\ifnum\hyphenchar\font=\defaultshyphenchar
3817 \iffontchar\font\bbl@c1{prehc}\relax
3818 \hyphenchar\font\bbl@c1{prehc}\relax
3819 \else\iffontchar\font"200B
3820 \hyphenchar\font"200B
3821 \else
3822 \bbl@warning

```


Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3872 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3873 \ifx\\#1% % \\ before, in case #1 is multiletter
3874 \bbl@exp{%
3875 \def\\bbl@tempa####1{%
3876 \ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3877 \else
3878 \toks@\expandafter{\the\toks@\or #1}%
3879 \expandafter\bbl@buildifcase
3880 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as a special case, for a fixed form (see babel-he.ini, for example).

```

3881 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\language}\{#2}}
3882 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3883 \newcommand\localecounter[2]{%
3884 \expandafter\bbl@localecntr
3885 \expandafter{\number\csname c@#2\endcsname}\{#1}}
3886 \def\bbl@alphnumeral#1#2{%
3887 \expandafter\bbl@alphnumeral@i\number#2 76543210\@{#1}}
3888 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@#9{%
3889 \ifcase\car#8\@nil\or % Currenty <10000, but prepared for bigger
3890 \bbl@alphnumeral@ii{#9}000000#1\or
3891 \bbl@alphnumeral@ii{#9}00000#1#2\or
3892 \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3893 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3894 \bbl@alphnum@invalid{>9999}%
3895 \fi}
3896 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3897 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3898 {\bbl@cs{cntr@#1.4@\language}\{#5}}
3899 {\bbl@cs{cntr@#1.3@\language}\{#6}}
3900 {\bbl@cs{cntr@#1.2@\language}\{#7}}
3901 {\bbl@cs{cntr@#1.1@\language}\{#8}}
3902 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3903 \bbl@ifunset{bbl@cntr@#1.S.321@\language}\{#8}}
3904 {\bbl@cs{cntr@#1.S.321@\language}\{#8}}
3905 \fi}%
3906 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}\{#8}}
3907 \def\bbl@alphnum@invalid#1{%
3908 \bbl@error{Alphabetic numeral too large (#1)}%
3909 {Currently this is the limit.}}

```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

3910 \newcommand\localeinfo[1]{%
3911 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
3912 {\bbl@error{I've found no info for the current locale.\\%
3913 The corresponding ini file has not been loaded\\%
3914 Perhaps it doesn't exist}%
3915 {See the manual for details.}}%
3916 {\bbl@cs{csname bbl@info@#1\endcsname @\language}\{#1}}
3917 \@namedef{bbl@info@name.locale}\{lcname}
3918 \@namedef{bbl@info@tag.ini}\{lini}

```

```

3919 \@namedef{bbl@info@name.english}{\elname}
3920 \@namedef{bbl@info@name.opentype}{\lname}
3921 \@namedef{bbl@info@tag.bcp47}{\lbc} % TODO
3922 \@namedef{bbl@info@tag.opentype}{\lotf}
3923 \@namedef{bbl@info@script.name}{\esname}
3924 \@namedef{bbl@info@script.name.opentype}{\sname}
3925 \@namedef{bbl@info@script.tag.bcp47}{\sbc}
3926 \@namedef{bbl@info@script.tag.opentype}{\sotf}
3927 \let\bbl@ensureinfo\@gobble
3928 \newcommand\BabelEnsureInfo{%
3929   \ifx\InputIfFileExists\undefined\else
3930     \def\bbl@ensureinfo##1{%
3931       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}}%
3932   \fi
3933   \bbl@foreach\bbl@loaded{%
3934     \def\language{##1}%
3935     \bbl@ensureinfo{##1}}}%

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3936 \newcommand\getlocaleproperty{%
3937   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3938 \def\bbl@getproperty@s#1#2#3{%
3939   \let#1\relax
3940   \def\bbl@elt##1##2##3{%
3941     \bbl@ifsamestring{##1/##2}{##3}%
3942     {\providecommand#1{##3}%
3943     \def\bbl@elt####1####2####3{}}}%
3944   {}}%
3945   \bbl@cs{inidata@#2}}%
3946 \def\bbl@getproperty@x#1#2#3{%
3947   \bbl@getproperty@s{#1}{#2}{#3}%
3948   \ifx#1\relax
3949     \bbl@error
3950     {Unknown key for locale '#2':\%
3951     #3\%
3952     \string#1 will be set to \relax}%
3953   {Perhaps you misspelled it.}%
3954   \fi}
3955 \let\bbl@ini@loaded\@empty
3956 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel behavior

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

```

3957 \newcommand\babeladjust[1]{% TODO. Error handling.
3958   \bbl@forkv{#1}{%
3959     \bbl@ifunset{bbl@ADJ@##1@##2}%
3960     {\bbl@cs{ADJ@##1}{##2}}%
3961     {\bbl@cs{ADJ@##1@##2}}}
3962 %
3963 \def\bbl@adjust@lua#1#2{%
3964   \ifvmode
3965     \ifnum\currentgrouplevel=\z@
3966       \directlua{ Babel.#2 }%
3967       \expandafter\expandafter\expandafter\@gobble

```

```

3968 \fi
3969 \fi
3970 {\bbl@error % The error is gobbled if everything went ok.
3971 {Currently, #1 related features can be adjusted only\\%
3972 in the main vertical list.}%
3973 {Maybe things change in the future, but this is what it is.}}
3974 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3975 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3976 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3977 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3978 \@namedef{bbl@ADJ@bidi.text@on}{%
3979 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3980 \@namedef{bbl@ADJ@bidi.text@off}{%
3981 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3982 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3983 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3984 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3985 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3986 %
3987 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3988 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3989 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3990 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3991 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3992 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3993 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3994 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3995 %
3996 \def\bbl@adjust@layout#1{%
3997 \ifvmode
3998 #1%
3999 \expandafter\@gobble
4000 \fi
4001 {\bbl@error % The error is gobbled if everything went ok.
4002 {Currently, layout related features can be adjusted only\\%
4003 in vertical mode.}%
4004 {Maybe things change in the future, but this is what it is.}}
4005 \@namedef{bbl@ADJ@layout.tabular@on}{%
4006 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
4007 \@namedef{bbl@ADJ@layout.tabular@off}{%
4008 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4009 \@namedef{bbl@ADJ@layout.lists@on}{%
4010 \bbl@adjust@layout{\let\list\bbl@NL@list}}
4011 \@namedef{bbl@ADJ@layout.lists@off}{%
4012 \bbl@adjust@layout{\let\list\bbl@OL@list}}
4013 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4014 \bbl@activateposthyphen}
4015 %
4016 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4017 \bbl@bcpallowedtrue}
4018 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4019 \bbl@bcpallowedfalse}
4020 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4021 \def\bbl@bcp@prefix{#1}}
4022 \def\bbl@bcp@prefix{bcp47-}
4023 \@namedef{bbl@ADJ@autoload.options}#1{%
4024 \def\bbl@autoload@options{#1}}
4025 \let\bbl@autoload@bcptoptions\@empty
4026 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%

```

```

4027 \def\bbl@autoload@bcptoptions{#1}}
4028 \newif\ifbbl@bcptname
4029 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4030   \bbl@bcptonametrue
4031   \BabelEnsureInfo}
4032 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4033   \bbl@bcptonamefalse}
4034 % TODO: use babel name, override
4035 %
4036 % As the final task, load the code for lua.
4037 %
4038 \ifx\directlua\@undefined\else
4039   \ifx\bbl@luapatterns\@undefined
4040     \input luababel.def
4041   \fi
4042 \fi
4043 </core>

A proxy file for switch.def

4044 <*kernel>
4045 \let\bbl@onlyswitch\@empty
4046 \input babel.def
4047 \let\bbl@onlyswitch\@undefined
4048 </kernel>
4049 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by $\text{\texttt{iniT\TeX}}$ because it should instruct $\text{\texttt{T\TeX}}$ 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.

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.

```

4050 <<Make sure ProvidesFile is defined>>
4051 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
4052 \xdef\bbl@format{\jobname}
4053 \def\bbl@version{<<version>>}
4054 \def\bbl@date{<<date>>}
4055 \ifx\AtBeginDocument\@undefined
4056   \def\@empty{}
4057   \let\orig@dump\dump
4058   \def\dump{%
4059     \ifx\@ztryfc\@undefined
4060     \else
4061       \toks0=\expandafter{\@preamblecmds}%
4062       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4063       \def\@begindocumenthook{}%
4064     \fi
4065     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4066 \fi
4067 <<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.

```

4068 \def\process@line#1#2 #3 #4 {%
4069   \ifx=#1%
4070     \process@synonym{#2}%
4071   \else
4072     \process@language{#1#2}{#3}{#4}%
4073   \fi
4074   \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.

```

4075 \toks@{}
4076 \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.

```

4077 \def\process@synonym#1{%
4078   \ifnum\last@language=\m@ne
4079     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4080   \else
4081     \expandafter\chardef\csname l@#1\endcsname\last@language
4082     \wlog{\string\l@#1=\string\language\the\last@language}%
4083     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
4084     \csname\language\endcsname\hyphenmins\endcsname
4085     \let\bbl@elt\relax
4086     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4087   \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 `\langhyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` and `\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.

```

4088 \def\process@language#1#2#3{%
4089   \expandafter\addlanguage\csname l@#1\endcsname
4090   \expandafter\language\csname l@#1\endcsname
4091   \edef\language#1{%
4092     \bbl@hook@everylanguage{#1}%
4093     % > luatex
4094     \bbl@get@enc#1::\@@@
4095     \begingroup
4096       \lefthyphenmin\m@ne
4097       \bbl@hook@loadpatterns{#2}%
4098       % > luatex
4099       \ifnum\lefthyphenmin=\m@ne
4100       \else
4101         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4102           \the\lefthyphenmin\the\righthyphenmin}%
4103       \fi
4104     \endgroup
4105   \def\bbl@tempa{#3}%
4106   \ifx\bbl@tempa\@empty\else
4107     \bbl@hook@loadexceptions{#3}%
4108     % > luatex
4109   \fi
4110   \let\bbl@elt\relax
4111   \edef\bbl@languages{%
4112     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4113   \ifnum\the\language=\z@
4114     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4115       \set@hyphenmins\tw@\thr@@\relax
4116     \else
4117       \expandafter\expandafter\expandafter\set@hyphenmins
4118       \csname #1hyphenmins\endcsname
4119     \fi
4120     \the\toks@
4121     \toks@{}%
4122   \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.

```

4123 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides `luatex`, format-specific configuration files are taken into account. `loadkernel` currently loads nothing, but define some basic macros instead.

```

4124 \def\bbl@hook@everylanguage#1{}
4125 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4126 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4127 \def\bbl@hook@loadkernel#1{%
4128   \def\addlanguage{\csname newlanguage\endcsname}%

```

```

4129 \def\adddialect##1##2{%
4130   \global\chardef##1##2\relax
4131   \wlog{\string##1 = a dialect from \string\language##2}}%
4132 \def\iflanguage##1{%
4133   \expandafter\ifx\csname l@##1\endcsname\relax
4134     \@nolanerr{##1}%
4135   \else
4136     \ifnum\csname l@##1\endcsname=\language
4137       \expandafter\expandafter\expandafter\@firstoftwo
4138     \else
4139       \expandafter\expandafter\expandafter\@secondoftwo
4140     \fi
4141   \fi}%
4142 \def\providehyphenmins##1##2{%
4143   \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4144     \@namedef{##1hyphenmins}{##2}%
4145   \fi}%
4146 \def\set@hyphenmins##1##2{%
4147   \lefthyphenmin##1\relax
4148   \righthyphenmin##2\relax}%
4149 \def\selectlanguage{%
4150   \errhelp{Selecting a language requires a package supporting it}%
4151   \errmessage{Not loaded}}%
4152 \let\foreignlanguage\selectlanguage
4153 \let\otherlanguage\selectlanguage
4154 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4155 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4156 \def\setlocale{%
4157   \errhelp{Find an armchair, sit down and wait}%
4158   \errmessage{Not yet available}}%
4159 \let\uselocale\setlocale
4160 \let\locale\setlocale
4161 \let\selectlocale\setlocale
4162 \let\localename\setlocale
4163 \let\textlocale\setlocale
4164 \let\textlanguage\setlocale
4165 \let\languagetext\setlocale}
4166 \begingroup
4167 \def\AddBabelHook#1#2{%
4168   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4169     \def\next{\toks1}%
4170   \else
4171     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4172   \fi
4173   \next}
4174 \ifx\directlua\@undefined
4175   \ifx\XeTeXinputencoding\@undefined\else
4176     \input xebabel.def
4177   \fi
4178 \else
4179   \input luababel.def
4180 \fi
4181 \openin1 = babel-\bbl@format.cfg
4182 \ifeof1
4183 \else
4184   \input babel-\bbl@format.cfg\relax
4185 \fi
4186 \closein1
4187 \endgroup

```



```
4188 \bbl@hook@loadkernel{switch.def}
```

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

```
4189 \openin1 = language.dat
```

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

```
4190 \def\language{english}%
4191 \ifeof1
4192   \message{I couldn't find the file language.dat,\space
4193           I will try the file hyphen.tex}
4194   \input hyphen.tex\relax
4195   \chardef\l@english\z@
4196 \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`.

```
4197   \last@language\m@ne
```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
4198   \loop
4199     \endlinechar\m@ne
4200     \read1 to \bbl@line
4201     \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.

```
4202     \if T\ifeof1F\fi T\relax
4203     \ifx\bbl@line\@empty\else
4204       \edef\bbl@line{\bbl@line\space\space\space}%
4205       \expandafter\process@line\bbl@line\relax
4206     \fi
4207   \repeat
```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns, and close the configuration file.

