

Babel

Version 3.48.2138
2020/09/22

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	26
1.15	Modifying a language	28
1.16	Creating a language	29
1.17	Digits and counters	33
1.18	Dates	34
1.19	Accessing language info	35
1.20	Hyphenation and line breaking	36
1.21	Selection based on BCP 47 tags	38
1.22	Selecting scripts	39
1.23	Selecting directions	40
1.24	Language attributes	44
1.25	Hooks	44
1.26	Languages supported by babel with ldf files	46
1.27	Unicode character properties in luatex	47
1.28	Tweaking some features	47
1.29	Tips, workarounds, known issues and notes	48
1.30	Current and future work	49
1.31	Tentative and experimental code	49
2	Loading languages with language.dat	50
2.1	Format	50
3	The interface between the core of babel and the language definition files	51
3.1	Guidelines for contributed languages	52
3.2	Basic macros	52
3.3	Skeleton	54
3.4	Support for active characters	55
3.5	Support for saving macro definitions	55
3.6	Support for extending macros	55
3.7	Macros common to a number of languages	56
3.8	Encoding-dependent strings	56
4	Changes	60
4.1	Changes in babel version 3.9	60
II	Source code	60

5	Identification and loading of required files	60
6	locale directory	61
7	Tools	61
7.1	Multiple languages	65
7.2	The Package File (L ^A T _E X, babel.sty)	66
7.3	base	68
7.4	Conditional loading of shorthands	70
7.5	Cross referencing macros	71
7.6	Marks	74
7.7	Preventing clashes with other packages	75
7.7.1	ifthen	75
7.7.2	varioref	76
7.7.3	hhline	76
7.7.4	hyperref	77
7.7.5	fancyhdr	77
7.8	Encoding and fonts	77
7.9	Basic bidi support	79
7.10	Local Language Configuration	84
8	The kernel of Babel (babel.def, common)	89
8.1	Tools	89
9	Multiple languages	90
9.1	Selecting the language	92
9.2	Errors	101
9.3	Hooks	103
9.4	Setting up language files	105
9.5	Shorthands	107
9.6	Language attributes	117
9.7	Support for saving macro definitions	119
9.8	Short tags	120
9.9	Hyphens	120
9.10	Multiencoding strings	122
9.11	Macros common to a number of languages	127
9.12	Making glyphs available	128
9.12.1	Quotation marks	128
9.12.2	Letters	129
9.12.3	Shorthands for quotation marks	130
9.12.4	Umlauts and tremas	131
9.13	Layout	132
9.14	Load engine specific macros	133
9.15	Creating and modifying languages	133
10	Adjusting the Babel bahavior	153
11	Loading hyphenation patterns	154
12	Font handling with fontspec	159

13	Hooks for XeTeX and LuaTeX	164
13.1	XeTeX	164
13.2	Layout	166
13.3	LuaTeX	167
13.4	Southeast Asian scripts	173
13.5	CJK line breaking	177
13.6	Automatic fonts and ids switching	177
13.7	Layout	188
13.8	Auto bidi with basic and basic-r	190
14	Data for CJK	201
15	The ‘nil’ language	201
16	Support for Plain T_EX (plain.def)	202
16.1	Not renaming hyphen.tex	202
16.2	Emulating some L ^A T _E X features	203
16.3	General tools	203
16.4	Encoding related macros	207
17	Acknowledgements	210

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. 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}
```

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.

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In 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 `other language` 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 discretionaries (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 T _E X, L ^A T _E X and ASCII strings), <code>unicode</code> (for engines like xetex and luatex) and <code>encoded</code> (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in <code>\MakeUppercase</code> and the like (this feature misuses some internal L ^A T _E X 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 ease 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 \ldots name strings).

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them currently (by means of \babelprovide), but a higher interface, based on package options, is under study. In other words, \babelprovide is mainly meant for auxiliary tasks, and as alternative when the ldf, for some reason, does not 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}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

Arabic Monolingual documents mostly work in luatex, but it must be fine tuned, and a recent version of fontspec/loaotfloat 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 `luatex` and `xetex`, but line breaking differs (rules can be modified in `luatex`; they are hard-coded in `xetex`). Lao seems to work, too, but there are no patterns for the latter in `luatex`. Khemer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and `lualatex` also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{၁၀ ၁၁ ၁၂ ၁၃ ၁၄ ၁၅} % 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}	asa	Asu
agg	Aghem	ast	Asturian ^{ul}
ak	Akan	az-Cyrl	Azerbaijani
am	Amharic ^{ul}	az-Latn	Azerbaijani
ar	Arabic ^{ul}	az	Azerbaijani ^{ul}
ar-DZ	Arabic ^{ul}	bas	Basaa
ar-MA	Arabic ^{ul}	be	Belarusian ^{ul}
ar-SY	Arabic ^{ul}	bem	Bemba
as	Assamese	bez	Bena

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

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

vai-Latn	Vai	zgh	Standard Moroccan
vai-Vaii	Vai		Tamazight
vai	Vai	zh-Hans-HK	Chinese
vi	Vietnamese ^{ul}	zh-Hans-MO	Chinese
vun	Vunjo	zh-Hans-SG	Chinese
wae	Walser	zh-Hans	Chinese
xog	Soga	zh-Hant-HK	Chinese
yav	Yangben	zh-Hant-MO	Chinese
yi	Yiddish	zh-Hant	Chinese
yo	Yoruba	zh	Chinese
yue	Cantonese	zu	Zulu

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

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

churchslavic-oldcyrillic¹³
churchslavic-glag
churchslavic-glagolitic
cognian
cornish
croatian
czech
danish
duala
dutch
dzongkha
embu
english-au
english-australia
english-ca
english-canada
english-gb
english-newzealand
english-nz
english-unitedkingdom
english-unitedstates
english-us
english
esperanto
estonian
ewe
ewondo
faroese
filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii

hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabye
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

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

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

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

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

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

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}
```

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

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

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

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

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

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

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

(In this particular case, instead of the captions group you may need to modify the `captions.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.polytonic]{babel}
```

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

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


script= $\langle script-name \rangle$

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

language= $\langle language-name \rangle$

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

alph= $\langle counter-name \rangle$

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

Alph= $\langle counter-name \rangle$

Same for `\Alph`.

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

onchar= ids | fonts

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

NOTE An alternative approach with luatex and Harfbuzz is the font option `RawFeature={multiscript=auto}`. It does not switch the babel language and therefore the line breaking rules, but in many cases it 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= $\langle base \rangle \langle shrink \rangle \langle stretch \rangle$

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

`intrapenalty=` $\langle\textit{penalty}\rangle$

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

NOTE (1) If you need shorthands, you can define them with `\usesshorthands` and `\defineshortand` 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 T_EX 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 `\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 פֶּרֶק.

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

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

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

`\localeid`

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

There are also some differences with `LaTeX`: (1) the character used is that set for the current font, while in `LaTeX` it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in `LaTeX`, but it can be changed to another value by redefining `\babenullhyphen`; (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):

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.

```

\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 (`[îú]`), the replacement could be `{1|îú|íú}`, which maps `î` to `í`, and `ú` to `û`, 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:

```

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

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

```

```

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}

```

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

`autoload.bcp47` with values `on` and `off`.

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

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

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

```

\babeladjust{ bcp47.toname = on }

```

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

1.22 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

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.

encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.¹⁷

`\ensureascii` $\langle text \rangle$

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

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

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

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.

¹⁷But still defined for backwards compatibility.

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

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

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

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

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

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

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

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

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

New 3.17 Something like:

```
\BabelFootnote{\parsfootnote}{\language}\{ }\}
```

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

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

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

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

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

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

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

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 `\usesshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also

applied to a specific language with the optional argument; language-specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three T_EX 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.

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

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

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

Then you preprocess it with devnag $\langle file \rangle$, 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).

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

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{`{}}{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 $\backslash\text{babelprovide}$, or, if the last argument is empty, removes them. The last argument is the locale name:

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

1.28 Tweaking some features

$\backslash\text{babeladjust}$ $\{\langle key-value-list \rangle\}$

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 $\backslash\text{babeladjust}\{\text{bidi.text=off}\}$ if you are using an alternative algorithm or with large sections not requiring it. With lua $\text{h}\text{b}\text{t}\text{e}\text{x}$ you may need $\text{bidi.mirroring=off}$. 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`), \TeX 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\extrarussian{\inputencoding{koi8-r}}
```

(A recent version of `inputenc` is required.)

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

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

csquotes Logical markup for quotes.

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

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

siunitx Typesetting of numbers and physical quantities.

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

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 \LaTeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study. Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-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.º 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.

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

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

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
```

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

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

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

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

```
rebuild the format. Now I will use the patterns
preloaded for english instead}}
```

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files have to be coded so that they can be read by both \LaTeX and plain \TeX . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date<lang>` but not `\captions<lang>` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in \LaTeX (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.

- Avoid adding things to `\noextras⟨lang⟩` except for `umlauth` and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras⟨lang⟩`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.²⁶
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by `babel` and it can lead to incompatibilities.

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

3.1 Guidelines for contributed languages

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

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

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

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

The following page provides a starting point for `ldf` files:

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

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

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

3.2 Basic macros

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

<code>\addlanguage</code>	The macro <code>\addlanguage</code> is a non-outer version of the macro <code>\newlanguage</code> , defined in <code>plain.tex</code> version 3.x. Here “language” is used in the \TeX sense of set of hyphenation patterns.
<code>\adddialect</code>	The macro <code>\adddialect</code> 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 <code>\language0</code> . Here “language” is used in the \TeX sense of set of hyphenation patterns.
<code>\<lang>hyphenmins</code>	The macro <code>\<lang>hyphenmins</code> is used to store the values of the <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> . Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:
<pre>\renewcommand\spanishhyphenmins{34}</pre>	
	(Assigning <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> directly in <code>\extras<lang></code> has no effect.)
<code>\providehyphenmins</code>	The macro <code>\providehyphenmins</code> should be used in the language definition files to set <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> . 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@declare@ttribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the \LaTeX command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, \LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions<lang></code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .

²⁶But not removed, for backward compatibility.

`\substitutefontfamily` (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct L^AT_EX 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`

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

`\bbl@deactivate`

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

`\declare@shorthand`

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

`\bbl@add@special`

The \TeX book states: “Plain \TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]

`\bbl@remove@special`

It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. \TeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

3.5 Support for saving macro definitions

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

`\babel@save`

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

`\babel@savevariable`

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

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

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`.

²⁷This mechanism was introduced by Bernd Raichle.

Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

3.7 Macros common to a number of languages

`\bbl@allowhyphens`

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

`\allowhyphens`

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

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

`\set@low@box`

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

`\save@sf@q`

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

`\bbl@frenchspacing`

`\bbl@nonfrenchspacing`

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

3.8 Encoding-dependent strings

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

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

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

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

`\StartBabelCommands`

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiname{M}{a}rz}
\SetString\monthivname{April}
\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
```

²⁸In future releases further categories may be added.

```

\SetString\monthviiname{Juli}
\SetString\monthviiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
\SetString\today{\number\day.\sim%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

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

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

$\backslash EndBabelCommands$ Marks the end of the series of blocks.

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

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

$\backslash SetString$ $\{ \langle macro-name \rangle \} \{ \langle string \rangle \}$

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.

$\backslash SetStringLoop$ $\{ \langle macro-name \rangle \} \{ \langle string-list \rangle \}$

A convenient way to define several ordered names at once. For example, to define $\backslash abmoniname$, $\backslash abmoniiname$, 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.

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

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap *{(to-lower-macros)}*

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

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

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

```
\SetHyphenMap{\BabelLowerMM{"100}{ "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`. 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 \TeX 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=3.48.2138>>

2 <<date=2020/09/22>>

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{%

```

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

```

37 \futurelet\bb1@trim@a\bb1@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38 \def\bb1@trim@c{%
39 \ifx\bb1@trim@a\@sptoken
40 \expandafter\bb1@trim@b
41 \else
42 \expandafter\bb1@trim@b\expandafter#1%
43 \fi}%
44 \long\def\bb1@trim@b#1##1 \@nil{\bb1@trim@i##1}}
45 \bb1@tempa{ }
46 \long\def\bb1@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bb1@trim@def#1{\bb1@trim{\def#1}}

```

`\bb1@ifunset` To check if a macro is defined, we create a new macro, which does the same as `\ifundefined`. However, in an *ε*-tex engine, it is based on `\ifcsname`, which is more efficient, and do not waste memory.

```

48 \begingroup
49 \gdef\bb1@ifunset#1{%
50 \expandafter\ifx\csname#1\endcsname\relax
51 \expandafter\@firstoftwo
52 \else
53 \expandafter\@secondoftwo
54 \fi}
55 \bb1@ifunset{ifcsname}%
56 {}%
57 {\gdef\bb1@ifunset#1{%
58 \ifcsname#1\endcsname
59 \expandafter\ifx\csname#1\endcsname\relax
60 \bb1@afterelse\expandafter\@firstoftwo
61 \else
62 \bb1@afterfi\expandafter\@secondoftwo
63 \fi
64 \else
65 \expandafter\@firstoftwo
66 \fi}}
67 \endgroup

```

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

```

68 \def\bb1@ifblank#1{%
69 \bb1@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bb1@ifblank@i#1#2\@nil#3#4#5\@nil{#4}

```

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

```

71 \def\bb1@forkv#1#2{%
72 \def\bb1@kvcmd##1##2##3{#2}%
73 \bb1@kvnext#1,\@nil,}
74 \def\bb1@kvnext#1,{%
75 \ifx\@nil#1\relax\else
76 \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
77 \expandafter\bb1@kvnext
78 \fi}
79 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
80 \bb1@trim@def\bb1@forkv@a{#1}%
81 \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@forkv@a}}{#2}{#4}}

```


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

```

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

```

\bbl@replace

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

7.1 Multiple languages

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

```

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

```

```

170 \fi
171 <</Define core switching macros>>

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

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

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

```

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

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

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

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

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

The first two options are for debugging.

```

177 <*package>
178 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
179 \ProvidesPackage{babel}[\<date>] [\<version>] The Babel package]
180 \@ifpackagewith{babel}{debug}
181   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
182   \let\bbl@debug\bbl@firstofone}
183   {\providecommand\bbl@trace[1]{}%
184   \let\bbl@debug\bbl@gobble}
185 <<Basic macros>>
186 % Temporarily repeat here the code for errors
187 \def\bbl@error#1#2{%
188   \begingroup
189     \def\{\MessageBreak}%
190     \PackageError{babel}{#1}{#2}%
191   \endgroup}
192 \def\bbl@warning#1{%
193   \begingroup
194     \def\{\MessageBreak}%
195     \PackageWarning{babel}{#1}%
196   \endgroup}
197 \def\bbl@infowarn#1{%
198   \begingroup
199     \def\{\MessageBreak}%
200     \GenericWarning
201       {(babel) \@spaces\@spaces\@spaces}%

```

```

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

```

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

```

248 \ifx\bbl@languages\undefined\else
249   \begingroup
250     \catcode\^^I=12
251     \@ifpackagewith{babel}{showlanguages}{%
252       \begingroup
253         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
254         \wlog{<*languages>}%
255         \bbl@languages
256         \wlog{</languages>}%
257       \endgroup}{%

```

```

258 \endgroup
259 \def\bbl@elt#1#2#3#4{%
260   \ifnum#2=\z@
261     \gdef\bbl@nulllanguage{#1}%
262   \def\bbl@elt##1##2##3##4{%
263     \fi}%
264   \bbl@languages
265 \fi%

```

7.3 base

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

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

```

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

```

```

307 \ifin@
308 \edef\bbl@tempc{%
309 \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
310 \else
311 \in@{=}{#1}%
312 \ifin@
313 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
314 \else
315 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
316 \bbl@csarg\edef{mod#1}{\bbl@tempb#2}%
317 \fi
318 \fi
319 \fi}
320 \let\bbl@tempc\@empty
321 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
322 \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.

```

323 \DeclareOption{KeepShorthandsActive}{}
324 \DeclareOption{activeacute}{}
325 \DeclareOption{activegrave}{}
326 \DeclareOption{debug}{}
327 \DeclareOption{noconfigs}{}
328 \DeclareOption{showlanguages}{}
329 \DeclareOption{silent}{}
330 \DeclareOption{mono}{}
331 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
332 % Don't use. Experimental. TODO.
333 \newif\ifbbl@single
334 \DeclareOption{selectors=off}{\bbl@singletrue}
335 \chardef\bbl@iniflag\z@
336 \DeclareOption{ini=*}{\chardef\bbl@iniflag\@ne} % main -> +1
337 \DeclareOption{ini.+*}{\chardef\bbl@iniflag\tw@} % add = 2
338 \DeclareOption{ini.*=*}{\chardef\bbl@iniflag\thr@@} % add + main
339 <<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.

```

340 \let\bbl@opt@shorthands\@nnil
341 \let\bbl@opt@config\@nnil
342 \let\bbl@opt@main\@nnil
343 \let\bbl@opt@headfoot\@nnil
344 \let\bbl@opt@layout\@nnil

```

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

```

345 \def\bbl@tempa#1=#2\bbl@tempa{%
346 \bbl@csarg\ifx{opt#1}\@nnil
347 \bbl@csarg\edef{opt#1}{#2}%
348 \else
349 \bbl@error
350 {Bad option `#1=#2'. Either you have misspelled the\\%
351 key or there is a previous setting of `#1'. Valid\\%

```

```

352     keys are, among others, `shorthands', `main', `bidi',\\%
353     `strings', `config', `headfoot', `safe', `math'.}%
354     {See the manual for further details.}
355 \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.

```

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

```

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

```

364 \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=...

```

365 \bbl@trace{Conditional loading of shorthands}
366 \def\bbl@sh@string#1{%
367   \ifx#1\@empty\else
368     \ifx#1t\string-
369     \else\ifx#1c\string,%
370     \else\string#1%
371   \fi\fi
372   \expandafter\bbl@sh@string
373 \fi}
374 \ifx\bbl@opt@shorthands\@nnil
375   \def\bbl@ifshorthand#1#2#3{#2}%
376 \else\ifx\bbl@opt@shorthands\@empty
377   \def\bbl@ifshorthand#1#2#3{#3}%
378 \else

```

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

```

379 \def\bbl@ifshorthand#1{%
380   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
381   \ifin@
382     \expandafter\@firstoftwo
383   \else
384     \expandafter\@secondoftwo
385   \fi}

```

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

```

386 \edef\bbl@opt@shorthands{%
387   \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.