```
4208   \begingroup
4209     \def\bbl@elt#1#2#3#4{%
4210       \global\language=#2\relax
4211       \gdef\language{#1}%
4212       \def\bbl@elt##1##2##3##4{}}%
4213   \bbl@languages
4214   \endgroup
4215 \fi
4216 \closein1
```

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

```
4217 \if\the\toks@/\else
4218   \errhelp{language.dat loads no language, only synonyms}
4219   \errmessage{Orphan language synonym}
4220 \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.

```
4221 \let\bbl@line\@undefined
4222 \let\process@line\@undefined
4223 \let\process@synonym\@undefined
4224 \let\process@language\@undefined
4225 \let\bbl@get@enc\@undefined
4226 \let\bbl@hyph@enc\@undefined
4227 \let\bbl@tempa\@undefined
4228 \let\bbl@hook@loadkernel\@undefined
4229 \let\bbl@hook@everylanguage\@undefined
4230 \let\bbl@hook@loadpatterns\@undefined
4231 \let\bbl@hook@loadexceptions\@undefined
4232 \let\patterns\@undefined
```

Here the code for iniT_EX ends.

12 Font handling with fontspec

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

```
4233 <<(*More package options)>> ≡
4234 \chardef\bbl@bidimode\z@
4235 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4236 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4237 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4238 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4239 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4240 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4241 <</More package options>>
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. bbl@font replaces hardcoded font names inside \..family by the corresponding macro \..default.

At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading message, which is replaced by a more explanatory one.

```
4242 <<(*Font selection)>> ≡
4243 \bbl@trace{Font handling with fontspec}
4244 \ifx\ExplSyntaxOn\@undefined\else
4245   \ExplSyntaxOn
4246   \catcode`\ =10
4247   \def\bbl@loadfontspec{%
4248     \usepackage{fontspec}%
4249     \expandafter
4250     \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4251       Font '\l_fontspec_fontname_tl' is using the\\%
4252       default features for language '##1'.\\%
4253       That's usually fine, because many languages\\%
4254       require no specific features, but if the output is\\%
4255       not as expected, consider selecting another font.}
4256     \expandafter
4257     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%
```

```

4258     Font '\l_fontspec_fontname_tl' is using the\\%
4259     default features for script '##2'.\\%
4260     That's not always wrong, but if the output is\\%
4261     not as expected, consider selecting another font.}}
4262 \ExplSyntaxOff
4263 \fi
4264 \@onlypreamble\babelfont
4265 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4266 \bbl@foreach{#1}{\%
4267 \expandafter\ifx\csname date##1\endcsname\relax
4268 \IfFileExists{babel-##1.tex}%
4269 {\babelprovide{##1}}%
4270 }%
4271 \fi}%
4272 \edef\bbl@tempa{#1}%
4273 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4274 \ifx\fontspec@undefined
4275 \bbl@loadfontspec
4276 \fi
4277 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4278 \bbl@bblfont}
4279 \newcommand\bbl@bblfont[2][\% 1=features 2=fontname, @font=rm|sf|tt
4280 \bbl@ifunset{\bbl@tempb family}%
4281 {\bbl@providefam{\bbl@tempb}}%
4282 {\bbl@exp{%
4283 \\\bbl@sreplace\<\bbl@tempb family >%
4284 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4285 % For the default font, just in case:
4286 \bbl@ifunset{\bbl@lsys\language\name}{\bbl@provide@lsys{\language\name}}}%
4287 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4288 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4289 \bbl@exp{%
4290 \let\<\bbl@tempb dflt@\language\name>\<\bbl@tempb dflt@>%
4291 \\\bbl@font@set\<\bbl@tempb dflt@\language\name>%
4292 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4293 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4294 \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:

```

4295 \def\bbl@providefam#1{%
4296 \bbl@exp{%
4297 \\\newcommand\<#1default>{}% Just define it
4298 \\\bbl@add@list\\\bbl@font@fams{#1}%
4299 \\\DeclareRobustCommand\<#1family>{%
4300 \\\not@math@alphabet\<#1family>\relax
4301 \\\fontfamily\<#1default>\selectfont}%
4302 \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}

```

The following macro is activated when the hook babel-fontspec is enabled. But before we define a macro for a warning, which sets a flag to avoid duplicate them.

```

4303 \def\bbl@nostdfont#1{%
4304 \bbl@ifunset{\bbl@WFF@\f@family}%
4305 {\bbl@csarg\gdef{\WFF@\f@family}}% Flag, to avoid dupl warns
4306 \bbl@infowarn{The current font is not a babel standard family:\\%
4307 #1%
4308 \fontname\font\\%
4309 There is nothing intrinsically wrong with this warning, and\\%
4310 you can ignore it altogether if you do not need these\\%
4311 families. But if they are used in the document, you should be\\%

```

```

4312     aware 'babel' will no set Script and Language for them, so\\%
4313     you may consider defining a new family with \\string\\babelfont.\\%
4314     See the manual for further details about \\string\\babelfont.\\%
4315     Reported}}
4316     }\\}%
4317 \\gdef\\bbl@switchfont{%
4318   \\bbl@ifunset{\\bbl@sys@\\language\\name}{\\bbl@provide@sys{\\language\\name}}{\\}%
4319   \\bbl@exp{%   eg Arabic -> arabic
4320     \\lowercase{\\edef\\bbl@tempa{\\bbl@cl{\\sname}}}}}%
4321   \\bbl@foreach\\bbl@font@fams{%
4322     \\bbl@ifunset{\\bbl@##1dflt@\\language\\name}%      (1) language?
4323     {\\bbl@ifunset{\\bbl@##1dflt@*\\bbl@tempa}%      (2) from script?
4324       {\\bbl@ifunset{\\bbl@##1dflt@}%                2=F - (3) from generic?
4325         }\\}%                                          123=F - nothing!
4326         {\\bbl@exp{%                                  3=T - from generic
4327           \\global\\let<\\bbl@##1dflt@\\language\\name>%
4328             \\<\\bbl@##1dflt@>}}}%
4329         {\\bbl@exp{%                                  2=T - from script
4330           \\global\\let<\\bbl@##1dflt@\\language\\name>%
4331             \\<\\bbl@##1dflt@*\\bbl@tempa>}}}%
4332         }\\}%                                          1=T - language, already defined
4333   \\def\\bbl@tempa{\\bbl@nostdfont}\\}%
4334   \\bbl@foreach\\bbl@font@fams{%   don't gather with prev for
4335     \\bbl@ifunset{\\bbl@##1dflt@\\language\\name}%
4336     {\\bbl@cs{famrst@##1}%
4337       \\global\\bbl@csarg\\let{famrst@##1}\\relax}%
4338     {\\bbl@exp{% order is relevant
4339       \\bbl@add\\originalTeX{%
4340         \\bbl@font@rst{\\bbl@cl{\\bbl@##1dflt}}}%
4341         \\<##1default>\\<##1family>{\\bbl@##1dflt}}}%
4342         \\bbl@font@set{\\bbl@##1dflt@\\language\\name}% the main part!
4343         \\<##1default>\\<##1family>}}}%
4344   \\bbl@ifrestoring{\\bbl@tempa}\\}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \\babelfont.

```

4345 \\ifx\\f@family\\undefined\\else   % if latex
4346 \\ifcase\\bbl@engine                % if pdftex
4347   \\let\\bbl@ckeckstdfonts\\relax
4348 \\else
4349   \\def\\bbl@ckeckstdfonts{%
4350     \\begingroup
4351     \\global\\let\\bbl@ckeckstdfonts\\relax
4352     \\let\\bbl@tempa\\empty
4353     \\bbl@foreach\\bbl@font@fams{%
4354       \\bbl@ifunset{\\bbl@##1dflt@}%
4355       {\\nameuse{\\bbl@##1family}%
4356         \\bbl@csarg\\gdef{WFF@\\f@family}}{\\}% Flag
4357         \\bbl@exp{\\bbl@add\\bbl@tempa{* \\<##1family>= \\f@family\\}%
4358           \\space\\space\\fontname\\font\\}%
4359         \\bbl@csarg\\xdef{\\bbl@##1dflt@}{\\f@family}%
4360         \\expandafter\\xdef\\csname \\bbl@##1default\\endcsname{\\f@family}}}%
4361       }\\}%
4362   \\ifx\\bbl@tempa\\empty\\else
4363     \\bbl@infowarn{The following font families will use the default\\%
4364       settings for all or some languages:\\%
4365       \\bbl@tempa
4366       There is nothing intrinsically wrong with it, but\\%
4367       'babel' will no set Script and Language, which could\\%

```

```

4368         be relevant in some languages. If your document uses\\%
4369         these families, consider redefining them with \string\babelfont.\\%
4370         Reported}%
4371     \fi
4372 \endgroup}
4373 \fi
4374 \fi

```

Now the macros defining the font with fontspec.

When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.

```

4375 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4376 \bbl@xin@{<>}{#1}%
4377 \ifin@
4378 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4379 \fi
4380 \bbl@exp{%
4381 \def\\#2#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4382 \\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\bbl@tempa\relax}{}}
4383 % TODO - next should be global?, but even local does its job. I'm
4384 % still not sure -- must investigate:
4385 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4386 \let\bbl@tempa\bbl@mapselect
4387 \let\bbl@mapselect\relax
4388 \let\bbl@tempa@fam#4% eg, '\rmfamily', to be restored below
4389 \let#4\empty % Make sure \renewfontfamily is valid
4390 \bbl@exp{%
4391 \let\\bbl@tempa@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily'
4392 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4393 {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4394 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4395 {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4396 \\renewfontfamily\\#4%
4397 [\bbl@cs{lsys}\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4398 \begingroup
4399 #4%
4400 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4401 \endgroup
4402 \let#4\bbl@tempa@fam
4403 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@tempa@pfam
4404 \let\bbl@mapselect\bbl@tempa}%

```

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.

```

4405 \def\bbl@font@rst#1#2#3#4{%
4406 \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4407 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4408 \newcommand\babelFSstore[2][{}]{%
4409 \bbl@ifblank{#1}%

```

```

4410 {\bbl@csarg\def\sname@#2}{Latin}}%
4411 {\bbl@csarg\def\sname@#2}{#1}}%
4412 \bbl@provide@dirs{#2}%
4413 \bbl@csarg\ifnum{wdir@#2}>\z@
4414 \let\bbl@beforeforeign\leavevmode
4415 \EnableBabelHook{babel-bidi}%
4416 \fi
4417 \bbl@foreach{#2}{%
4418 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4419 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4420 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4421 \def\bbl@FSstore#1#2#3#4{%
4422 \bbl@csarg\edef{#2default#1}{#3}%
4423 \expandafter\addto\csname extras#1\endcsname{%
4424 \let#4#3%
4425 \ifx#3\f@family
4426 \edef#3{\csname bbl@#2default#1\endcsname}%
4427 \fontfamily{#3}\selectfont
4428 \else
4429 \edef#3{\csname bbl@#2default#1\endcsname}%
4430 \fi}%
4431 \expandafter\addto\csname noextras#1\endcsname{%
4432 \ifx#3\f@family
4433 \fontfamily{#4}\selectfont
4434 \fi
4435 \let#3#4}}
4436 \let\bbl@langfeatures\@empty
4437 \def\babelFSfeatures{% make sure \fontspec is redefined once
4438 \let\bbl@ori@fontspec\fontspec
4439 \renewcommand\fontspec[1][{}]{%
4440 \bbl@ori@fontspec[\bbl@langfeatures##1]}
4441 \let\babelFSfeatures\bbl@FSfeatures
4442 \babelFSfeatures}
4443 \def\bbl@FSfeatures#1#2{%
4444 \expandafter\addto\csname extras#1\endcsname{%
4445 \babel@save\bbl@langfeatures
4446 \edef\bbl@langfeatures{#2,}}
4447 <</Font selection>>

```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

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

```

4448 <<{*Footnote changes}>> ≡
4449 \bbl@trace{Bidi footnotes}
4450 \ifnum\bbl@bidimode>\z@
4451 \def\bbl@footnote#1#2#3{%
4452 \@ifnextchar[%
4453 {\bbl@footnote@o{#1}{#2}{#3}}%
4454 {\bbl@footnote@x{#1}{#2}{#3}}}
4455 \def\bbl@footnote@x#1#2#3#4{%
4456 \bgroup
4457 \select@language@x{\bbl@main@language}%
4458 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4459 \egroup}

```

```

4460 \def\bb1@footnote@o#1#2#3[#4]#5{%
4461   \bgroup
4462   \select@language@x{\bb1@main@language}%
4463   \bb1@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4464   \egroup}
4465 \def\bb1@footnotetext#1#2#3{%
4466   \@ifnextchar[%
4467     {\bb1@footnotetext@o{#1}{#2}{#3}}%
4468     {\bb1@footnotetext@x{#1}{#2}{#3}}}%
4469 \def\bb1@footnotetext@x#1#2#3#4{%
4470   \bgroup
4471   \select@language@x{\bb1@main@language}%
4472   \bb1@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4473   \egroup}
4474 \def\bb1@footnotetext@o#1#2#3[#4]#5{%
4475   \bgroup
4476   \select@language@x{\bb1@main@language}%
4477   \bb1@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4478   \egroup}
4479 \def\BabelFootnote#1#2#3#4{%
4480   \ifx\bb1@fn@footnote\@undefined
4481     \let\bb1@fn@footnote\footnote
4482   \fi
4483   \ifx\bb1@fn@footnotetext\@undefined
4484     \let\bb1@fn@footnotetext\footnotetext
4485   \fi
4486   \bb1@ifblank{#2}%
4487   {\def#1{\bb1@footnote{\@firstofone}{#3}{#4}}
4488    \@namedef{\bb1@stripslash#1text}%
4489    {\bb1@footnotetext{\@firstofone}{#3}{#4}}}%
4490   {\def#1{\bb1@exp{\bb1@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4491    \@namedef{\bb1@stripslash#1text}%
4492    {\bb1@exp{\bb1@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4493 \fi
4494 <</Footnote changes>>

```

Now, the code.