```
388 \bbl@ifshorthand{'}%
389 {\PassOptionsToPackage{activeacute}{babel}}{}
390 \bbl@ifshorthand{`}%
391 {\PassOptionsToPackage{activegrave}{babel}}{}
392 \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.

```
393 \ifx\bbl@opt@headfoot\@nnil\else
394 \g@addto@macro\@resetactivechars{%
395 \set@typeset@protect
396 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
397 \let\protect\noexpand}
398 \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.

```
399 \ifx\bbl@opt@safe\@undefined
400 \def\bbl@opt@safe{BR}
401 \fi
402 \ifx\bbl@opt@main\@nnil\else
403 \def\bbl@language@opts{%
404 \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
405 \bbl@opt@main}
406 \fi
```

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

```
407 \bbl@trace{Defining IfBabelLayout}
408 \ifx\bbl@opt@layout\@nnil
409 \newcommand\IfBabelLayout[3]{#3}%
410 \else
411 \newcommand\IfBabelLayout[1]{%
412 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
413 \ifin@
414 \expandafter\@firstoftwo
415 \else
416 \expandafter\@secondoftwo
417 \fi}
418 \fi
```

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

```
419 \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.

```
420 <<*More package options>> ≡
421 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
422 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
423 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
424 <</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.

```
425 \bbl@trace{Cross referencing macros}
426 \ifx\bbl@opt@safe\@empty\else
427   \def\@newl@bel#1#2#3{%
428     {\@safe@activestrue
429       \bbl@ifunset{#1@#2}%
430         \relax
431         {\gdef\@multiplelabels{%
432           \@latex@warning@no@line{There were multiply-defined labels}}%
433           \@latex@warning@no@line{Label `#2' multiply defined}}%
434       \global\@namedef{#1@#2}{#3}}}
```

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

```
435 \CheckCommand*\@testdef[3]{%
436   \def\reserved@a{#3}%
437   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
438   \else
439     \@tempwattrue
440     \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.

```
441 \def\@testdef#1#2#3{% TODO. With @samestring?
442   \@safe@activestrue
443   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
444   \def\bbl@tempb{#3}%
445   \@safe@activesfalse
446   \ifx\bbl@tempa\relax
447   \else
448     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
449     \fi
450     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
451     \ifx\bbl@tempa\bbl@tempb
452     \else
453       \@tempwattrue
454       \fi}
455 \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.

```

456 \bbl@xin@{R}\bbl@opt@safe
457 \ifin@
458 \bbl@redefineroobust\ref#1{%
459 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
460 \bbl@redefineroobust\pageref#1{%
461 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
462 \else
463 \let\org@ref\ref
464 \let\org@pageref\pageref
465 \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.

```

466 \bbl@xin@{B}\bbl@opt@safe
467 \ifin@
468 \bbl@redefine\@citex[#1]#2{%
469 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
470 \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.

```

471 \AtBeginDocument{%
472 \@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.)

```

473 \def\@citex[#1][#2]#3{%
474 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
475 \org@@citex[#1][#2]{\@tempa}}%
476 }{}

```

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

```

477 \AtBeginDocument{%
478 \@ifpackageloaded{cite}{%
479 \def\@citex[#1]#2{%
480 \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
481 }{}

```

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

```

482 \bbl@redefine\nocite#1{%
483 \@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.

```
484 \bbl@redefine\bibcite{%
485   \bbl@cite@choice
486   \bibcite}
```

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

```
487 \def\bbl@bibcite#1#2{%
488   \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.

```
489 \def\bbl@cite@choice{%
490   \global\let\bibcite\bbl@bibcite
491   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
492   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
493   \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.

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

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

```
495 \bbl@redefine\@bibitem#1{%
496   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
497 \else
498   \let\org@nocite\nocite
499   \let\org@@citex\@citex
500   \let\org@bibcite\bibcite
501   \let\org@@bibitem\@bibitem
502 \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.

```
503 \bbl@trace{Marks}
504 \IfBabelLayout{sectioning}
505   {\ifx\bbl@opt@headfoot\@nnil
506     \g@addto@macro\@resetactivechars{%
507       \set@typeset@protect
508       \expandafter\select@language@x\expandafter{\bbl@main@language}%
509       \let\protect\noexpand
510       \ifcase\bbl@bidimode\else % Only with bidi. See also above
511         \edef\thepage{%
512           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
513       \fi}%
514   \fi}
515 {\ifbbl@single\else
516   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
```

```

517 \markright#1{%
518 \bbl@ifblank{#1}%
519 {\org@markright{}}}%
520 {\toks@{#1}%
521 \bbl@exp{%
522 \\\org@markright{\\protect\\foreignlanguage{\language}%
523 {\\protect\\bbl@restore@actives\the\toks@}}}}}%

```

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

```

524 \ifx\@mkboth\markboth
525 \def\bbl@tempc{\let\@mkboth\markboth}
526 \else
527 \def\bbl@tempc{
528 \fi
529 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
530 \markboth#1#2{%
531 \protected@edef\bbl@tempb##1{%
532 \protect\foreignlanguage
533 {\language}{\protect\bbl@restore@actives##1}}%
534 \bbl@ifblank{#1}%
535 {\toks@{}}%
536 {\toks@\expandafter{\bbl@tempb{#1}}}%
537 \bbl@ifblank{#2}%
538 {\@temptokena{}}%
539 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
540 \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}
541 \bbl@tempc
542 \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.

```

543 \bbl@trace{Preventing clashes with other packages}
544 \bbl@xin@{R}\bbl@opt@safe
545 \ifin@

```

```

546 \AtBeginDocument{%
547   \@ifpackageloaded{ifthen}{%
548     \bbl@redefine@long\ifthenelse#1#2#3{%
549       \let\bbl@temp@pref\pageref
550       \let\pageref\org@pageref
551       \let\bbl@temp@ref\ref
552       \let\ref\org@ref
553       \@safe@activestrue
554       \org@ifthenelse{#1}%
555       {\let\pageref\bbl@temp@pref
556        \let\ref\bbl@temp@ref
557        \@safe@activesfalse
558        #2}%
559       {\let\pageref\bbl@temp@pref
560        \let\ref\bbl@temp@ref
561        \@safe@activesfalse
562        #3}%
563     }%
564   }{}%
565 }

```

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`

```

566 \AtBeginDocument{%
567   \@ifpackageloaded{varioref}{%
568     \bbl@redefine\@@vpageref#1[#2]#3{%
569       \@safe@activestrue
570       \org@@@vpageref{#1}[#2]{#3}%
571       \@safe@activesfalse}%
572     \bbl@redefine\vrefpagenum#1#2{%
573       \@safe@activestrue
574       \org\vrefpagenum{#1}{#2}%
575       \@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.

```

576   \expandafter\def\csname Ref \endcsname#1{%
577     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
578   }{}%
579 }
580 \fi

```

7.7.3 hhline

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

```

581 \AtEndOfPackage{%
582   \AtBeginDocument{%

```

```

583 \ifpackageloaded{hhline}%
584 {\expandafter\ifx\csname normal@char\string:\endcsname\relax
585 \else
586 \makeatletter
587 \def\@currname{hhline}\input{hhline.sty}\makeatother
588 \fi}%
589 {}}}

```

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.

```

590 % \AtBeginDocument{%
591 % \ifx\pdfstringdefDisableCommands\undefined\else
592 % \pdfstringdefDisableCommands{\languageshortands{system}}%
593 % \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.

```

594 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
595 \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 L^AT_EX.

```

596 \def\substitutefontfamily#1#2#3{%
597 \lowercase{\immediate\openout15=#1#2.fd\relax}%
598 \immediate\write15{%
599 \string\ProvidesFile{#1#2.fd}%
600 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
601 \space generated font description file]^^J
602 \string\DeclareFontFamily{#1}{#2}{^^J
603 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
604 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
605 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
606 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
607 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
608 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
609 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
610 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
611 }%
612 \closeout15
613 }
614 \@onlypreamble\substitutefontfamily

```

7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of T_EX and L^AT_EX always come out in the right encoding. There is a list of non-ASCII encodings.

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

\ensureascii

```

615 \bbl@trace{Encoding and fonts}
616 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
617 \newcommand\BabelNonText{TS1,T3,TS3}
618 \let\org@TeX\TeX
619 \let\org@LaTeX\LaTeX
620 \let\ensureascii\@firstofone
621 \AtBeginDocument{%
622   \in@false
623   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
624     \ifin@
625       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
626     \fi}%
627   \ifin@ % if a text non-ascii has been loaded
628     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
629     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
630     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
631     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@@}%
632     \def\bbl@tempc#1ENC.DEF#2\@@{%
633       \ifx\empty#2\else
634         \bbl@ifunset{T@#1}%
635         {}%
636         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
637         \ifin@
638           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
639           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
640         \else
641           \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
642         \fi}%
643     \fi}%
644   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
645   \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
646   \ifin@
647     \edef\ensureascii#1{%
648       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}%
649   \fi
650 \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.