```

4495 (*xetex)
4496 \def\BabelStringsDefault{unicode}
4497 \let\xebbl@stop\relax
4498 \AddBabelHook{xetex}{encodedcommands}{%
4499   \def\bb1@tempa{#1}%
4500   \ifx\bb1@tempa\@empty
4501     \XeTeXinputencoding"bytes"%
4502   \else
4503     \XeTeXinputencoding"#1"%
4504   \fi
4505   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4506 \AddBabelHook{xetex}{stopcommands}{%
4507   \xebbl@stop
4508   \let\xebbl@stop\relax}
4509 \def\bb1@intraspace#1 #2 #3\@@{%
4510   \bb1@csarg\gdef{xeisp@\languagename}%
4511   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4512 \def\bb1@intrapenalty#1\@@{%
4513   \bb1@csarg\gdef{xeipn@\languagename}%
4514   {\XeTeXlinebreakpenalty #1\relax}}
4515 \def\bb1@provide@intraspace{%
4516   \bb1@xin@{\bb1@cl{lnbrk}}{s}%

```

```

4517 \ifin@else\bblexin@{\bbcl{lnbrk}}{c}\fi
4518 \ifin@
4519 \bb@ifunset{\bb@intsp@{language}}{ }%
4520 {\expandafter\ifx\csname \bb@intsp@{language}\endcsname\@empty\else
4521 \ifx\bb@KVP@intraspace\@nil
4522 \bb@exp{%
4523 \\\bb@intraspace\bbcl{intsp}\@@}%
4524 \fi
4525 \ifx\bb@KVP@intrapenalty\@nil
4526 \bb@intrapenalty0\@@
4527 \fi
4528 \fi
4529 \ifx\bb@KVP@intraspace\@nil\else % We may override the ini
4530 \expandafter\bb@intraspace\bb@KVP@intraspace\@@
4531 \fi
4532 \ifx\bb@KVP@intrapenalty\@nil\else
4533 \expandafter\bb@intrapenalty\bb@KVP@intrapenalty\@@
4534 \fi
4535 \bb@exp{%
4536 \\\bb@add\<extras\language>{%
4537 \XeTeXlinebreaklocale "\bbcl{lbcpr}"%
4538 \<\bb@xeisp@{language}>%
4539 \<\bb@xeipn@{language}>%
4540 \\\bb@tglobal\<extras\language>%
4541 \\\bb@add\<noextras\language>{%
4542 \XeTeXlinebreaklocale "en"%
4543 \\\bb@tglobal\<noextras\language>}%
4544 \ifx\bb@ispacesize\@undefined
4545 \gdef\bb@ispacesize{\bbcl{xeisp}}%
4546 \ifx\AtBeginDocument\@notprerr
4547 \expandafter\@secondoftwo % to execute right now
4548 \fi
4549 \AtBeginDocument{%
4550 \expandafter\bb@add
4551 \csname selectfont \endcsname{\bb@ispacesize}%
4552 \expandafter\bb@tglobal\csname selectfont \endcsname}%
4553 \fi}%
4554 \fi}
4555 \ifx\DisableBabelHook\@undefined\endinput\fi
4556 \AddBabelHook{babel-fontspec}{afterextras}{\bb@switchfont}
4557 \AddBabelHook{babel-fontspec}{beforestart}{\bb@ckeckstdfonts}
4558 \DisableBabelHook{babel-fontspec}
4559 <<Font selection>>
4560 \input txtbabel.def
4561 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

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

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

```
4562 <*\texet>
```



```

4563 \providecommand\bbl@provide@intraspace{}
4564 \bbl@trace{Redefinitions for bidi layout}
4565 \def\bbl@sspre@caption{%
4566   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4567 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4568 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4569 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4570 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4571   \def\@hangfrom#1{%
4572     \setbox\@tempboxa\hbox{#1}%
4573     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4574     \noindent\box\@tempboxa}
4575 \def\raggedright{%
4576   \let\@centercr
4577   \bbl@startskip\z@skip
4578   \@rightskip\@flushglue
4579   \bbl@endskip\@rightskip
4580   \parindent\z@
4581   \parfillskip\bbl@startskip}
4582 \def\raggedleft{%
4583   \let\@centercr
4584   \bbl@startskip\@flushglue
4585   \bbl@endskip\z@skip
4586   \parindent\z@
4587   \parfillskip\bbl@endskip}
4588 \fi
4589 \IfBabelLayout{lists}
4590 {\bbl@sreplace\list
4591   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4592   \def\bbl@listleftmargin{%
4593     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4594   \ifcase\bbl@engine
4595     \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4596     \def\p@enumiii{\p@enumii}\theenumii}%
4597   \fi
4598   \bbl@sreplace\@verbatim
4599   {\leftskip\@totalleftmargin}%
4600   {\bbl@startskip\textwidth
4601     \advance\bbl@startskip-\linewidth}%
4602   \bbl@sreplace\@verbatim
4603   {\rightskip\z@skip}%
4604   {\bbl@endskip\z@skip}}%
4605 {}
4606 \IfBabelLayout{contents}
4607 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4608   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4609 {}
4610 \IfBabelLayout{columns}
4611 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4612   \def\bbl@outputbox#1{%
4613     \hb@xt@\textwidth{%
4614       \hskip\columnwidth
4615       \hfil
4616       {\normalcolor\vrule \@width\columnseprule}%
4617       \hfil
4618       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4619       \hskip-\textwidth
4620       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4621       \hskip\columnsep

```

```

4622 \hskip\columnwidth}}}%
4623 {}
4624 <<Footnote changes>>
4625 \IfBabelLayout{footnotes}%
4626 {\BabelFootnote\footnote\language\language{}{}}%
4627 \BabelFootnote\localfootnote\language\language{}{}}%
4628 \BabelFootnote\mainfootnote{}{}{}}
4629 {}

```

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

```

4630 \IfBabelLayout{counters}%
4631 {\let\bb1@latin@arabic=\@arabic
4632 \def\@arabic#1{\babelsublr{\bb1@latin@arabic#1}}}%
4633 \let\bb1@ascii@roman=\@roman
4634 \def\@roman#1{\babelsublr{\ensureascii{\bb1@ascii@roman#1}}}%
4635 \let\bb1@ascii@Roman=\@Roman
4636 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@ascii@Roman#1}}}{}}
4637 </texxet>

```

13.3 LuaTeX

The loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bb1@hyphendata@<num>` exists (with the names of the files read).

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

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

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

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

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). **FIX** - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the base option); (2) at `hyphen.cfg`, to modify

some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4638 (*luatex)
4639 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4640 \bbl@trace{Read language.dat}
4641 \ifx\bbl@readstream\undefined
4642   \csname newread\endcsname\bbl@readstream
4643 \fi
4644 \begingroup
4645   \toks@{}
4646   \count@ \z@ % 0=start, 1=0th, 2=normal
4647   \def\bbl@process@line#1#2 #3 #4 {%
4648     \ifx=#1%
4649       \bbl@process@synonym{#2}%
4650     \else
4651       \bbl@process@language{#1#2}{#3}{#4}%
4652     \fi
4653     \ignorespaces}
4654   \def\bbl@manylang{%
4655     \ifnum\bbl@last>\@ne
4656       \bbl@info{Non-standard hyphenation setup}%
4657     \fi
4658     \let\bbl@manylang\relax}
4659   \def\bbl@process@language#1#2#3{%
4660     \ifcase\count@
4661       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4662     \or
4663       \count@\tw@
4664     \fi
4665     \ifnum\count@=\tw@
4666       \expandafter\addlanguage\csname l@#1\endcsname
4667       \language\allocationnumber
4668       \chardef\bbl@last\allocationnumber
4669       \bbl@manylang
4670       \let\bbl@elt\relax
4671       \xdef\bbl@languages{%
4672         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4673     \fi
4674     \the\toks@
4675     \toks@{}}
4676   \def\bbl@process@synonym@aux#1#2{%
4677     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4678     \let\bbl@elt\relax
4679     \xdef\bbl@languages{%
4680       \bbl@languages\bbl@elt{#1}{#2}{}}}%
4681   \def\bbl@process@synonym#1{%
4682     \ifcase\count@
4683       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4684     \or
4685       \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{}}}%
4686     \else
4687       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4688     \fi}
4689   \ifx\bbl@languages\undefined % Just a (sensible?) guess
4690     \chardef\l@english\z@
4691     \chardef\l@USenglish\z@
4692     \chardef\bbl@last\z@
4693     \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}

```

```

4694 \gdef\bbl@languages{%
4695     \bbl@elt{english}{0}{hyphen.tex}{}%
4696     \bbl@elt{USenglish}{0}{}}
4697 \else
4698     \global\let\bbl@languages@format\bbl@languages
4699     \def\bbl@elt#1#2#3#4{% Remove all except language 0
4700         \ifnum#2>\z@\else
4701             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4702         \fi}%
4703     \xdef\bbl@languages{\bbl@languages}%
4704 \fi
4705 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4706 \bbl@languages
4707 \openin\bbl@readstream=language.dat
4708 \ifeof\bbl@readstream
4709     \bbl@warning{I couldn't find language.dat. No additional\\%
4710                 patterns loaded. Reported}%
4711 \else
4712     \loop
4713         \endlinechar\m@ne
4714         \read\bbl@readstream to \bbl@line
4715         \endlinechar\^^M
4716         \if T\ifeof\bbl@readstream F\fi T\relax
4717         \ifx\bbl@line\empty\else
4718             \edef\bbl@line{\bbl@line\space\space\space}%
4719             \expandafter\bbl@process@line\bbl@line\relax
4720         \fi
4721     \repeat
4722 \fi
4723 \endgroup
4724 \bbl@trace{Macros for reading patterns files}
4725 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
4726 \ifx\babelcatcodetablenum\undefined
4727     \ifx\newcatcodetable\undefined
4728         \def\babelcatcodetablenum{5211}
4729         \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4730     \else
4731         \newcatcodetable\babelcatcodetablenum
4732         \newcatcodetable\bbl@pattcodes
4733     \fi
4734 \else
4735     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4736 \fi
4737 \def\bbl@luapatterns#1#2{%
4738     \bbl@get@enc#1::@@@
4739     \setbox\z@\hbox\bgroup
4740     \begingroup
4741         \savecatcodetable\babelcatcodetablenum\relax
4742         \initcatcodetable\bbl@pattcodes\relax
4743         \catcodetable\bbl@pattcodes\relax
4744         \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4745         \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\-=13
4746         \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4747         \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4748         \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4749         \catcode\`=12 \catcode\`=12 \catcode\`=12
4750         \input #1\relax
4751         \catcodetable\babelcatcodetablenum\relax
4752     \endgroup

```

```

4753 \def\bbl@tempa{#2}%
4754 \ifx\bbl@tempa@empty\else
4755 \input #2\relax
4756 \fi
4757 \egroup}%
4758 \def\bbl@patterns@lua#1{%
4759 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4760 \csname l@#1\endcsname
4761 \edef\bbl@tempa{#1}%
4762 \else
4763 \csname l@#1:\f@encoding\endcsname
4764 \edef\bbl@tempa{#1:\f@encoding}%
4765 \fi\relax
4766 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4767 \@ifundefined{bbl@hyphendata@the\language}%
4768 {\def\bbl@elt##1##2##3##4{%
4769 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4770 \def\bbl@tempb{##3}%
4771 \ifx\bbl@tempb@empty\else % if not a synonymous
4772 \def\bbl@tempc{{##3}{##4}}%
4773 \fi
4774 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4775 \fi}%
4776 \bbl@languages
4777 \@ifundefined{bbl@hyphendata@the\language}%
4778 {\bbl@info{No hyphenation patterns were set for\%
4779 language '\bbl@tempa'. Reported}}%
4780 {\expandafter\expandafter\expandafter\bbl@luapatterns
4781 \csname bbl@hyphendata@the\language\endcsname}}}%
4782 \endinput\fi
4783 % Here ends \ifx\AddBabelHook\@undefined
4784 % A few lines are only read by hyphen.cfg
4785 \ifx\DisableBabelHook\@undefined
4786 \AddBabelHook{luatex}{everylanguage}{%
4787 \def\process@language##1##2##3{%
4788 \def\process@line####1####2 ####3 ####4 {}}}
4789 \AddBabelHook{luatex}{loadpatterns}{%
4790 \input #1\relax
4791 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4792 {{#1}}}%
4793 \AddBabelHook{luatex}{loadexceptions}{%
4794 \input #1\relax
4795 \def\bbl@tempb##1##2{{##1}{##2}}%
4796 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4797 {\expandafter\expandafter\expandafter\bbl@tempb
4798 \csname bbl@hyphendata@the\language\endcsname}}
4799 \endinput\fi
4800 % Here stops reading code for hyphen.cfg
4801 % The following is read the 2nd time it's loaded
4802 \begingroup
4803 \catcode`\%=12
4804 \catcode`\'=12
4805 \catcode`\\"=12
4806 \catcode`\:=12
4807 \directlua{
4808 Babel = Babel or {}
4809 function Babel.bytes(line)
4810 return line:gsub("(.)",
4811 function (chr) return unicode.utf8.char(string.byte(chr)) end)

```

```

4812 end
4813 function Babel.begin_process_input()
4814   if luatexbase and luatexbase.add_to_callback then
4815     luatexbase.add_to_callback('process_input_buffer',
4816                               Babel.bytes, 'Babel.bytes')
4817   else
4818     Babel.callback = callback.find('process_input_buffer')
4819     callback.register('process_input_buffer', Babel.bytes)
4820   end
4821 end
4822 function Babel.end_process_input ()
4823   if luatexbase and luatexbase.remove_from_callback then
4824     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4825   else
4826     callback.register('process_input_buffer', Babel.callback)
4827   end
4828 end
4829 function Babel.addpatterns(pp, lg)
4830   local lg = lang.new(lg)
4831   local pats = lang.patterns(lg) or ''
4832   lang.clear_patterns(lg)
4833   for p in pp:gmatch('[^%s]+') do
4834     ss = ''
4835     for i in string.utfcharacters(p:gsub('%d', '')) do
4836       ss = ss .. '%d?' .. i
4837     end
4838     ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4839     ss = ss:gsub('%.%%d%?$', '%%.')
4840     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4841     if n == 0 then
4842       tex.sprint(
4843         [[\string\csname\space bbl@info\endcsname{New pattern: }
4844         .. p .. [{}]]])
4845       pats = pats .. ' ' .. p
4846     else
4847       tex.sprint(
4848         [[\string\csname\space bbl@info\endcsname{Renew pattern: }
4849         .. p .. [{}]]])
4850     end
4851   end
4852   lang.patterns(lg, pats)
4853 end
4854 }
4855 \endgroup
4856 \ifx\newattribute\@undefined\else
4857   \newattribute\bbl@attr@locale
4858   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4859   \AddBabelHook{luatex}{beforeextras}{%
4860     \setattribute\bbl@attr@locale\localeid}
4861 \fi
4862 \def\BabelStringsDefault{unicode}
4863 \let\luabbl@stop\relax
4864 \AddBabelHook{luatex}{encodedcommands}{%
4865   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4866   \ifx\bbl@tempa\bbl@tempb\else
4867     \directlua{Babel.begin_process_input()}%
4868     \def\luabbl@stop{%
4869       \directlua{Babel.end_process_input()}}%
4870   \fi}%

```

```

4871 \AddBabelHook{luatex}{stopcommands}{%
4872   \luabb1@stop
4873   \let\luabb1@stop\relax}
4874 \AddBabelHook{luatex}{patterns}{%
4875   \@ifundefined{bbl@hyphendata@the\language}%
4876     {\def\bbl@elt##1##2##3##4{%
4877       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4878       \def\bbl@tempb{##3}%
4879       \ifx\bbl@tempb\@empty\else % if not a synonymous
4880         \def\bbl@tempc{##3}{##4}}%
4881       \fi
4882       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4883     \fi}%
4884   \bbl@languages
4885   \@ifundefined{bbl@hyphendata@the\language}%
4886     {\bbl@info{No hyphenation patterns were set for\%
4887       language '#2'. Reported}}%
4888     {\expandafter\expandafter\expandafter\bbl@luapatterns
4889       \csname bbl@hyphendata@the\language\endcsname}}}%
4890   \@ifundefined{bbl@patterns@}{}%
4891   \begingroup
4892     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4893     \ifin@else
4894       \ifx\bbl@patterns@\@empty\else
4895         \directlua{ Babel.addpatterns(
4896           [[\bbl@patterns@]], \number\language) }%
4897       \fi
4898       \@ifundefined{bbl@patterns@#1}%
4899         \@empty
4900         {\directlua{ Babel.addpatterns(
4901           [[\space\csname bbl@patterns@#1\endcsname]],
4902           \number\language) }}%
4903       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4904     \fi
4905   \endgroup}%
4906   \bbl@exp{%
4907     \bbl@ifunset{bbl@prehc@\language\name}{}%
4908     {\bbl@ifblank{\bbl@cs{prehc@\language\name}}{}}%
4909     {\prehyphenchar=\bbl@c1{prehc}\relax}}}%

```

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

```

4910 \@onlypreamble\babelpatterns
4911 \AtEndOfPackage{%
4912   \newcommand\babelpatterns[2][\@empty]{%
4913     \ifx\bbl@patterns@\relax
4914       \let\bbl@patterns@\@empty
4915     \fi
4916     \ifx\bbl@pttnlist\@empty\else
4917       \bbl@warning{%
4918         You must not intermingle \string\selectlanguage\space and\%
4919         \string\babelpatterns\space or some patterns will not\%
4920         be taken into account. Reported}%
4921       \fi
4922       \ifx\@empty#1%
4923         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4924       \else

```

```

4925 \edef\bbl@tempb{\zap@space#1 \@empty}%
4926 \bbl@for\bbl@tempa\bbl@tempb{%
4927   \bbl@fixname\bbl@tempa
4928   \bbl@iflanguage\bbl@tempa{%
4929     \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4930       \ifundefined\bbl@patterns@\bbl@tempa}%
4931       \@empty
4932       {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4933       #2}}}%
4934 \fi}}

```

13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

In progress. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4935 \directlua{
4936   Babel = Babel or {}
4937   Babel.linebreaking = Babel.linebreaking or {}
4938   Babel.linebreaking.before = {}
4939   Babel.linebreaking.after = {}
4940   Babel.locale = {} % Free to use, indexed with \localeid
4941   function Babel.linebreaking.add_before(func)
4942     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
4943     table.insert(Babel.linebreaking.before , func)
4944   end
4945   function Babel.linebreaking.add_after(func)
4946     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
4947     table.insert(Babel.linebreaking.after, func)
4948   end
4949 }
4950 \def\bbl@intraspace#1 #2 #3\@@{%
4951   \directlua{
4952     Babel = Babel or {}
4953     Babel.intraspaces = Babel.intraspaces or {}
4954     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
4955       {b = #1, p = #2, m = #3}
4956     Babel.locale_props[\the\localeid].intraspace = %
4957       {b = #1, p = #2, m = #3}
4958   }}
4959 \def\bbl@intrapenalty#1\@@{%
4960   \directlua{
4961     Babel = Babel or {}
4962     Babel.intrapenalties = Babel.intrapenalties or {}
4963     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4964     Babel.locale_props[\the\localeid].intrapenalty = #1
4965   }}
4966 \begingroup
4967 \catcode`\%=12
4968 \catcode`\^=14
4969 \catcode`\'=12
4970 \catcode`\~=12
4971 \gdef\bbl@seaintraspace{^
4972   \let\bbl@seaintraspace\relax
4973   \directlua{
4974     Babel = Babel or {}

```



```

4975 Babel.sea_enabled = true
4976 Babel.sea_ranges = Babel.sea_ranges or {}
4977 function Babel.set_chranges (script, chrng)
4978     local c = 0
4979     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
4980         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4981         c = c + 1
4982     end
4983 end
4984 function Babel.sea_disc_to_space (head)
4985     local sea_ranges = Babel.sea_ranges
4986     local last_char = nil
4987     local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4988     for item in node.traverse(head) do
4989         local i = item.id
4990         if i == node.id'glyph' then
4991             last_char = item
4992         elseif i == 7 and item.subtype == 3 and last_char
4993             and last_char.char > 0x0C99 then
4994             quad = font.getfont(last_char.font).size
4995             for lg, rg in pairs(sea_ranges) do
4996                 if last_char.char > rg[1] and last_char.char < rg[2] then
4997                     lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
4998                     local intraspace = Babel.intraspaces[lg]
4999                     local intrapenalty = Babel.intrapenalties[lg]
5000                     local n
5001                     if intrapenalty ~= 0 then
5002                         n = node.new(14, 0) ^^ penalty
5003                         n.penalty = intrapenalty
5004                         node.insert_before(head, item, n)
5005                     end
5006                     n = node.new(12, 13) ^^ (glue, spaceskip)
5007                     node.setglue(n, intraspace.b * quad,
5008                                 intraspace.p * quad,
5009                                 intraspace.m * quad)
5010                     node.insert_before(head, item, n)
5011                     node.remove(head, item)
5012                 end
5013             end
5014         end
5015     end
5016 end
5017 }^^
5018 \bbl@luahyphenate}
5019 \catcode`\%=14
5020 \gdef\bbl@cjkintraspacespace{%
5021     \let\bbl@cjkintraspacespace\relax
5022     \directlua{
5023         Babel = Babel or {}
5024         require'babel-data-cjk.lua'
5025         Babel.cjk_enabled = true
5026         function Babel.cjk_linebreak(head)
5027             local GLYPH = node.id'glyph'
5028             local last_char = nil
5029             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5030             local last_class = nil
5031             local last_lang = nil
5032
5033             for item in node.traverse(head) do

```

```

5034     if item.id == GLYPH then
5035
5036         local lang = item.lang
5037
5038         local LOCALE = node.get_attribute(item,
5039             luatexbase.registernumber'bbl@attr@locale')
5040         local props = Babel.locale_props[LOCALE]
5041
5042         local class = Babel.cjk_class[item.char].c
5043
5044         if class == 'cp' then class = 'cl' end % )] as CL
5045         if class == 'id' then class = 'I' end
5046
5047         local br = 0
5048         if class and last_class and Babel.cjk_breaks[last_class][class] then
5049             br = Babel.cjk_breaks[last_class][class]
5050         end
5051
5052         if br == 1 and props.linebreak == 'c' and
5053             lang ~= \the\l@nohyphenation\space and
5054             last_lang ~= \the\l@nohyphenation then
5055             local intrapenalty = props.intrapenalty
5056             if intrapenalty ~= 0 then
5057                 local n = node.new(14, 0) % penalty
5058                 n.penalty = intrapenalty
5059                 node.insert_before(head, item, n)
5060             end
5061             local intraspace = props.intraspace
5062             local n = node.new(12, 13) % (glue, spaceskip)
5063             node.setglue(n, intraspace.b * quad,
5064                 intraspace.p * quad,
5065                 intraspace.m * quad)
5066             node.insert_before(head, item, n)
5067         end
5068
5069         quad = font.getfont(item.font).size
5070         last_class = class
5071         last_lang = lang
5072         else % if penalty, glue or anything else
5073             last_class = nil
5074         end
5075     end
5076     lang.hyphenate(head)
5077 end
5078 }%
5079 \bbl@luahyphenate}
5080 \gdef\bbl@luahyphenate{%
5081 \let\bbl@luahyphenate\relax
5082 \directlua{
5083     luatexbase.add_to_callback('hyphenate',
5084         function (head, tail)
5085             if Babel.linebreaking.before then
5086                 for k, func in ipairs(Babel.linebreaking.before) do
5087                     func(head)
5088                 end
5089             end
5090             if Babel.cjk_enabled then
5091                 Babel.cjk_linebreak(head)
5092             end

```

```

5093 lang.hyphenate(head)
5094 if Babel.linebreaking.after then
5095     for k, func in ipairs(Babel.linebreaking.after) do
5096         func(head)
5097     end
5098 end
5099 if Babel.sea_enabled then
5100     Babel.sea_disc_to_space(head)
5101 end
5102 end,
5103 'Babel.hyphenate')
5104 }
5105 }
5106 \endgroup
5107 \def\bbl@provide@intraspace{%
5108     \bbl@ifunset\bbl@intsp@{\languagename}{}%
5109     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5110         \bbl@xin@{\bbl@cl{\lnbrk}}{c}%
5111         \ifin@           % cjk
5112         \bbl@cjk@intraspace
5113         \directlua{
5114             Babel = Babel or {}
5115             Babel.locale_props = Babel.locale_props or {}
5116             Babel.locale_props[\the\localeid].linebreak = 'c'
5117         }%
5118         \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5119         \ifx\bbl@KVP@intrapenalty\@nil
5120             \bbl@intrapenalty0\@
5121         \fi
5122     \else           % sea
5123         \bbl@sea@intraspace
5124         \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5125         \directlua{
5126             Babel = Babel or {}
5127             Babel.sea_ranges = Babel.sea_ranges or {}
5128             Babel.set_chranges('\bbl@cl{sbc} ',
5129                               '\bbl@cl{chrng}')
5130         }%
5131         \ifx\bbl@KVP@intrapenalty\@nil
5132             \bbl@intrapenalty0\@
5133         \fi
5134     \fi
5135 \fi
5136 \ifx\bbl@KVP@intrapenalty\@nil\else
5137     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5138 \fi}}

```

13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

Work in progress.