```

651 \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.

```

652 \AtBeginDocument{%

```

```

653 \ifpackageloaded{fontspec}%
654   {\xdef\latinencoding{%
655     \ifx\UTFencname\@undefined
656       EU\ifcase\bbl@engine\or2\or1\fi
657     \else
658       \UTFencname
659     \fi}}%
660 {\gdef\latinencoding{OT1}%
661   \ifx\cf@encoding\bbl@t@one
662     \xdef\latinencoding{\bbl@t@one}%
663   \else
664     \ifx\@fontenc@load@list\@undefined
665       \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
666     \else
667       \def\@elt#1{,#1,}%
668       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
669       \let\@elt\relax
670       \bbl@xin@{,T1,}\bbl@tempa
671       \ifin@
672         \xdef\latinencoding{\bbl@t@one}%
673       \fi
674     \fi
675   \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.

```

676 \DeclareRobustCommand{\latintext}{%
677   \fontencoding{\latinencoding}\selectfont
678   \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.

```

679 \ifx\@undefined\DeclareTextFontCommand
680   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
681 \else
682   \DeclareTextFontCommand{\textlatin}{\latintext}
683 \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 `LuaTeX-j` 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.

```

684 \ifodd\bbl@engine
685   \def\bbl@activate@preotf{%
686     \let\bbl@activate@preotf\relax % only once
687     \directlua{
688       Babel = Babel or {}
689       %
690       function Babel.pre_otfload_v(head)
691         if Babel.numbers and Babel.digits_mapped then
692           head = Babel.numbers(head)
693         end
694         if Babel.bidi_enabled then
695           head = Babel.bidi(head, false, dir)
696         end
697         return head
698       end
699       %
700       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
701         if Babel.numbers and Babel.digits_mapped then
702           head = Babel.numbers(head)
703         end
704         if Babel.bidi_enabled then
705           head = Babel.bidi(head, false, dir)
706         end
707         return head
708       end
709       %
710       luatexbase.add_to_callback('pre_linebreak_filter',
711         Babel.pre_otfload_v,
712         'Babel.pre_otfload_v',
713         luatexbase.priority_in_callback('pre_linebreak_filter',
714           'luaotfload.node_processor') or nil)
715       %
716       luatexbase.add_to_callback('hpack_filter',
717         Babel.pre_otfload_h,
718         'Babel.pre_otfload_h',
719         luatexbase.priority_in_callback('hpack_filter',
720           'luaotfload.node_processor') or nil)
721     }}
722 \fi

```

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

```

723 \bbl@trace{Loading basic (internal) bidi support}
724 \ifodd\bbl@engine
725   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
726     \let\bbl@beforeforeign\leavevmode
727     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
728     \RequirePackage{luatexbase}
729     \bbl@activate@preotf
730     \directlua{

```

```

731     require('babel-data-bidi.lua')
732     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
733         require('babel-bidi-basic.lua')
734     \or
735         require('babel-bidi-basic-r.lua')
736     \fi}
737 % TODO - to locale_props, not as separate attribute
738 \newattribute\bbl@attr@dir
739 % TODO. I don't like it, hackish:
740 \bbl@exp{\output{\bodydir\pagedir\the\output}}
741 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
742 \fi\fi
743 \else
744 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
745     \bbl@error
746     {The bidi method 'basic' is available only in\%
747     luatex. I'll continue with 'bidi=default', so\%
748     expect wrong results}%
749     {See the manual for further details.}%
750 \let\bbl@beforeforeign\leavevmode
751 \AtEndOfPackage{%
752     \EnableBabelHook{babel-bidi}%
753     \bbl@xebidipar}
754 \fi\fi
755 \def\bbl@loadxebidi#1{%
756     \ifx\RTLfootnotetext\undefined
757         \AtEndOfPackage{%
758             \EnableBabelHook{babel-bidi}%
759             \ifx\fontspec\undefined
760                 \bbl@loadfontspec % bidi needs fontspec
761             \fi
762             \usepackage#1{bidi}}%
763     \fi}
764 \ifnum\bbl@bidimode>200
765     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
766         \bbl@tentative{bidi=bidi}
767         \bbl@loadxebidi{}
768     \or
769         \bbl@loadxebidi{[rldocument]}
770     \or
771         \bbl@loadxebidi{}
772     \fi
773 \fi
774 \fi
775 \ifnum\bbl@bidimode=\@ne
776     \let\bbl@beforeforeign\leavevmode
777     \ifodd\bbl@engine
778         \newattribute\bbl@attr@dir
779         \bbl@exp{\output{\bodydir\pagedir\the\output}}%
780     \fi
781     \AtEndOfPackage{%
782         \EnableBabelHook{babel-bidi}%
783         \ifodd\bbl@engine\else
784             \bbl@xebidipar
785         \fi}
786 \fi

```

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

```

787 \bbl@trace{Macros to switch the text direction}
788 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
789 \def\bbl@rscripts{% TODO. Base on codes ??
790   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
791   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
792   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
793   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
794   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
795   Old South Arabian,}%
796 \def\bbl@provide@dirs#1{%
797   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
798   \ifin@
799     \global\bbl@csarg\chardef{wdir@#1}\@ne
800     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
801     \ifin@
802       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
803       \fi
804     \else
805       \global\bbl@csarg\chardef{wdir@#1}\z@
806       \fi
807     \ifodd\bbl@engine
808       \bbl@csarg\ifcase{wdir@#1}%
809         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
810         \or
811         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
812         \or
813         \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
814         \fi
815       \fi}
816 \def\bbl@switchdir{%
817   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
818   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
819   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
820 \def\bbl@setdirs#1{% TODO - math
821   \ifcase\bbl@select@type % TODO - strictly, not the right test
822     \bbl@bodydir{#1}%
823     \bbl@pardir{#1}%
824     \fi
825     \bbl@texmdir{#1}}
826 % TODO. Only if \bbl@bidimode > 0?:
827 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
828 \DisableBabelHook{babel-bidi}

```

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

```

829 \ifodd\bbl@engine % luatex=1
830   \chardef\bbl@thetexmdir\z@
831   \chardef\bbl@thepardir\z@
832   \def\bbl@getluadir#1{%
833     \directlua{
834       if tex.#1dir == 'TLT' then
835         tex.sprint('0')
836       elseif tex.#1dir == 'TRT' then
837         tex.sprint('1')
838       end}}
839 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
840   \ifcase#3\relax
841     \ifcase\bbl@getluadir{#1}\relax\else
842       #2 TLT\relax
843     \fi

```

```

844 \else
845 \ifcase\bbbl@getluadir{#1}\relax
846 #2 TRT\relax
847 \fi
848 \fi}
849 \def\bbbl@textdir#1{%
850 \bbbl@setluadir{text}\textdir{#1}%
851 \chardef\bbbl@thetextdir#1\relax
852 \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*3+#1}}
853 \def\bbbl@pardir#1{%
854 \bbbl@setluadir{par}\pardir{#1}%
855 \chardef\bbbl@thepardir#1\relax}
856 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}
857 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}
858 \def\bbbl@dirparastext{\pardir\the\textdir\relax}% %%%
859 % Sadly, we have to deal with boxes in math with basic.
860 % Activated every math with the package option bidi=:
861 \def\bbbl@mathboxdir{%
862 \ifcase\bbbl@thetextdir\relax
863 \everyhbox{\textdir TLT\relax}%
864 \else
865 \everyhbox{\textdir TRT\relax}%
866 \fi}
867 \frozen@everymath\expandafter{%
868 \expandafter\bbbl@mathboxdir\the\frozen@everymath}
869 \frozen@everydisplay\expandafter{%
870 \expandafter\bbbl@mathboxdir\the\frozen@everydisplay}
871 \else % pdftex=0, xetex=2
872 \newcount\bbbl@dirlevel
873 \chardef\bbbl@thetextdir\z@
874 \chardef\bbbl@thepardir\z@
875 \def\bbbl@textdir#1{%
876 \ifcase#1\relax
877 \chardef\bbbl@thetextdir\z@
878 \bbbl@textdir@i\beginL\endL
879 \else
880 \chardef\bbbl@thetextdir\@ne
881 \bbbl@textdir@i\beginR\endR
882 \fi}
883 \def\bbbl@textdir@i#1#2{%
884 \ifhmode
885 \ifnum\currentgrouplevel>\z@
886 \ifnum\currentgrouplevel=\bbbl@dirlevel
887 \bbbl@error{Multiple bidi settings inside a group}%
888 {I'll insert a new group, but expect wrong results.}%
889 \bgroup\aftergroup#2\aftergroup\egroup
890 \else
891 \ifcase\currentgroup\or % 0 bottom
892 \aftergroup#2% 1 simple {}
893 \or
894 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
895 \or
896 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
897 \or\or\or % vbox vtop align
898 \or
899 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
900 \or\or\or\or\or\or % output math disc insert vcent mathchoice
901 \or
902 \aftergroup#2% 14 \begingroup

```

```

903         \else
904         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
905         \fi
906         \fi
907         \bbl@dirlevel\currentgrouplevel
908         \fi
909         #1%
910         \fi}
911 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
912 \let\bbl@bodydir\@gobble
913 \let\bbl@pagedir\@gobble
914 \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).

```

915 \def\bbl@xebidipar{%
916   \let\bbl@xebidipar\relax
917   \TeXeTstate\@ne
918   \def\bbl@xeverypar{%
919     \ifcase\bbl@thepardir
920       \ifcase\bbl@thetextdir\else\beginR\fi
921     \else
922       {\setbox\z@\lastbox\beginR\box\z@}%
923     \fi}%
924   \let\bbl@severypar\everypar
925   \newtoks\everypar
926   \everypar=\bbl@severypar
927   \bbl@severypar{\bbl@xeverypar\the\everypar}}
928 \ifnum\bbl@bidimode>200
929   \let\bbl@textdir\i\@gobbletwo
930   \let\bbl@xebidipar\@empty
931   \AddBabelHook{bidi}{foreign}{%
932     \def\bbl@tempa{\def\BabelText####1}%
933     \ifcase\bbl@thetextdir
934       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
935     \else
936       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
937     \fi}
938   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
939   \fi
940 \fi

```

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

```

941 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
942 \AtBeginDocument{%
943   \ifx\pdfstringdefDisableCommands\@undefined\else
944     \ifx\pdfstringdefDisableCommands\relax\else
945       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
946     \fi
947   \fi}

```

7.10 Local Language Configuration

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

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

```

948 \bbl@trace{Local Language Configuration}
949 \ifx\loadlocalcfg\@undefined
950   \@ifpackagewith{babel}{noconfigs}%
951   {\let\loadlocalcfg@gobble}%
952   {\def\loadlocalcfg#1{%
953     \InputIfFileExists{#1.cfg}%
954     {\typeout{*****^J%
955               * Local config file #1.cfg used^^J%
956               *}}%
957     \@empty}}
958 \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?

```

959 \ifx\@unexpandable@protect\@undefined
960   \def\@unexpandable@protect{\noexpand\protect\noexpand}
961   \long\def\protected@write#1#2#3{%
962     \begingroup
963       \let\thepage\relax
964       #2%
965       \let\protect\@unexpandable@protect
966       \edef\reserved@a{\write#1{#3}}%
967       \reserved@a
968     \endgroup
969     \if@nobreak\ifvmode\nobreak\fi\fi}
970 \fi
971 %
972 % \subsection{Language options}
973 %
974 % Languages are loaded when processing the corresponding option
975 % \textit{except} if a |main| language has been set. In such a
976 % case, it is not loaded until all options has been processed.
977 % The following macro inputs the ldf file and does some additional
978 % checks (|\input| works, too, but possible errors are not caught).
979 %
980 % \begin{macrocode}
981 \bbl@trace{Language options}
982 \let\bbl@afterlang\relax
983 \let\BabelModifiers\relax
984 \let\bbl@loaded\@empty
985 \def\bbl@load@language#1{%
986   \InputIfFileExists{#1.ldf}%
987   {\edef\bbl@loaded{\CurrentOption
988     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
989     \expandafter\let\expandafter\bbl@afterlang
990     \csname\CurrentOption.ldf-h@@k\endcsname
991     \expandafter\let\expandafter\BabelModifiers
992     \csname bbl@mod@\CurrentOption\endcsname}%
993   {\bbl@error{%
994     Unknown option '\CurrentOption'. Either you misspelled it\\%
995     or the language definition file \CurrentOption.ldf was not found}}%
996     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
997     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
998     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.

```

999 \def\bbl@try@load@lang#1#2#3{%
1000 \IfFileExists{\CurrentOption.ldf}%
1001   {\bbl@load@language{\CurrentOption}}%
1002   {#1\bbl@load@language{#2}#3}}
1003 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
1004 \DeclareOption{hebrew}{%
1005   \input{rlbabel.def}%
1006   \bbl@load@language{hebrew}}
1007 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1008 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1009 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1010 \DeclareOption{polutonikogreek}{%
1011   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1012 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1013 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1014 \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.

```

1015 \ifx\bbl@opt@config\@nnil
1016   \@ifpackagewith{babel}{noconfigs}{}%
1017   {\InputIfFileExists{bblopts.cfg}%
1018     {\typeout{*****^^J%
1019               * Local config file bblopts.cfg used^^J%
1020               *}}%
1021     {}}%
1022 \else
1023   \InputIfFileExists{\bbl@opt@config.cfg}%
1024   {\typeout{*****^^J%
1025             * Local config file \bbl@opt@config.cfg used^^J%
1026             *}}%
1027   {\bbl@error{%
1028     Local config file '\bbl@opt@config.cfg' not found}{%
1029     Perhaps you misspelled it.}}%
1030 \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.

```

1031 \let\bbl@tempc\relax
1032 \bbl@foreach\bbl@language@opts{%
1033   \ifcase\bbl@iniflag
1034     \bbl@ifunset{ds@#1}%
1035     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1036     {}%
1037   \or
1038     \@gobble % case 2 same as 1
1039   \or
1040     \bbl@ifunset{ds@#1}%
1041     {\IfFileExists{#1.ldf}{}}%
1042     {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}{}}}%

```

```

1043     {}%
1044     \bbl@ifunset{ds@#1}%
1045     {\def\bbl@tempc{#1}%
1046     \DeclareOption{#1}{%
1047         \ifnum\bbl@iniflag>\@ne
1048             \bbl@ldfinit
1049             \babelprovide[import]{#1}%
1050             \bbl@afterldf}%
1051         \else
1052             \bbl@load@language{#1}%
1053         \fi}}%
1054     {}%
1055 \or
1056 \def\bbl@tempc{#1}%
1057 \bbl@ifunset{ds@#1}%
1058 {\DeclareOption{#1}{%
1059     \bbl@ldfinit
1060     \babelprovide[import]{#1}%
1061     \bbl@afterldf}}}%
1062 {}%
1063 \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.

```

1064 \let\bbl@tempb\@nnil
1065 \bbl@foreach\@classoptionslist{%
1066     \bbl@ifunset{ds@#1}%
1067     {\IfFileExists{#1.ldf}{}%
1068     {\IfFileExists{babel-#1.tex}{\DeclareOption{#1}}}}}%
1069 {}%
1070 \bbl@ifunset{ds@#1}%
1071 {\def\bbl@tempb{#1}%
1072     \DeclareOption{#1}{%
1073         \ifnum\bbl@iniflag>\@ne
1074             \bbl@ldfinit
1075             \babelprovide[import]{#1}%
1076             \bbl@afterldf}%
1077         \else
1078             \bbl@load@language{#1}%
1079         \fi}}%
1080 {}%

```

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

```

1081 \ifnum\bbl@iniflag=\z@\else
1082     \ifx\bbl@opt@main\@nnil
1083         \ifx\bbl@tempc\relax
1084             \let\bbl@opt@main\bbl@tempb
1085         \else
1086             \let\bbl@opt@main\bbl@tempc
1087         \fi
1088     \fi
1089 \fi
1090 \ifx\bbl@opt@main\@nnil\else
1091     \expandafter
1092     \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1093     \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1094 \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):

```

1095 \def\AfterBabelLanguage#1{%
1096   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1097 \DeclareOption*{}
1098 \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`.

```

1099 \bbl@trace{Option 'main'}
1100 \ifx\bbl@opt@main\@nnil
1101   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1102   \let\bbl@tempc\@empty
1103   \bbl@for\bbl@tempb\bbl@tempa{%
1104     \bbl@xin{,\bbl@tempb,}{,\bbl@loaded,}%
1105     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
1106   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1107   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1108   \ifx\bbl@tempb\bbl@tempc\else
1109     \bbl@warning{%
1110       Last declared language option is '\bbl@tempc',\%
1111       but the last processed one was '\bbl@tempb'.\%
1112       The main language cannot be set as both a global\%
1113       and a package option. Use 'main=\bbl@tempc' as\%
1114       option. Reported}%
1115   \fi
1116 \else
1117   \ifodd\bbl@iniflag % case 1,3
1118     \bbl@ldfinit
1119     \bbl@exp{\@babelprovide[import,main]{\bbl@opt@main}}
1120     \bbl@afterldf{}%
1121   \else % case 0,2
1122     \chardef\bbl@iniflag\z@ % Force ldf
1123     \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1124     \DeclareOption*{}%
1125     \ProcessOptions*
1126   \fi
1127 \fi
1128 \def\AfterBabelLanguage{%
1129   \bbl@error
1130   {Too late for \string\AfterBabelLanguage}%
1131   {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.

```

1132 \ifx\bbl@main@language\@undefined
1133   \bbl@info{%
1134     You haven't specified a language. I'll use 'nil'\%
1135     as the main language. Reported}
1136   \bbl@load@language{nil}
1137 \fi
1138 \</package>

```

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

```
1140 \ifx\ldf@quit\undefined\else
1141 \endinput\fi % Same line!
1142 <<Make sure ProvidesFile is defined>>
1143 \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 an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading babel. `\BabelModifiers` can be set too (but not sure it works).