Common stuff.

```
5139 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5140 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5141 \DisableBabelHook{babel-fontspec}
5142 <<Font selection>>
```

13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5143 \directlua{
5144 Babel.script_blocks = {
5145   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5146             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5147   ['Armn'] = {{0x0530, 0x058F}},
5148   ['Beng'] = {{0x0980, 0x09FF}},
5149   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5150   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5151   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5152             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5153   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5154   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5155             {0xAB00, 0xAB2F}},
5156   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5157   % Don't follow strictly Unicode, which places some Coptic letters in
5158   % the 'Greek and Coptic' block
5159   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5160   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5161             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5162             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5163             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5164             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5165             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5166   ['Hebr'] = {{0x0590, 0x05FF}},
5167   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5168             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5169   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5170   ['Knda'] = {{0x0C80, 0x0CFF}},
5171   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5172             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5173             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5174   ['Lao0'] = {{0x0E80, 0x0EFF}},
5175   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5176             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5177             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5178   ['Mahj'] = {{0x11150, 0x1117F}},
5179   ['Mlym'] = {{0x0D00, 0x0D7F}},
5180   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5181   ['Orya'] = {{0x0B00, 0x0B7F}},
5182   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5183   ['Syr1'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
```

```

5184 ['Taml'] = {{0x0B80, 0x0BFF}},
5185 ['Telu'] = {{0x0C00, 0x0C7F}},
5186 ['Tfng'] = {{0x2D30, 0x2D7F}},
5187 ['Thai'] = {{0x0E00, 0x0E7F}},
5188 ['Tibt'] = {{0x0F00, 0x0FFF}},
5189 ['Vaii'] = {{0xA500, 0xA63F}},
5190 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5191 }
5192
5193 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5194 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5195 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5196
5197 function Babel.locale_map(head)
5198   if not Babel.locale_mapped then return head end
5199
5200   local LOCALE = luatexbase.registernumber'bb1@attr@locale'
5201   local GLYPH = node.id('glyph')
5202   local inmath = false
5203   local toloc_save
5204   for item in node.traverse(head) do
5205     local toloc
5206     if not inmath and item.id == GLYPH then
5207       % Optimization: build a table with the chars found
5208       if Babel.chr_to_loc[item.char] then
5209         toloc = Babel.chr_to_loc[item.char]
5210       else
5211         for lc, maps in pairs(Babel.loc_to_scr) do
5212           for _, rg in pairs(maps) do
5213             if item.char >= rg[1] and item.char <= rg[2] then
5214               Babel.chr_to_loc[item.char] = lc
5215               toloc = lc
5216               break
5217             end
5218           end
5219         end
5220       end
5221       % Now, take action, but treat composite chars in a different
5222       % fashion, because they 'inherit' the previous locale. Not yet
5223       % optimized.
5224       if not toloc and
5225         (item.char >= 0x0300 and item.char <= 0x036F) or
5226         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5227         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5228         toloc = toloc_save
5229       end
5230       if toloc and toloc > -1 then
5231         if Babel.locale_props[toloc].lg then
5232           item.lang = Babel.locale_props[toloc].lg
5233           node.set_attribute(item, LOCALE, toloc)
5234         end
5235         if Babel.locale_props[toloc]['/'..item.font] then
5236           item.font = Babel.locale_props[toloc]['/'..item.font]
5237         end
5238         toloc_save = toloc
5239       end
5240     elseif not inmath and item.id == 7 then
5241       item.replace = item.replace and Babel.locale_map(item.replace)
5242       item.pre = item.pre and Babel.locale_map(item.pre)

```

```

5243     item.post    = item.post and Babel.locale_map(item.post)
5244     elseif item.id == node.id'math' then
5245         inmath = (item.subtype == 0)
5246     end
5247 end
5248 return head
5249 end
5250 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

5251 \newcommand\babelcharproperty[1]{%
5252   \count@=#1\relax
5253   \ifvmode
5254     \expandafter\babel@chprop
5255   \else
5256     \babel@error{\string\babelcharproperty\space can be used only in\\%
5257       vertical mode (preamble or between paragraphs)}%
5258     {See the manual for futher info}%
5259   \fi}
5260 \newcommand\babel@chprop[3][\the\count@]{%
5261   \@tempcnta=#1\relax
5262   \babel@ifunset{\babel@chprop@#2}%
5263   {\babel@error{No property named '#2'. Allowed values are\\%
5264     direction (bc), mirror (bmg), and linebreak (lb)}%
5265     {See the manual for futher info}}%
5266   }%
5267   \loop
5268     \babel@cs{\babel@chprop@#2}{#3}%
5269   \ifnum\count@<\@tempcnta
5270     \advance\count@\@ne
5271   \repeat}
5272 \def\babel@chprop@direction#1{%
5273   \directlua{
5274     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5275     Babel.characters[\the\count@]['d'] = '#1'
5276   }}
5277 \let\babel@chprop@bc\babel@chprop@direction
5278 \def\babel@chprop@mirror#1{%
5279   \directlua{
5280     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5281     Babel.characters[\the\count@]['m'] = '\number#1'
5282   }}
5283 \let\babel@chprop@bmg\babel@chprop@mirror
5284 \def\babel@chprop@linebreak#1{%
5285   \directlua{
5286     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5287     Babel.cjk_characters[\the\count@]['c'] = '#1'
5288   }}
5289 \let\babel@chprop@lb\babel@chprop@linebreak
5290 \def\babel@chprop@locale#1{%
5291   \directlua{
5292     Babel.chr_to_loc = Babel.chr_to_loc or {}
5293     Babel.chr_to_loc[\the\count@] =
5294       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
5295   }}

```

Post-handling hyphenation patterns for non-standard rules, like `ff` to `ff-f`. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a `utf8` position. With `first`, the last byte can be the leading byte in a `utf8` sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

5296 \begingroup
5297 \catcode`\#=12
5298 \catcode`\%=12
5299 \catcode`\&=14
5300 \directlua{
5301   Babel.linebreaking.post_replacements = {}
5302   Babel.linebreaking.pre_replacements = {}
5303
5304   function Babel.str_to_nodes(fn, matches, base)
5305     local n, head, last
5306     if fn == nil then return nil end
5307     for s in string.utfvalues(fn(matches)) do
5308       if base.id == 7 then
5309         base = base.replace
5310       end
5311       n = node.copy(base)
5312       n.char = s
5313       if not head then
5314         head = n
5315       else
5316         last.next = n
5317       end
5318       last = n
5319     end
5320     return head
5321   end
5322
5323   function Babel.fetch_word(head, funct)
5324     local word_string = ''
5325     local word_nodes = {}
5326     local lang
5327     local item = head
5328     local inmath = false
5329
5330     while item do
5331
5332       if item.id == 29
5333         and not(item.char == 124) && ie, not |
5334         and not(item.char == 61) && ie, not =
5335         and not inmath
5336         and (item.lang == lang or lang == nil) then
5337         lang = lang or item.lang
5338         word_string = word_string .. unicode.utf8.char(item.char)
5339         word_nodes[#word_nodes+1] = item
5340
5341       elseif item.id == 7 and item.subtype == 2 and not inmath then

```

```

5342     word_string = word_string .. '='
5343     word_nodes[#word_nodes+1] = item
5344
5345     elseif item.id == 7 and item.subtype == 3 and not inmath then
5346         word_string = word_string .. '|'
5347         word_nodes[#word_nodes+1] = item
5348
5349     elseif item.id == 11 and item.subtype == 0 then
5350         inmath = true
5351
5352     elseif word_string == '' then
5353         &% pass
5354
5355     else
5356         return word_string, word_nodes, item, lang
5357     end
5358
5359     item = item.next
5360 end
5361 end
5362
5363 function Babel.post_hyphenate_replace(head)
5364     local u = unicode.utf8
5365     local lbkr = Babel.linebreaking.post_replacements
5366     local word_head = head
5367
5368     while true do
5369         local w, wn, nw, lang = Babel.fetch_word(word_head)
5370         if not lang then return head end
5371
5372         if not lbkr[lang] then
5373             break
5374         end
5375
5376         for k=1, #lbkr[lang] do
5377             local p = lbkr[lang][k].pattern
5378             local r = lbkr[lang][k].replace
5379
5380             while true do
5381                 local matches = { u.match(w, p) }
5382                 if #matches < 2 then break end
5383
5384                 local first = table.remove(matches, 1)
5385                 local last = table.remove(matches, #matches)
5386
5387                 &% Fix offsets, from bytes to unicode.
5388                 first = u.len(w:sub(1, first-1)) + 1
5389                 last = u.len(w:sub(1, last-1))
5390
5391                 local new &% used when inserting and removing nodes
5392                 local changed = 0
5393
5394                 &% This loop traverses the replace list and takes the
5395                 &% corresponding actions
5396                 for q = first, last do
5397                     local crep = r[q-first+1]
5398                     local char_node = wn[q]
5399                     local char_base = char_node
5400

```



```

5401     if crep and crep.data then
5402         char_base = wn[crep.data+first-1]
5403     end
5404
5405     if crep == {} then
5406         break
5407     elseif crep == nil then
5408         changed = changed + 1
5409         node.remove(head, char_node)
5410     elseif crep and (crep.pre or crep.no or crep.post) then
5411         changed = changed + 1
5412         d = node.new(7, 0)    %% (disc, discretionary)
5413         d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5414         d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5415         d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5416         d.attr = char_base.attr
5417         if crep.pre == nil then    %% TeXbook p96
5418             d.penalty = crep.penalty or tex.hyphenpenalty
5419         else
5420             d.penalty = crep.penalty or tex.exhyphenpenalty
5421         end
5422         head, new = node.insert_before(head, char_node, d)
5423         node.remove(head, char_node)
5424         if q == 1 then
5425             word_head = new
5426         end
5427     elseif crep and crep.string then
5428         changed = changed + 1
5429         local str = crep.string(matches)
5430         if str == '' then
5431             if q == 1 then
5432                 word_head = char_node.next
5433             end
5434             head, new = node.remove(head, char_node)
5435         elseif char_node.id == 29 and u.len(str) == 1 then
5436             char_node.char = string.utfvalue(str)
5437         else
5438             local n
5439             for s in string.utfvalues(str) do
5440                 if char_node.id == 7 then
5441                     log('Automatic hyphens cannot be replaced, just removed.')
5442                 else
5443                     n = node.copy(char_base)
5444                     end
5445                     n.char = s
5446                     if q == 1 then
5447                         head, new = node.insert_before(head, char_node, n)
5448                         word_head = new
5449                     else
5450                         node.insert_before(head, char_node, n)
5451                     end
5452                 end
5453             end
5454             node.remove(head, char_node)
5455             end    %% string length
5456         end    %% if char and char.string
5457     end    %% for char in match
5458     if changed > 20 then
5459         texio.write('Too many changes. Ignoring the rest.')

```

```

5460         elseif changed > 0 then
5461             w, wn, nw = Babel.fetch_word(word_head)
5462         end
5463     end
5464     end %% for match
5465     end %% for patterns
5466     word_head = nw
5467 end %% for words
5468 return head
5469 end
5470
5471 &%%&
5472 &% Preliminary code for \babelprehyphenation
5473 &% TODO. Copy paste pattern. Merge with fetch_word
5474 function Babel.fetch_subtext(head, funct)
5475     local word_string = ''
5476     local word_nodes = {}
5477     local lang
5478     local item = head
5479     local inmath = false
5480
5481     while item do
5482
5483         if item.id == 29 then
5484             local locale = node.get_attribute(item, Babel.attr_locale)
5485
5486             if not(item.char == 124) &% ie, not | = space
5487                 and not inmath
5488                 and (locale == lang or lang == nil) then
5489                 lang = lang or locale
5490                 word_string = word_string .. unicode.utf8.char(item.char)
5491                 word_nodes[#word_nodes+1] = item
5492             end
5493
5494             if item == node.tail(head) then
5495                 item = nil
5496                 return word_string, word_nodes, item, lang
5497             end
5498
5499             elseif item.id == 12 and item.subtype == 13 and not inmath then
5500                 word_string = word_string .. '|'
5501                 word_nodes[#word_nodes+1] = item
5502
5503                 if item == node.tail(head) then
5504                     item = nil
5505                     return word_string, word_nodes, item, lang
5506                 end
5507
5508             elseif item.id == 11 and item.subtype == 0 then
5509                 inmath = true
5510
5511             elseif word_string == '' then
5512                 &% pass
5513
5514             else
5515                 return word_string, word_nodes, item, lang
5516             end
5517
5518             item = item.next

```

```

5519     end
5520 end
5521
5522 %% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5523 function Babel.pre_hyphenate_replace(head)
5524     local u = unicode.utf8
5525     local lbkr = Babel.linebreaking.pre_replacements
5526     local word_head = head
5527
5528     while true do
5529         local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5530         if not lang then return head end
5531
5532         if not lbkr[lang] then
5533             break
5534         end
5535
5536         for k=1, #lbkr[lang] do
5537             local p = lbkr[lang][k].pattern
5538             local r = lbkr[lang][k].replace
5539
5540             while true do
5541                 local matches = { u.match(w, p) }
5542                 if #matches < 2 then break end
5543
5544                 local first = table.remove(matches, 1)
5545                 local last = table.remove(matches, #matches)
5546
5547                 %% Fix offsets, from bytes to unicode.
5548                 first = u.len(w:sub(1, first-1)) + 1
5549                 last = u.len(w:sub(1, last-1))
5550
5551                 local new %% used when inserting and removing nodes
5552                 local changed = 0
5553
5554                 %% This loop traverses the replace list and takes the
5555                 %% corresponding actions
5556                 for q = first, last do
5557                     local crep = r[q-first+1]
5558                     local char_node = wn[q]
5559                     local char_base = char_node
5560
5561                     if crep and crep.data then
5562                         char_base = wn[crep.data+first-1]
5563                     end
5564
5565                     if crep == {} then
5566                         break
5567                     elseif crep == nil then
5568                         changed = changed + 1
5569                         node.remove(head, char_node)
5570                     elseif crep and crep.string then
5571                         changed = changed + 1
5572                         local str = crep.string(matches)
5573                         if str == '' then
5574                             if q == 1 then
5575                                 word_head = char_node.next
5576                             end
5577                             head, new = node.remove(head, char_node)

```

```

5578         elseif char_node.id == 29 and u.len(str) == 1 then
5579             char_node.char = string.utfvalue(str)
5580         else
5581             local n
5582             for s in string.utfvalues(str) do
5583                 if char_node.id == 7 then
5584                     log('Automatic hyphens cannot be replaced, just removed.')
5585                 else
5586                     n = node.copy(char_base)
5587                 end
5588                 n.char = s
5589                 if q == 1 then
5590                     head, new = node.insert_before(head, char_node, n)
5591                     word_head = new
5592                 else
5593                     node.insert_before(head, char_node, n)
5594                 end
5595             end
5596
5597             node.remove(head, char_node)
5598         end &% string length
5599     end &% if char and char.string
5600 end &% for char in match
5601 if changed > 20 then
5602     texio.write('Too many changes. Ignoring the rest.')
5603 elseif changed > 0 then
5604     &% For one-to-one can we modify directly the
5605     &% values without re-fetching? Very likely.
5606     w, wn, nw = Babel.fetch_subtext(word_head)
5607 end
5608
5609     end &% for match
5610 end &% for patterns
5611 word_head = nw
5612 end &% for words
5613 return head
5614 end
5615 &%% end of preliminary code for \babelprehyphenation
5616
5617 &% The following functions belong to the next macro
5618
5619 &% This table stores capture maps, numbered consecutively
5620 Babel.capture_maps = {}
5621
5622 function Babel.capture_func(key, cap)
5623     local ret = "[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[" .. "]"
5624     ret = ret:gsub('{([0-9])|([^\]|+)|(-)}', Babel.capture_func_map)
5625     ret = ret:gsub("%[%[%]%.%", '')
5626     ret = ret:gsub("%.%[%[%]%.%", '')
5627     return key .. [[=function(m) return ]] .. ret .. [[ end]]
5628 end
5629
5630 function Babel.capt_map(from, mapno)
5631     return Babel.capture_maps[mapno][from] or from
5632 end
5633
5634 &% Handle the {n|abc|ABC} syntax in captures
5635 function Babel.capture_func_map(capno, from, to)
5636     local froms = {}

```

```

5637   for s in string.utfcharacters(from) do
5638       table.insert(froms, s)
5639   end
5640   local cnt = 1
5641   table.insert(Babel.capture_maps, {})
5642   local mlen = table.getn(Babel.capture_maps)
5643   for s in string.utfcharacters(to) do
5644       Babel.capture_maps[mlen][froms[cnt]] = s
5645       cnt = cnt + 1
5646   end
5647   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5648       (mlen) .. ").. " .. "[["
5649 end
5650 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ - becomes `function(m) return m[1]..m[1]..'-' end`, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to $m[1]$. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

```

5651 \catcode`\#=6
5652 \gdef\babelposthyphenation#1#2#3{&%
5653   \bbl@activateposthyphen
5654   \begingroup
5655     \def\babeltempa{\bbl@add@list\babeltempb}&%
5656     \let\babeltempb\@empty
5657     \bbl@foreach{#3}{&%
5658       \bbl@ifsamestring{##1}{remove}&%
5659       {\bbl@add@list\babeltempb{nil}}&%
5660       {\directlua{
5661         local rep = [[#1]]
5662         rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5663         rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5664         rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5665         rep = rep:gsub( '(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5666         tex.print([[\\string\babeltempa{}}] .. rep .. [{}]])
5667       }}&%
5668     \directlua{
5669       local lbkr = Babel.linebreaking.post_replacements
5670       local u = unicode.utf8
5671       &% Convert pattern:
5672       local patt = string.gsub(==[#2]==, '%s', '')
5673       if not u.find(patt, '()', nil, true) then
5674         patt = '()' .. patt .. '()'
5675       end
5676       patt = string.gsub(patt, '%(%)%^\s', '^()')
5677       patt = string.gsub(patt, '%$$(%)', '()$')
5678       texio.write('*****' .. patt)
5679       patt = u.gsub(patt, '{(.)}',
5680         function (n)
5681           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5682         end)
5683       lbkr[\\the\\csname l@#1\\endcsname] = lbkr[\\the\\csname l@#1\\endcsname] or {}

```

```

5684     table.insert(lbkr[\the\csname l@#1\endcsname],
5685                  { pattern = patt, replace = { \babeltempb } })
5686   }&%
5687 \endgroup}
5688 % TODO. Working !!! Copypaste pattern.
5689 \gdef\babelprehyphenation#1#2#3{&%
5690   \bbl@activateprehyphen
5691   \begingroup
5692     \def\babeltempa{\bbl@add@list\babeltempb}&%
5693     \let\babeltempb\@empty
5694     \bbl@foreach{#3}{&%
5695       \bbl@ifsamestring{##1}{remove}&%
5696       {\bbl@add@list\babeltempb{nil}}&%
5697       {\directlua{
5698         local rep = {[##1]}
5699         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5700         tex.print([[string\babeltempa{}}] .. rep .. [[]]])
5701       }}&%
5702     \directlua{
5703       local lbkr = Babel.linebreaking.pre_replacements
5704       local u = unicode.utf8
5705       &% Convert pattern:
5706       local patt = string.gsub([=[#2]=], '%s', '')
5707       if not u.find(patt, '()', nil, true) then
5708         patt = '()' .. patt .. '()'
5709       end
5710       patt = u.gsub(patt, '{(.)}',
5711                    function (n)
5712                      return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5713                    end)
5714       lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5715       table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5716                   { pattern = patt, replace = { \babeltempb } })
5717     }&%
5718   \endgroup}
5719 \endgroup
5720 \def\bbl@activateposthyphen{%
5721   \let\bbl@activateposthyphen\relax
5722   \directlua{
5723     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5724   }}
5725 % TODO. Working !!!
5726 \def\bbl@activateprehyphen{%
5727   \let\bbl@activateprehyphen\relax
5728   \directlua{
5729     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5730   }}

```

13.7 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems to work (at least in simple cases) with array, tabularx, hline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```

5731 \bbl@trace{Redefinitions for bidi layout}
5732 \ifx\@eqnnum\undefined\else
5733   \ifx\bbl@attr@dir\undefined\else
5734     \edef\@eqnnum{%
5735       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5736       \unexpanded\expandafter{\@eqnnum}}%
5737   \fi
5738 \fi
5739 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5740 \ifnum\bbl@bidimode>\z@
5741   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5742     \bbl@exp{%
5743       \mathdir\the\bodydir
5744       #1%           Once entered in math, set boxes to restore values
5745       \<ifmmode>%
5746         \everyvbox{%
5747           \the\everyvbox
5748           \bodydir\the\bodydir
5749           \mathdir\the\mathdir
5750           \everyhbox{\the\everyhbox}%
5751           \everyvbox{\the\everyvbox}}%
5752         \everyhbox{%
5753           \the\everyhbox
5754           \bodydir\the\bodydir
5755           \mathdir\the\mathdir
5756           \everyhbox{\the\everyhbox}%
5757           \everyvbox{\the\everyvbox}}%
5758       \<fi>}}%
5759   \def\@hangfrom#1{%
5760     \setbox\@tempboxa\hbox{{#1}}%
5761     \hangindent\wd\@tempboxa
5762     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5763       \shapemode\@ne
5764     \fi
5765     \noindent\box\@tempboxa}
5766 \fi
5767 \IfBabelLayout{tabular}
5768   {\let\bbl@OL@tabular\@tabular
5769     \bbl@replace\@tabular{$}\bbl@nextfake$}%
5770   \let\bbl@NL@tabular\@tabular
5771   \AtBeginDocument{%
5772     \ifx\bbl@NL@tabular\@tabular\else
5773       \bbl@replace\@tabular{$}\bbl@nextfake$}%
5774     \let\bbl@NL@tabular\@tabular
5775   \fi}}
5776 {}
5777 \IfBabelLayout{lists}
5778   {\let\bbl@OL@list\list
5779     \bbl@sreplace\list{\parshape}\bbl@listparshape}%
5780   \let\bbl@NL@list\list
5781   \def\bbl@listparshape#1#2#3{%
5782     \parshape #1 #2 #3 %
5783     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5784       \shapemode\tw@

```

```

5785 \fi}}
5786 {}
5787 \IfBabelLayout{graphics}
5788 {\let\bb1@pictresetdir\relax
5789 \def\bb1@pictsetdir{%
5790 \ifcase\bb1@thetextdir
5791 \let\bb1@pictresetdir\relax
5792 \else
5793 \textdir TLT\relax
5794 \def\bb1@pictresetdir{\textdir TRT\relax}%
5795 \fi}%
5796 \let\bb1@OL@@picture\@picture
5797 \let\bb1@OL@put\put
5798 \bb1@sreplace\@picture{\hskip-}\{ \bb1@pictsetdir\hskip-}%
5799 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5800 \@killglue
5801 \raise#2\unitlength
5802 \hb@xt@#3\z@{\kern#1\unitlength{\bb1@pictresetdir#3}\hss}}%
5803 \AtBeginDocument
5804 {\ifx\tikz@atbegin@node\undefined\else
5805 \let\bb1@OL@pgfpicture\pgfpicture
5806 \bb1@sreplace\pgfpicture{\pgfpicturetrue}\{ \bb1@pictsetdir\pgfpicturetrue}%
5807 \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir}%
5808 \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5809 \fi}}
5810 {}

```

```

5811 \IfBabelLayout{counters}%
5812   {\let\bbl@OL@@textsuperscript\@textsuperscript
5813     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5814     \let\bbl@latinarabic=\@arabic
5815     \let\bbl@OL@@arabic\@arabic
5816     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5817     \ifpackagewith{babel}{\bidi=default}%
5818     {\let\bbl@asciroman=\@roman
5819       \let\bbl@OL@@roman\@roman
5820       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5821       \let\bbl@asciiRoman=\@Roman
5822       \let\bbl@OL@@roman\@Roman
5823       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5824       \let\bbl@OL@labelenumii\labelenumii
5825       \def\labelenumii{}\theenumii{}%
5826       \let\bbl@OL@p@enumiii\p@enumiii
5827       \def\p@enumiii{\p@enumii}\theenumii{}}{}{}%
5828   }{Footnote changes}}
5829 \IfBabelLayout{footnotes}%
5830   {\let\bbl@OL@footnote\footnote
5831     \BabelFootnote\footnote\languagenamex{}{}%
5832     \BabelFootnote\localfootnote\languagenamex{}{}%
5833     \BabelFootnote\mainfootnote{}{}{}}
5834   {}

```

```
5835 \IfBabelLayout{extras}%
5836   {\let\bbI@OL@underline\underline
```



```

5837 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5838 \let\bbl@OL@LaTeX2e\LaTeX2e
5839 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5840 \if b\expandafter\@car\@series\@nil\boldmath\fi
5841 \babelsublr{%
5842 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
5843 {}
5844 </luatex>

```

13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```

5845 <(*basic-r>
5846 Babel = Babel or {}
5847
5848 Babel.bidi_enabled = true

```

```

5849
5850 require('babel-data-bidi.lua')
5851
5852 local characters = Babel.characters
5853 local ranges = Babel.ranges
5854
5855 local DIR = node.id("dir")
5856
5857 local function dir_mark(head, from, to, outer)
5858   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5859   local d = node.new(DIR)
5860   d.dir = '+' .. dir
5861   node.insert_before(head, from, d)
5862   d = node.new(DIR)
5863   d.dir = '-' .. dir
5864   node.insert_after(head, to, d)
5865 end
5866
5867 function Babel.bidi(head, ispar)
5868   local first_n, last_n      -- first and last char with nums
5869   local last_es              -- an auxiliary 'last' used with nums
5870   local first_d, last_d      -- first and last char in L/R block
5871   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):

```

5872   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5873   local strong_lr = (strong == 'l') and 'l' or 'r'
5874   local outer = strong
5875
5876   local new_dir = false
5877   local first_dir = false
5878   local inmath = false
5879
5880   local last_lr
5881
5882   local type_n = ''
5883
5884   for item in node.traverse(head) do
5885
5886     -- three cases: glyph, dir, otherwise
5887     if item.id == node.id'glyph'
5888       or (item.id == 7 and item.subtype == 2) then
5889
5890       local itemchar
5891       if item.id == 7 and item.subtype == 2 then
5892         itemchar = item.replace.char
5893       else
5894         itemchar = item.char
5895       end
5896       local chardata = characters[itemchar]
5897       dir = chardata and chardata.d or nil
5898       if not dir then
5899         for nn, et in ipairs(ranges) do
5900           if itemchar < et[1] then
5901             break
5902           elseif itemchar <= et[2] then
5903             dir = et[3]

```

```

5904         break
5905     end
5906 end
5907 end
5908 dir = dir or 'l'
5909 if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end

```

Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a 'dir' node. We don't know the current language until then. This is not exactly true, as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute force (just above).

```

5910     if new_dir then
5911         attr_dir = 0
5912         for at in node.traverse(item.attr) do
5913             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5914                 attr_dir = at.value % 3
5915             end
5916         end
5917         if attr_dir == 1 then
5918             strong = 'r'
5919         elseif attr_dir == 2 then
5920             strong = 'al'
5921         else
5922             strong = 'l'
5923         end
5924         strong_lr = (strong == 'l') and 'l' or 'r'
5925         outer = strong_lr
5926         new_dir = false
5927     end
5928
5929     if dir == 'nsm' then dir = strong end -- W1

```

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

```

5930     dir_real = dir -- We need dir_real to set strong below
5931     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:

```

5932     if strong == 'al' then
5933         if dir == 'en' then dir = 'an' end -- W2
5934         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5935         strong_lr = 'r' -- W3
5936     end

```

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

```

5937     elseif item.id == node.id'dir' and not inmath then
5938         new_dir = true
5939         dir = nil
5940     elseif item.id == node.id'math' then
5941         inmath = (item.subtype == 0)
5942     else
5943         dir = nil -- Not a char
5944     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>.

```

5945   if dir == 'en' or dir == 'an' or dir == 'et' then
5946       if dir ~= 'et' then
5947           type_n = dir
5948       end
5949       first_n = first_n or item
5950       last_n = last_es or item
5951       last_es = nil
5952   elseif dir == 'es' and last_n then -- W3+W6
5953       last_es = item
5954   elseif dir == 'cs' then             -- it's right - do nothing
5955   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5956       if strong_lr == 'r' and type_n ~= '' then
5957           dir_mark(head, first_n, last_n, 'r')
5958       elseif strong_lr == 'l' and first_d and type_n == 'an' then
5959           dir_mark(head, first_n, last_n, 'r')
5960           dir_mark(head, first_d, last_d, outer)
5961           first_d, last_d = nil, nil
5962       elseif strong_lr == 'l' and type_n ~= '' then
5963           last_d = last_n
5964       end
5965       type_n = ''
5966       first_n, last_n = nil, nil
5967   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:

```

5968   if dir == 'l' or dir == 'r' then
5969       if dir ~= outer then
5970           first_d = first_d or item
5971           last_d = item
5972       elseif first_d and dir ~= strong_lr then
5973           dir_mark(head, first_d, last_d, outer)
5974           first_d, last_d = nil, nil
5975       end
5976   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.

```

5977   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5978       item.char = characters[item.char] and
5979           characters[item.char].m or item.char
5980   elseif (dir or new_dir) and last_lr ~= item then
5981       local mir = outer .. strong_lr .. (dir or outer)
5982       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5983           for ch in node.traverse(node.next(last_lr)) do
5984               if ch == item then break end
5985               if ch.id == node.id'glyph' and characters[ch.char] then
5986                   ch.char = characters[ch.char].m or ch.char
5987               end
5988           end

```

```

5989     end
5990 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).