```
1144 \ifx\AtBeginDocument\undefined % TODO. change test.
1145   <<Emulate LaTeX>>
1146   \def\language#1{english}%
1147   \let\bbl@opt@shorthands\@nnil
1148   \def\bbl@ifshorthand#1#2#3{#2}%
1149   \let\bbl@language@opts\@empty
1150   \ifx\babeloptionstrings\undefined
1151     \let\bbl@opt@strings\@nnil
1152   \else
1153     \let\bbl@opt@strings\babeloptionstrings
1154   \fi
1155   \def\BabelStringsDefault{generic}
1156   \def\bbl@tempa{normal}
1157   \ifx\babeloptionmath\bbl@tempa
1158     \def\bbl@mathnormal{\noexpand\textormath}
1159   \fi
1160   \def\AfterBabelLanguage#1#2{}
1161   \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
1162   \let\bbl@afterlang\relax
1163   \def\bbl@opt@safe{BR}
1164   \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
1165   \ifx\bbl@trace\undefined\def\bbl@trace#1{\fi
1166     \expandafter\newif\csname ifbbl@single\endcsname
1167     \chardef\bbl@bidimode\z@
1168   \fi
```

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

```
1169 \ifx\bbl@trace\undefined
```

```

1170 \let\LdfInit\endinput
1171 \def\ProvidesLanguage#1{\endinput}
1172 \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.

```

1173 <<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.

```

1174 \def\bbl@version{<<version>>}
1175 \def\bbl@date{<<date>>}
1176 \def\adddialect#1#2{%
1177   \global\chardef#1#2\relax
1178   \bbl@usehooks{adddialect}{#1}{#2}}%
1179   \begingroup
1180     \count#1\relax
1181     \def\bbl@elt##1##2##3##4{%
1182       \ifnum\count@##2\relax
1183         \bbl@info{\string#1 = using hyphenrules for ##1\\%
1184           (\string\language\the\count@)}%
1185         \def\bbl@elt####1####2####3####4}%
1186       \fi}%
1187   \bbl@cs{languages}%
1188   \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.

```

1189 \def\bbl@fixname#1{%
1190   \begingroup
1191     \def\bbl@tempe{l@}%
1192     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1193     \bbl@tempd
1194     {\lowercase\expandafter{\bbl@tempd}}%
1195     {\uppercase\expandafter{\bbl@tempd}}%
1196     \@empty
1197     {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1198     {\uppercase\expandafter{\bbl@tempd}}}%
1199     {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1200     {\lowercase\expandafter{\bbl@tempd}}}%
1201     \@empty
1202     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1203   \bbl@tempd
1204   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
1205 \def\bbl@iflanguage#1{%
1206   \@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@bcplookup` either returns the found ini or it is `\relax`.

```

1207 \def\bbl@bcpcase#1#2#3#4\@#5{%
1208   \ifx\@empty#3%
1209     \uppercase{\def#5{#1#2}}%
1210   \else
1211     \uppercase{\def#5{#1}}%
1212     \lowercase{\edef#5{#5#2#3#4}}%
1213   \fi}
1214 \def\bbl@bcplookup#1-#2-#3-#4\@{%
1215   \let\bbl@bcp\relax
1216   \lowercase{\def\bbl@tempa{#1}}%
1217   \ifx\@empty#2%
1218     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1219   \else\ifx\@empty#3%
1220     \bbl@bcpcase#2\@empty\@empty\@#\bbl@tempb
1221     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1222       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1223       {}%
1224     \ifx\bbl@bcp\relax
1225       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1226     \fi
1227   \else
1228     \bbl@bcpcase#2\@empty\@empty\@#\bbl@tempb
1229     \bbl@bcpcase#3\@empty\@empty\@#\bbl@tempc
1230     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1231       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1232       {}%
1233     \ifx\bbl@bcp\relax
1234       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1235       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1236       {}%
1237     \fi
1238     \ifx\bbl@bcp\relax
1239       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1240       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1241       {}%
1242     \fi
1243     \ifx\bbl@bcp\relax
1244       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1245     \fi
1246   \fi\fi}
1247 \let\bbl@autoload@options\@empty
1248 \let\bbl@initoload\relax
1249 \def\bbl@provide@locale{%
1250   \ifx\babelprovide\undefined
1251     \bbl@error{For a language to be defined on the fly 'base'\\%
1252               is not enough, and the whole package must be\\%
1253               loaded. Either delete the 'base' option or\\%
1254               request the languages explicitly}%
1255     {See the manual for further details.}%
1256   \fi
1257 % TODO. Option to search if loaded, with \LocaleForEach

```

```

1258 \let\bbl@auxname\language % Still necessary. TODO
1259 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1260 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1261 \ifbbl@bcpallowed
1262 \expandafter\ifx\csname date\language\endcsname\relax
1263 \expandafter
1264 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1265 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1266 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1267 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1268 \expandafter\ifx\csname date\language\endcsname\relax
1269 \let\bbl@initoload\bbl@bcp
1270 \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcpoptions]{\language}}%
1271 \let\bbl@initoload\relax
1272 \fi
1273 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1274 \fi
1275 \fi
1276 \fi
1277 \expandafter\ifx\csname date\language\endcsname\relax
1278 \IfFileExists{babel-\language.tex}%
1279 {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
1280 {}%
1281 \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.

```

1282 \def\iflanguage#1{%
1283 \bbl@iflanguage{#1}{%
1284 \ifnum\csname l@#1\endcsname=\language
1285 \expandafter\@firstoftwo
1286 \else
1287 \expandafter\@secondoftwo
1288 \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.

```

1289 \let\bbl@select@type\z@
1290 \edef\selectlanguage{%
1291 \noexpand\protect
1292 \expandafter\noexpand\csname selectlanguage \endcsname}

```

Because the command `\selectlanguage` could be used in a moving argument it expands to `\protect\selectlanguageL`. Therefore, we have to make sure that a macro `\protect` exists. If it doesn't it is `\let` to `\relax`.

```

1293 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

1294 \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.

```
1295 \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`

```
1296 \def\bbl@push@language{%
1297   \ifx\language\@undefined\else
1298     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1299   \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`.

```
1300 \def\bbl@pop@lang#1+#2\@{%
1301   \edef\language{#1}%
1302   \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).

```
1303 \let\bbl@ifrestoring\@secondoftwo
1304 \def\bbl@pop@language{%
1305   \expandafter\bbl@pop@lang\bbl@language@stack\@
1306   \let\bbl@ifrestoring\@firstoftwo
1307   \expandafter\bbl@set@language\expandafter{\language}%
1308   \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).

```
1309 \chardef\localeid\z@
1310 \def\bbl@id@last{0} % No real need for a new counter
```

```

1311 \def\bbl@id@assign{%
1312   \bbl@ifunset{bbl@id@@\language}%
1313   {\count@bbl@id@last\relax
1314    \advance\count@@ne
1315    \bbl@csarg\chardef{id@@\language}\count@
1316    \edef\bbl@id@last{\the\count@}%
1317    \ifcase\bbl@engine\or
1318      \directlua{
1319        Babel = Babel or {}
1320        Babel.locale_props = Babel.locale_props or {}
1321        Babel.locale_props[\bbl@id@last] = {}
1322        Babel.locale_props[\bbl@id@last].name = '\language'
1323      }%
1324    \fi}%
1325  }%
1326  \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of \selectlanguage.

```

1327 \expandafter\def\csname selectlanguage \endcsname#1{%
1328   \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw\fi
1329   \bbl@push@language
1330   \aftergroup\bbl@pop@language
1331   \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.

```

1332 \def\BabelContentsFiles{toc,lof,lot}
1333 \def\bbl@set@language#1{% from selectlanguage, pop@
1334   % The old buggy way. Preserved for compatibility.
1335   \edef\language{%
1336     \ifnum\escapechar=\expandafter`\string#1\@empty
1337     \else\string#1\@empty\fi}%
1338   \ifcat\relax\noexpand#1%
1339     \expandafter\ifx\csname date\language\endcsname\relax
1340       \edef\language{#1}%
1341       \let\localename\language
1342     \else
1343       \bbl@info{Using '\string\language' instead of 'language' is\%
1344         deprecated. If what you want is to use a\%
1345         macro containing the actual locale, make\%
1346         sure it does not not match any language.\%
1347         Reported}%
1348       I'll\%
1349       try to fix '\string\localename', but I cannot promise\%
1350       anything. Reported}%
1351     \ifx\scantokens\undefined
1352       \def\localename{??}%
1353     \else
1354       \scantokens\expandafter{\expandafter
1355         \def\expandafter\localename\expandafter{\language}}%
1356     \fi
1357   \fi

```

```

1358 \else
1359 \def\localename{#1}% This one has the correct catcodes
1360 \fi
1361 \select@language{\language}%
1362 % write to auxs
1363 \expandafter\ifx\csname date\language\endcsname\relax\else
1364 \if@filesw
1365 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1366 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1367 \fi
1368 \bbl@usehooks{write}}}%
1369 \fi
1370 \fi}
1371 %
1372 \newif\ifbbl@bcpallowed
1373 \bbl@bcpallowedfalse
1374 \def\select@language#1{% from set@, babel@aux
1375 % set hymap
1376 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1377 % set name
1378 \edef\language{#1}%
1379 \bbl@fixname\language
1380 % TODO. name@map must be here?
1381 \bbl@provide@locale
1382 \bbl@iflanguage\language{%
1383 \expandafter\ifx\csname date\language\endcsname\relax
1384 \bbl@error
1385 {Unknown language '\language'. Either you have\\%
1386 misspelled its name, it has not been installed,\\%
1387 or you requested it in a previous run. Fix its name,\\%
1388 install it or just rerun the file, respectively. In\\%
1389 some cases, you may need to remove the aux file}%
1390 {You may proceed, but expect wrong results}%
1391 \else
1392 % set type
1393 \let\bbl@select@type\z@
1394 \expandafter\bbl@switch\expandafter{\language}%
1395 \fi}}
1396 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1397 \select@language{#1}%
1398 \bbl@foreach\BabelContentsFiles{%
1399 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1400 \def\babel@toc#1#2{%
1401 \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.