```

5991 if dir == 'l' or dir == 'r' then
5992     last_lr = item
5993     strong = dir_real          -- Don't search back - best save now
5994     strong_lr = (strong == 'l') and 'l' or 'r'
5995 elseif new_dir then
5996     last_lr = nil
5997 end
5998 end

```

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

```

5999 if last_lr and outer == 'r' then
6000     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6001         if characters[ch.char] then
6002             ch.char = characters[ch.char].m or ch.char
6003         end
6004     end
6005 end
6006 if first_n then
6007     dir_mark(head, first_n, last_n, outer)
6008 end
6009 if first_d then
6010     dir_mark(head, first_d, last_d, outer)
6011 end

```

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

```

6012 return node.prev(head) or head
6013 end
6014 </basic-r>

```

And here the Lua code for bidi=basic:

```

6015 <(*basic>
6016 Babel = Babel or {}
6017
6018 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6019
6020 Babel.fontmap = Babel.fontmap or {}
6021 Babel.fontmap[0] = {}          -- l
6022 Babel.fontmap[1] = {}          -- r
6023 Babel.fontmap[2] = {}          -- al/an
6024
6025 Babel.bidi_enabled = true
6026 Babel.mirroring_enabled = true
6027
6028 require('babel-data-bidi.lua')
6029
6030 local characters = Babel.characters
6031 local ranges = Babel.ranges
6032
6033 local DIR = node.id('dir')
6034 local GLYPH = node.id('glyph')
6035
6036 local function insert_implicit(head, state, outer)
6037     local new_state = state

```

```

6038 if state.sim and state.eim and state.sim ~= state.eim then
6039     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6040     local d = node.new(DIR)
6041     d.dir = '+' .. dir
6042     node.insert_before(head, state.sim, d)
6043     local d = node.new(DIR)
6044     d.dir = '-' .. dir
6045     node.insert_after(head, state.eim, d)
6046 end
6047 new_state.sim, new_state.eim = nil, nil
6048 return head, new_state
6049 end
6050
6051 local function insert_numeric(head, state)
6052     local new
6053     local new_state = state
6054     if state.san and state.ean and state.san ~= state.ean then
6055         local d = node.new(DIR)
6056         d.dir = '+TLT'
6057         _, new = node.insert_before(head, state.san, d)
6058         if state.san == state.sim then state.sim = new end
6059         local d = node.new(DIR)
6060         d.dir = '-TLT'
6061         _, new = node.insert_after(head, state.ean, d)
6062         if state.ean == state.eim then state.eim = new end
6063     end
6064     new_state.san, new_state.ean = nil, nil
6065     return head, new_state
6066 end
6067
6068 -- TODO - \hbox with an explicit dir can lead to wrong results
6069 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6070 -- was s made to improve the situation, but the problem is the 3-dir
6071 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6072 -- well.
6073
6074 function Babel.bidi(head, ispar, hdir)
6075     local d -- d is used mainly for computations in a loop
6076     local prev_d = ''
6077     local new_d = false
6078
6079     local nodes = {}
6080     local outer_first = nil
6081     local inmath = false
6082
6083     local glue_d = nil
6084     local glue_i = nil
6085
6086     local has_en = false
6087     local first_et = nil
6088
6089     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6090
6091     local save_outer
6092     local temp = node.get_attribute(head, ATDIR)
6093     if temp then
6094         temp = temp % 3
6095         save_outer = (temp == 0 and 'l') or
6096                     (temp == 1 and 'r') or

```

```

6097             (temp == 2 and 'al')
6098     elseif ispar then             -- Or error? Shouldn't happen
6099         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6100     else                         -- Or error? Shouldn't happen
6101         save_outer = ('TRT' == hdir) and 'r' or 'l'
6102     end
6103     -- when the callback is called, we are just _after_ the box,
6104     -- and the textdir is that of the surrounding text
6105     -- if not ispar and hdir ~= tex.textdir then
6106     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6107     -- end
6108     local outer = save_outer
6109     local last = outer
6110     -- 'al' is only taken into account in the first, current loop
6111     if save_outer == 'al' then save_outer = 'r' end
6112
6113     local fontmap = Babel.fontmap
6114
6115     for item in node.traverse(head) do
6116
6117         -- In what follows, #node is the last (previous) node, because the
6118         -- current one is not added until we start processing the neutrals.
6119
6120         -- three cases: glyph, dir, otherwise
6121         if item.id == GLYPH
6122             or (item.id == 7 and item.subtype == 2) then
6123
6124             local d_font = nil
6125             local item_r
6126             if item.id == 7 and item.subtype == 2 then
6127                 item_r = item.replace    -- automatic discs have just 1 glyph
6128             else
6129                 item_r = item
6130             end
6131             local chardata = characters[item_r.char]
6132             d = chardata and chardata.d or nil
6133             if not d or d == 'nsm' then
6134                 for nn, et in ipairs(ranges) do
6135                     if item_r.char < et[1] then
6136                         break
6137                     elseif item_r.char <= et[2] then
6138                         if not d then d = et[3]
6139                         elseif d == 'nsm' then d_font = et[3]
6140                         end
6141                         break
6142                     end
6143                 end
6144             end
6145             d = d or 'l'
6146
6147             -- A short 'pause' in bidi for mapfont
6148             d_font = d_font or d
6149             d_font = (d_font == 'l' and 0) or
6150                 (d_font == 'nsm' and 0) or
6151                 (d_font == 'r' and 1) or
6152                 (d_font == 'al' and 2) or
6153                 (d_font == 'an' and 2) or nil
6154             if d_font and fontmap and fontmap[d_font][item_r.font] then
6155                 item_r.font = fontmap[d_font][item_r.font]

```

```

6156     end
6157
6158     if new_d then
6159         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6160         if inmath then
6161             attr_d = 0
6162         else
6163             attr_d = node.get_attribute(item, ATDIR)
6164             attr_d = attr_d % 3
6165         end
6166         if attr_d == 1 then
6167             outer_first = 'r'
6168             last = 'r'
6169         elseif attr_d == 2 then
6170             outer_first = 'r'
6171             last = 'al'
6172         else
6173             outer_first = 'l'
6174             last = 'l'
6175         end
6176         outer = last
6177         has_en = false
6178         first_et = nil
6179         new_d = false
6180     end
6181
6182     if glue_d then
6183         if (d == 'l' and 'l' or 'r') ~= glue_d then
6184             table.insert(nodes, {glue_i, 'on', nil})
6185         end
6186         glue_d = nil
6187         glue_i = nil
6188     end
6189
6190     elseif item.id == DIR then
6191         d = nil
6192         new_d = true
6193
6194     elseif item.id == node.id'glue' and item.subtype == 13 then
6195         glue_d = d
6196         glue_i = item
6197         d = nil
6198
6199     elseif item.id == node.id'math' then
6200         inmath = (item.subtype == 0)
6201
6202     else
6203         d = nil
6204     end
6205
6206     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6207     if last == 'al' and d == 'en' then
6208         d = 'an'          -- W3
6209     elseif last == 'al' and (d == 'et' or d == 'es') then
6210         d = 'on'          -- W6
6211     end
6212
6213     -- EN + CS/ES + EN      -- W4
6214     if d == 'en' and #nodes >= 2 then

```



```

6215     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6216         and nodes[#nodes-1][2] == 'en' then
6217         nodes[#nodes][2] = 'en'
6218     end
6219 end
6220
6221 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6222 if d == 'an' and #nodes >= 2 then
6223     if (nodes[#nodes][2] == 'cs')
6224         and nodes[#nodes-1][2] == 'an' then
6225         nodes[#nodes][2] = 'an'
6226     end
6227 end
6228
6229 -- ET/EN                -- W5 + W7->l / W6->on
6230 if d == 'et' then
6231     first_et = first_et or (#nodes + 1)
6232 elseif d == 'en' then
6233     has_en = true
6234     first_et = first_et or (#nodes + 1)
6235 elseif first_et then    -- d may be nil here !
6236     if has_en then
6237         if last == 'l' then
6238             temp = 'l'    -- W7
6239         else
6240             temp = 'en'   -- W5
6241         end
6242     else
6243         temp = 'on'      -- W6
6244     end
6245     for e = first_et, #nodes do
6246         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6247     end
6248     first_et = nil
6249     has_en = false
6250 end
6251
6252 if d then
6253     if d == 'al' then
6254         d = 'r'
6255         last = 'al'
6256     elseif d == 'l' or d == 'r' then
6257         last = d
6258     end
6259     prev_d = d
6260     table.insert(nodes, {item, d, outer_first})
6261 end
6262
6263 outer_first = nil
6264
6265 end
6266
6267 -- TODO -- repeated here in case EN/ET is the last node. Find a
6268 -- better way of doing things:
6269 if first_et then    -- dir may be nil here !
6270     if has_en then
6271         if last == 'l' then
6272             temp = 'l'    -- W7
6273         else

```

```

6274         temp = 'en'    -- W5
6275     end
6276 else
6277     temp = 'on'        -- W6
6278 end
6279 for e = first_et, #nodes do
6280     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6281 end
6282 end
6283
6284 -- dummy node, to close things
6285 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6286
6287 ----- NEUTRAL -----
6288
6289 outer = save_outer
6290 last = outer
6291
6292 local first_on = nil
6293
6294 for q = 1, #nodes do
6295     local item
6296
6297     local outer_first = nodes[q][3]
6298     outer = outer_first or outer
6299     last = outer_first or last
6300
6301     local d = nodes[q][2]
6302     if d == 'an' or d == 'en' then d = 'r' end
6303     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6304
6305     if d == 'on' then
6306         first_on = first_on or q
6307     elseif first_on then
6308         if last == d then
6309             temp = d
6310         else
6311             temp = outer
6312         end
6313         for r = first_on, q - 1 do
6314             nodes[r][2] = temp
6315             item = nodes[r][1]    -- MIRRORING
6316             if Babel.mirroring_enabled and item.id == GLYPH
6317                 and temp == 'r' and characters[item.char] then
6318                 local font_mode = font.fonts[item.font].properties.mode
6319                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6320                     item.char = characters[item.char].m or item.char
6321                 end
6322             end
6323         end
6324         first_on = nil
6325     end
6326
6327     if d == 'r' or d == 'l' then last = d end
6328 end
6329
6330 ----- IMPLICIT, REORDER -----
6331
6332 outer = save_outer

```

```

6333 last = outer
6334
6335 local state = {}
6336 state.has_r = false
6337
6338 for q = 1, #nodes do
6339
6340     local item = nodes[q][1]
6341
6342     outer = nodes[q][3] or outer
6343
6344     local d = nodes[q][2]
6345
6346     if d == 'nsm' then d = last end          -- W1
6347     if d == 'en' then d = 'an' end
6348     local isdir = (d == 'r' or d == 'l')
6349
6350     if outer == 'l' and d == 'an' then
6351         state.san = state.san or item
6352         state.ean = item
6353     elseif state.san then
6354         head, state = insert_numeric(head, state)
6355     end
6356
6357     if outer == 'l' then
6358         if d == 'an' or d == 'r' then      -- im -> implicit
6359             if d == 'r' then state.has_r = true end
6360             state.sim = state.sim or item
6361             state.eim = item
6362         elseif d == 'l' and state.sim and state.has_r then
6363             head, state = insert_implicit(head, state, outer)
6364         elseif d == 'l' then
6365             state.sim, state.eim, state.has_r = nil, nil, false
6366         end
6367     else
6368         if d == 'an' or d == 'l' then
6369             if nodes[q][3] then -- nil except after an explicit dir
6370                 state.sim = item -- so we move sim 'inside' the group
6371             else
6372                 state.sim = state.sim or item
6373             end
6374             state.eim = item
6375         elseif d == 'r' and state.sim then
6376             head, state = insert_implicit(head, state, outer)
6377         elseif d == 'r' then
6378             state.sim, state.eim = nil, nil
6379         end
6380     end
6381
6382     if isdir then
6383         last = d          -- Don't search back - best save now
6384     elseif d == 'on' and state.san then
6385         state.san = state.san or item
6386         state.ean = item
6387     end
6388
6389 end
6390
6391 return node.prev(head) or head

```

```
6392 end
6393 </basic>
```

14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

For the meaning of these codes, see the Unicode standard.

15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available. The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
6394 <*nil>
6395 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6396 \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.