```
1402 \newif\ifbbl@usedategroup
1403 \def\bbl@switch#1{% from select@, foreign@
1404 % make sure there is info for the language if so requested
1405 \bbl@ensureinfo{#1}%
1406 % restore
1407 \originalTeX
1408 \expandafter\def\expandafter\originalTeX\expandafter{%
1409 \csname noextras#1\endcsname
1410 \let\originalTeX\@empty
1411 \babel@beginsave}%
1412 \bbl@usehooks{afterreset}{}%
1413 \languageshorthands{none}%
1414 % set the locale id
1415 \bbl@id@assign
1416 % switch captions, date
1417 % No text is supposed to be added here, so we remove any
1418 % spurious spaces.
1419 \bbl@bsphack
1420 \ifcase\bbl@select@type
1421 \csname captions#1\endcsname\relax
1422 \csname date#1\endcsname\relax
1423 \else
1424 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1425 \ifin@
1426 \csname captions#1\endcsname\relax
1427 \fi
1428 \bbl@xin@{,date,}{,\bbl@select@opts,}%
1429 \ifin@ % if \foreign... within \<lang>date
1430 \csname date#1\endcsname\relax
1431 \fi
1432 \fi
1433 \bbl@esphack
1434 % switch extras
1435 \bbl@usehooks{beforeextras}{}%
1436 \csname extras#1\endcsname\relax
1437 \bbl@usehooks{afterextras}{}%
1438 % > babel-ensure
1439 % > babel-sh-<short>
1440 % > babel-bidi
1441 % > babel-fontspec
1442 % hyphenation - case mapping
1443 \ifcase\bbl@opt@hyphenmap\or
1444 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1445 \ifnum\bbl@hymapsel>4\else
1446 \csname\language @bbl@hyphenmap\endcsname
1447 \fi
1448 \chardef\bbl@opt@hyphenmap\z@
1449 \else
1450 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1451 \csname\language @bbl@hyphenmap\endcsname
1452 \fi
1453 \fi
1454 \global\let\bbl@hymapsel\@cclv
1455 % hyphenation - patterns
1456 \bbl@patterns{#1}%
1457 % hyphenation - mins
1458 \babel@savevariable\lefthyphenmin
```

```

1459 \babel@savevariable\rightthyphenmin
1460 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1461 \set@hyphenmins\tw@\thr@\relax
1462 \else
1463 \expandafter\expandafter\expandafter\set@hyphenmins
1464 \csname #1hyphenmins\endcsname\relax
1465 \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.

```

1466 \long\def\otherlanguage#1{%
1467 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
1468 \csname selectlanguage \endcsname{#1}%
1469 \ignorespaces}

```

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

```

1470 \long\def\endotherlanguage{%
1471 \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`.

```

1472 \expandafter\def\csname otherlanguage*\endcsname{%
1473 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1474 \def\bbl@otherlanguage@s[#1]#2{%
1475 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1476 \def\bbl@select@opts{#1}%
1477 \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”.

```

1478 \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.

```

1479 \providecommand\bbl@beforeforeign{}
1480 \edef\foreignlanguage{%
1481   \noexpand\protect
1482   \expandafter\noexpand\csname foreignlanguage \endcsname}
1483 \expandafter\def\csname foreignlanguage \endcsname{%
1484   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1485 \providecommand\bbl@foreign@x[3][{}]{%
1486   \begingroup
1487     \def\bbl@select@opts{#1}%
1488     \let\BabelText\@firstofone
1489     \bbl@beforeforeign
1490     \foreign@language{#2}%
1491     \bbl@usehooks{foreign}{}%
1492     \BabelText{#3}% Now in horizontal mode!
1493   \endgroup}
1494 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1495   \begingroup
1496     {\par}%
1497     \let\BabelText\@firstofone
1498     \foreign@language{#1}%
1499     \bbl@usehooks{foreign*}{}%
1500     \bbl@dirparastext
1501     \BabelText{#2}% Still in vertical mode!
1502     {\par}%
1503   \endgroup}

```

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

```

1504 \def\foreign@language#1{%
1505   % set name
1506   \edef\language#1%
1507   \ifbbl@usedategroup
1508     \bbl@add\bbl@select@opts{,date,}%
1509     \bbl@usedategroupfalse
1510   \fi
1511   \bbl@fixname\language
1512   % TODO. name@map here?
1513   \bbl@provide@locale
1514   \bbl@iflanguage\language{%
1515     \expandafter\ifx\csname date\language\endcsname\relax
1516       \bbl@warning % TODO - why a warning, not an error?
1517         {Unknown language `#1'. Either you have\\%
1518           misspelled its name, it has not been installed,\\%
1519           or you requested it in a previous run. Fix its name,\\%
1520           install it or just rerun the file, respectively. In\\%
1521           some cases, you may need to remove the aux file.\\%
1522           I'll proceed, but expect wrong results.\\%
1523           Reported}%
1524     \fi
1525     % set type
1526     \let\bbl@select@type\@ne
1527     \expandafter\bbl@switch\expandafter{\language}}

```

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

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

```

1528 \let\bbl@hyphlist\@empty
1529 \let\bbl@hyphenation@ \relax
1530 \let\bbl@pttnlist\@empty
1531 \let\bbl@patterns@ \relax
1532 \let\bbl@hymapsel=\@ccclv
1533 \def\bbl@patterns#1{%
1534   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1535     \csname l@#1\endcsname
1536     \edef\bbl@tempa{#1}%
1537   \else
1538     \csname l@#1:\f@encoding\endcsname
1539     \edef\bbl@tempa{#1:\f@encoding}%
1540   \fi
1541   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1542   % > luatex
1543   \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1544     \begingroup
1545       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1546       \ifin@else
1547         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1548         \hyphenation{%
1549           \bbl@hyphenation@
1550           \@ifundefined{bbl@hyphenation@#1}%
1551             \@empty
1552             {\space\csname bbl@hyphenation@#1\endcsname}}%
1553         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1554       \fi
1555     \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*.

```

1556 \def\hyphenrules#1{%
1557   \edef\bbl@tempf{#1}%
1558   \bbl@fixname\bbl@tempf
1559   \bbl@iflanguage\bbl@tempf{%
1560     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1561     \languageshortands{none}%
1562     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1563       \set@hyphenmins\tw@\thr@@\relax
1564     \else
1565       \expandafter\expandafter\expandafter\set@hyphenmins
1566       \csname\bbl@tempf hyphenmins\endcsname\relax
1567     \fi}}
1568 \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.

```
1569 \def\providehyphenmins#1#2{%
1570   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1571     \@namedef{#1hyphenmins}{#2}%
1572   \fi}
```

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

```
1573 \def\set@hyphenmins#1#2{%
1574   \lefthyphenmin#1\relax
1575   \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.

```
1576 \ifx\ProvidesFile\@undefined
1577   \def\ProvidesLanguage#1[#2 #3 #4]{%
1578     \wlog{Language: #1 #4 #3 <#2>}%
1579   }
1580 \else
1581   \def\ProvidesLanguage#1{%
1582     \begingroup
1583       \catcode`\ 10 %
1584       \@makeother\%
1585       \@ifnextchar[%]
1586         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1587     \def\@provideslanguage#1[#2]{%
1588       \wlog{Language: #1 #2}%
1589       \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1590     \endgroup}
1591 \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`.