```
6397 \ifx\l@nil\undefined
6398   \newlanguage\l@nil
6399   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
6400   \let\bbl@elt\relax
6401   \edef\bbl@languages{% Add it to the list of languages
6402     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
6403 \fi
```

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

```
6404 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
6405 \let\captionnil\@empty
6406 \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.

```
6407 \ldf@finish{nil}
6408 </nil>
```

16.1 Not renaming hyphen.tex

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

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`.

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that's done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.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.

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some L^AT_EX features

The following code duplicates or emulates parts of L^AT_EX 2_ε that are needed for babel.

```
6428 <<*Emulate LaTeX>> ≡
6429 % == Code for plain ==
6430 \def\@empty{}
6431 \def\loadlocalcfg#1{%
6432   \openin0#1.cfg
6433   \ifeof0
6434     \closein0
6435   \else
6436     \closein0
6437     {\immediate\write16{*****}%
6438      \immediate\write16{* Local config file #1.cfg used}%
6439      \immediate\write16{*}%
6440     }
6441     \input #1.cfg\relax
6442   \fi
6443   \@endofldf}
```

16.3 General tools

A number of L^AT_EX macro's that are needed later on.

```
6444 \long\def\@firstofone#1{#1}
6445 \long\def\@firstoftwo#1#2{#1}
6446 \long\def\@secondoftwo#1#2{#2}
6447 \def\@nnil{\@nil}
6448 \def\@gobbletwo#1#2{}
6449 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6450 \def\@star@or@long#1{%
6451   \@ifstar
6452   {\let\l@ngrel@x\relax#1}%
6453   {\let\l@ngrel@x\long#1}}
6454 \let\l@ngrel@x\relax
6455 \def\@car#1#2\@nil{#1}
6456 \def\@cdr#1#2\@nil{#2}
6457 \let\@typeset@protect\relax
6458 \let\protected@edef\edef
6459 \long\def\@gobble#1{}
6460 \edef\@backslashchar{\expandafter\@gobble\string\}
6461 \def\strip@prefix#1>{}
6462 \def\g@addto@macro#1#2{%
6463   \toks@\expandafter{#1#2}%
6464   \xdef#1{\the\toks@}}
6465 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6466 \def\@nameuse#1{\csname #1\endcsname}
6467 \def\@ifundefined#1{%
6468   \expandafter\ifx\csname#1\endcsname\relax
6469     \expandafter\@firstoftwo
6470   \else
6471     \expandafter\@secondoftwo
6472   \fi}
6473 \def\@expandtwoargs#1#2#3{%
6474   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6475 \def\zap@space#1 #2{%
6476   #1%
6477   \ifx#2\@empty\else\expandafter\zap@space\fi
6478   #2}
```

```
6479 \let\bbl@trace\@gobble
```

$\LaTeX 2_{\epsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
6480 \ifx\@preamblecmds\undefined
6481   \def\@preamblecmds{}
6482 \fi
6483 \def\@onlypreamble#1{%
6484   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6485     \@preamblecmds\do#1}}
6486 \@onlypreamble\@onlypreamble
```

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

```
6487 \def\begindocument{%
6488   \@begindocumenthook
6489   \global\let\@begindocumenthook\undefined
6490   \def\do##1{\global\let##1\undefined}%
6491   \@preamblecmds
6492   \global\let\do\noexpand}
6493 \ifx\@begindocumenthook\undefined
6494   \def\@begindocumenthook{}
6495 \fi
6496 \@onlypreamble\@begindocumenthook
6497 \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`.

```
6498 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6499 \@onlypreamble\AtEndOfPackage
6500 \def\@endofldf{}
6501 \@onlypreamble\@endofldf
6502 \let\bbl@afterlang\@empty
6503 \chardef\bbl@opt@hyphenmap\z@
```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```
6504 \catcode`\&=\z@
6505 \ifx&\if@files\@undefined
6506   \expandafter\let\csname if@files\expandafter\endcsname
6507     \csname iffalse\endcsname
6508 \fi
6509 \catcode`\&=4
```

Mimick \LaTeX 's commands to define control sequences.

```
6510 \def\newcommand{\@star@or@long\new@command}
6511 \def\new@command#1{%
6512   \@testopt{\@newcommand#1}0}
6513 \def\@newcommand#1[#2]{%
6514   \@ifnextchar [{\@xargdef#1[#2]}%
6515     {\@argdef#1[#2]}}
6516 \long\def\@argdef#1[#2]#3{%
6517   \@yargdef#1\@ne{#2}{#3}}
6518 \long\def\@xargdef#1[#2][#3]#4{%
6519   \expandafter\def\expandafter#1\expandafter{%
6520     \expandafter\@protected@testopt\expandafter #1%
6521     \csname\string#1\expandafter\endcsname{#3}}}%

```

```

6522 \expandafter\@yargdef \csname\string#1\endcsname
6523 \tw@{#2}{#4}}
6524 \long\def\@yargdef#1#2#3{%
6525 \@tempcnta#3\relax
6526 \advance \@tempcnta \@ne
6527 \let\@hash@\relax
6528 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6529 \@tempcntb #2%
6530 \@whilenum\@tempcntb <\@tempcnta
6531 \do{%
6532 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6533 \advance\@tempcntb \@ne}%
6534 \let\@hash@###
6535 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6536 \def\providecommand{\@star@or@long\provide@command}
6537 \def\provide@command#1{%
6538 \begingroup
6539 \escapechar\m@ne\xdef\@gtempa{\string#1}}%
6540 \endgroup
6541 \expandafter\@ifundefined\@gtempa
6542 {\def\reserved@a{\new@command#1}}%
6543 {\let\reserved@a\relax
6544 \def\reserved@a{\new@command\reserved@a}}%
6545 \reserved@a}%

6546 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6547 \def\declare@robustcommand#1{%
6548 \edef\reserved@a{\string#1}%
6549 \def\reserved@b{#1}%
6550 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6551 \edef#1{%
6552 \ifx\reserved@a\reserved@b
6553 \noexpand\x@protect
6554 \noexpand#1%
6555 \fi
6556 \noexpand\protect
6557 \expandafter\noexpand\csname
6558 \expandafter\@gobble\string#1 \endcsname
6559 }%
6560 \expandafter\new@command\csname
6561 \expandafter\@gobble\string#1 \endcsname
6562 }
6563 \def\x@protect#1{%
6564 \ifx\protect\@typeset@protect\else
6565 \@x@protect#1%
6566 \fi
6567 }
6568 \catcode`\&=\z@ % Trick to hide conditionals
6569 \def\@x@protect#1&\fi#2#3{&\fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

6570 \def\bbl@tempa{\csname newif\endcsname&\ifin@}
6571 \catcode`\&=4
6572 \ifx\in@\@undefined
6573 \def\in@#1#2{%
6574 \def\in@@##1#1##2##3\in@@{%

```



```

6575     \ifx\in@##2\in@false\else\in@true\fi}%
6576     \in@@#2#1\in@\in@@}
6577 \else
6578   \let\bbl@tempa\@empty
6579 \fi
6580 \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).

```

6581 \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.

```

6582 \def\ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\LaTeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

6583 \ifx\@tempcnta\@undefined
6584   \csname newcount\endcsname\@tempcnta\relax
6585 \fi
6586 \ifx\@tempcntb\@undefined
6587   \csname newcount\endcsname\@tempcntb\relax
6588 \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`).

```

6589 \ifx\bye\@undefined
6590   \advance\count10 by -2\relax
6591 \fi
6592 \ifx\@ifnextchar\@undefined
6593   \def\@ifnextchar#1#2#3{%
6594     \let\reserved@d=#1%
6595     \def\reserved@a{#2}\def\reserved@b{#3}%
6596     \futurelet\@let@token\@ifnch}
6597   \def\@ifnch{%
6598     \ifx\@let@token\@sptoken
6599       \let\reserved@c\@xifnch
6600     \else
6601       \ifx\@let@token\reserved@d
6602         \let\reserved@c\reserved@a
6603       \else
6604         \let\reserved@c\reserved@b
6605       \fi
6606     \fi
6607     \reserved@c}
6608   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6609   \def\{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6610 \fi
6611 \def\@testopt#1#2{%
6612   \@ifnextchar[#{1}{#1[#2]}}
6613 \def\@protected@testopt#1{%
6614   \ifx\protect\@typeset@protect
6615     \expandafter\@testopt

```

```

6616 \else
6617 \x@protect#1%
6618 \fi}
6619 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6620 #2\relax}\fi}
6621 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6622 \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

6623 \def\DeclareTextCommand{%
6624 \@dec@text@cmd\providecommand
6625 }
6626 \def\ProvideTextCommand{%
6627 \@dec@text@cmd\providecommand
6628 }
6629 \def\DeclareTextSymbol#1#2#3{%
6630 \@dec@text@cmd\chardef#1{#2}#3\relax
6631 }
6632 \def\@dec@text@cmd#1#2#3{%
6633 \expandafter\def\expandafter#2%
6634 \expandafter{%
6635 \csname#3-cmd\expandafter\endcsname
6636 \expandafter#2%
6637 \csname#3\string#2\endcsname
6638 }%
6639 % \let\@ifdefinable\@rc@ifdefinable
6640 \expandafter#1\csname#3\string#2\endcsname
6641 }
6642 \def\@current@cmd#1{%
6643 \ifx\protect\@typeset@protect\else
6644 \noexpand#1\expandafter\@gobble
6645 \fi
6646 }
6647 \def\@changed@cmd#1#2{%
6648 \ifx\protect\@typeset@protect
6649 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6650 \expandafter\ifx\csname ?\string#1\endcsname\relax
6651 \expandafter\def\csname ?\string#1\endcsname{%
6652 \@changed@\x@err{#1}%
6653 }%
6654 \fi
6655 \global\expandafter\let
6656 \csname\cf@encoding\string#1\expandafter\endcsname
6657 \csname ?\string#1\endcsname
6658 \fi
6659 \csname\cf@encoding\string#1%
6660 \expandafter\endcsname
6661 \else
6662 \noexpand#1%
6663 \fi
6664 }
6665 \def\@changed@\x@err#1{%
6666 \errhelp{Your command will be ignored, type <return> to proceed}%
6667 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6668 \def\DeclareTextCommandDefault#1{%
6669 \DeclareTextCommand#1?%

```

```

6670 }
6671 \def\ProvideTextCommandDefault#1{%
6672   \ProvideTextCommand#1?%
6673 }
6674 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6675 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6676 \def\DeclareTextAccent#1#2#3{%
6677   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6678 }
6679 \def\DeclareTextCompositeCommand#1#2#3#4{%
6680   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6681   \edef\reserved@b{\string##1}%
6682   \edef\reserved@c{%
6683     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6684   \ifx\reserved@b\reserved@c
6685     \expandafter\expandafter\expandafter\ifx
6686       \expandafter\@car\reserved@a\relax\relax\@nil
6687       \@text@composite
6688     \else
6689       \edef\reserved@b##1{%
6690         \def\expandafter\noexpand
6691           \csname#2\string#1\endcsname####1{%
6692             \noexpand\@text@composite
6693             \expandafter\noexpand\csname#2\string#1\endcsname
6694             ####1\noexpand\@empty\noexpand\@text@composite
6695             {##1}%
6696           }%
6697         }%
6698       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6699     \fi
6700     \expandafter\def\csname\expandafter\string\csname
6701       #2\endcsname\string#1-\string#3\endcsname{#4}
6702   \else
6703     \errhelp{Your command will be ignored, type <return> to proceed}%
6704     \errmessage{\string\DeclareTextCompositeCommand\space used on
6705       inappropriate command \protect#1}
6706   \fi
6707 }
6708 \def\@text@composite#1#2#3\@text@composite{%
6709   \expandafter\@text@composite@x
6710   \csname\string#1-\string#2\endcsname
6711 }
6712 \def\@text@composite@x#1#2{%
6713   \ifx#1\relax
6714     #2%
6715   \else
6716     #1%
6717   \fi
6718 }
6719 %
6720 \def\@strip@args#1:#2-#3\@strip@args{#2}
6721 \def\DeclareTextComposite#1#2#3#4{%
6722   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6723   \bgroup
6724     \lccode`\@=#4%
6725     \lowercase{%
6726       \egroup
6727       \reserved@a @%
6728     }%

```

```

6729 }
6730 %
6731 \def\UseTextSymbol#1#2{#2}
6732 \def\UseTextAccent#1#2#3{}
6733 \def\@use@text@encoding#1{}
6734 \def\DeclareTextSymbolDefault#1#2{%
6735   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6736 }
6737 \def\DeclareTextAccentDefault#1#2{%
6738   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6739 }
6740 \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.

```

6741 \DeclareTextAccent{"}{OT1}{127}
6742 \DeclareTextAccent{'}{OT1}{19}
6743 \DeclareTextAccent{^}{OT1}{94}
6744 \DeclareTextAccent`{OT1}{18}
6745 \DeclareTextAccent~{OT1}{126}

```

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

```

6746 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6747 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
6748 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
6749 \DeclareTextSymbol{\textquoteright}{OT1}{`\' }
6750 \DeclareTextSymbol{\i}{OT1}{16}
6751 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \LaTeX -control sequence `\scriptsize` to be available. Because plain $\text{T}_{\text{E}}\text{X}$ doesn't have such a sophisticated font mechanism as \LaTeX has, we just \let it to `\sevenrm`.

```

6752 \ifx\scriptsize\@undefined
6753   \let\scriptsize\sevenrm
6754 \fi
6755 % End of code for plain
6756 <</Emulate LaTeX>>

```

A proxy file:

```

6757 <(*plain)
6758 \input babel.def
6759 >/plain>

```

17 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.

- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).