```
1592 \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.

```
1593 \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’:

```
1594 \providecommand\setlocale{%
1595   \bbl@error
1596   {Not yet available}%
1597   {Find an armchair, sit down and wait}}
1598 \let\uselocale\setlocale
1599 \let\locale\setlocale
1600 \let\selectlocale\setlocale
1601 \let\localename\setlocale
1602 \let\textlocale\setlocale
1603 \let\textlanguage\setlocale
1604 \let\languagetext\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 $\text{\LaTeX 2}_{\epsilon}$, 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.

```

1605 \edef\bbl@nulllanguage{\string\language=0}
1606 \ifx\PackageError\@undefined % TODO. Move to Plain
1607   \def\bbl@error#1#2{%
1608     \begingroup
1609       \newlinechar=`^^J
1610       \def\{^^J(babel) }%
1611       \errhelp{#2}\errmessage{\#1}%
1612     \endgroup}
1613   \def\bbl@warning#1{%
1614     \begingroup
1615       \newlinechar=`^^J
1616       \def\{^^J(babel) }%
1617       \message{\#1}%
1618     \endgroup}
1619   \let\bbl@infowarn\bbl@warning
1620   \def\bbl@info#1{%
1621     \begingroup
1622       \newlinechar=`^^J
1623       \def\{^^J}%
1624       \wlog{#1}%
1625     \endgroup}
1626 \fi
1627 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1628 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1629   \global\@namedef{#2}{\textbf{?#1?}}%
1630   \@nameuse{#2}%
1631   \bbl@warning{%
1632     \@backslashchar#2 not set. Please, define it\\%
1633     after the language has been loaded (typically\\%
1634     in the preamble) with something like:\\%
1635     \string\renewcommand\@backslashchar#2{..}\\%
1636     Reported}}
1637 \def\bbl@tentative{\protect\bbl@tentative@i}
1638 \def\bbl@tentative@i#1{%
1639   \bbl@warning{%
1640     Some functions for '#1' are tentative.\\%
1641     They might not work as expected and their behavior\\%
1642     could change in the future.\\%
1643     Reported}}
1644 \def\@nolanerr#1{%
1645   \bbl@error
1646   {You haven't defined the language #1\space yet.\\%
1647     Perhaps you misspelled it or your installation\\%
1648     is not complete}%

```

```

1649     {Your command will be ignored, type <return> to proceed}}
1650 \def\@nopatterns#1{%
1651   \bbl@warning
1652   {No hyphenation patterns were preloaded for\\%
1653    the language `#1' into the format.\\%
1654    Please, configure your TeX system to add them and\\%
1655    rebuild the format. Now I will use the patterns\\%
1656    preloaded for \bbl@nulllanguage\space instead}}
1657 \let\bbl@usehooks\@gobbletwo
1658 \ifx\bbl@onlyswitch\@empty\endinput\fi
1659 % Here ended switch.def

Here ended switch.def.

1660 \ifx\directlua\@undefined\else
1661   \ifx\bbl@luapatterns\@undefined
1662     \input luababel.def
1663   \fi
1664 \fi
1665 <<Basic macros>>
1666 \bbl@trace{Compatibility with language.def}
1667 \ifx\bbl@languages\@undefined
1668   \ifx\directlua\@undefined
1669     \openin1 = language.def % TODO. Remove hardcoded number
1670     \ifeof1
1671       \closein1
1672       \message{I couldn't find the file language.def}
1673     \else
1674       \closein1
1675       \begingroup
1676         \def\addlanguage#1#2#3#4#5{%
1677           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1678             \global\expandafter\let\csname l@#1\endcsname
1679             \csname lang@#1\endcsname
1680           \fi}%
1681         \def\uselanguage#1{%
1682           \input language.def
1683         \endgroup
1684       \fi
1685     \fi
1686     \chardef\l@english\z@
1687 \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.

```

1688 \def\addto#1#2{%
1689   \ifx#1\@undefined
1690     \def#1{#2}%
1691   \else
1692     \ifx#1\relax
1693       \def#1{#2}%
1694     \else
1695       {\toks@\expandafter{#1#2}%
1696        \xdef#1{\the\toks@}}%
1697   \fi

```

```
1698 \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?

```
1699 \def\bbl@withactive#1#2{%
1700 \begingroup
1701 \lccode`~=`#2\relax
1702 \lowercase{\endgroup#1~}}
```

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

```
1703 \def\bbl@redefine#1{%
1704 \edef\bbl@tempa{\bbl@stripslash#1}%
1705 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1706 \expandafter\def\csname\bbl@tempa\endcsname{
1707 \@onlypreamble\bbl@redefine
```

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

```
1708 \def\bbl@redefine@long#1{%
1709 \edef\bbl@tempa{\bbl@stripslash#1}%
1710 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1711 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1712 \@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_`.

```
1713 \def\bbl@redefineroobust#1{%
1714 \edef\bbl@tempa{\bbl@stripslash#1}%
1715 \bbl@ifunset{\bbl@tempa\space}%
1716 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1717 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1718 {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1719 \@namedef{\bbl@tempa\space}%
1720 \@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.

```
1721 \bbl@trace{Hooks}
1722 \newcommand\AddBabelHook[3][{}]{%
1723 \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1724 \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1725 \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1726 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1727 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1728 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%

```



```

1729 \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1730 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1731 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1732 \def\bbl@usehooks#1#2{%
1733   \def\bbl@elt##1{%
1734     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}}%
1735   \bbl@cs{ev@#1@}%
1736   \ifx\language\undefined\else % Test required for Plain (?)
1737     \def\bbl@elt##1{%
1738       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}}%
1739     \bbl@cl{ev@#1}%
1740   \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).

```

1741 \def\bbl@evargs{,% <- don't delete this comma
1742   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1743   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1744   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1745   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1746   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.

```

1747 \bbl@trace{Defining babelensure}
1748 \newcommand\babelensure[2][{}]{% TODO - revise test files
1749   \AddBabelHook{babel-ensure}{afterextras}{%
1750     \ifcase\bbl@select@type
1751       \bbl@cl{e}%
1752     \fi}%
1753   \begingroup
1754     \let\bbl@ens@include\@empty
1755     \let\bbl@ens@exclude\@empty
1756     \def\bbl@ens@fontenc{\relax}%
1757     \def\bbl@tempb##1{%
1758       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1759     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1760     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1761     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1762     \def\bbl@tempc{\bbl@ensure}%
1763     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1764       \expandafter{\bbl@ens@include}}%
1765     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1766       \expandafter{\bbl@ens@exclude}}%
1767     \toks@\expandafter{\bbl@tempc}%
1768     \bbl@exp{%
1769   \endgroup
1770   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}

```

```

1771 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1772 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1773 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1774 \edef##1{\noexpand\bbl@nocaption
1775 {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1776 \fi
1777 \ifx##1\@empty\else
1778 \in@{##1}{#2}%
1779 \ifin\else
1780 \bbl@ifunset{\bbl@ensure@\language}%
1781 {\bbl@exp{%
1782 \\\DeclareRobustCommand\<bbl@ensure@[1]{%
1783 \\\foreignlanguage{\language}%
1784 {\ifx\relax#3\else
1785 \\\fontencoding{#3}\selectfont
1786 \fi
1787 #####1}}}%
1788 }%
1789 \toks@\expandafter{##1}%
1790 \edef##1{%
1791 \bbl@csarg\noexpand{ensure@\language}%
1792 {\the\toks@}}%
1793 \fi
1794 \expandafter\bbl@tempb
1795 \fi}%
1796 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1797 \def\bbl@tempa##1{% elt for include list
1798 \ifx##1\@empty\else
1799 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1800 \ifin\else
1801 \bbl@tempb##1\@empty
1802 \fi
1803 \expandafter\bbl@tempa
1804 \fi}%
1805 \bbl@tempa#1\@empty}
1806 \def\bbl@captionslist{%
1807 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1808 \contentsname\listfigurename\listtablename\indexname\figurename
1809 \tablename\partname\encname\ccname\headtoname\pagename\seename
1810 \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.

```
1811 \bbl@trace{Macros for setting language files up}
1812 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1813   \let\bbl@screset\@empty
1814   \let\BabelStrings\bbl@opt@string
1815   \let\BabelOptions\@empty
1816   \let\BabelLanguages\relax
1817   \ifx\originalTeX\@undefined
1818     \let\originalTeX\@empty
1819   \else
1820     \originalTeX
1821   \fi}
1822 \def\LdfInit#1#2{%
1823   \chardef\atcatcode=\catcode`\@
1824   \catcode`\@=11\relax
1825   \chardef\eqcatcode=\catcode`\=
1826   \catcode`\==12\relax
1827   \expandafter\if\expandafter\@backslashchar
1828     \expandafter\@car\string#2\@nil
1829     \ifx#2\@undefined\else
1830       \ldf@quit{#1}%
1831     \fi
1832   \else
1833     \expandafter\ifx\csname#2\endcsname\relax\else
1834       \ldf@quit{#1}%
1835     \fi
1836   \fi
1837   \bbl@ldfinit}
```

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

```
1838 \def\ldf@quit#1{%
1839   \expandafter\main@language\expandafter{#1}%
1840   \catcode`\@=\atcatcode \let\atcatcode\relax
1841   \catcode`\==\eqcatcode \let\eqcatcode\relax
1842   \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.

```
1843 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1844   \bbl@afterlang
1845   \let\bbl@afterlang\relax
1846   \let\BabelModifiers\relax
1847   \let\bbl@screset\relax}%
1848 \def\ldf@finish#1{%
1849   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1850     \loadlocalcfg{#1}%
1851   \fi
1852   \bbl@afterldf{#1}%
1853   \expandafter\main@language\expandafter{#1}%
1854   \catcode`\@=\atcatcode \let\atcatcode\relax
1855   \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 .

```
1856 \@onlypreamble\LdfInit
1857 \@onlypreamble\ldf@quit
1858 \@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.

```
1859 \def\main@language#1{%
1860   \def\bbl@main@language{#1}%
1861   \let\language\name\bbl@main@language % TODO. Set localename
1862   \bbl@id@assign
1863   \bbl@patterns{\language}%}
```

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

```
1864 \def\bbl@beforestart{%
1865   \bbl@usehooks{beforestart}{}%
1866   \global\let\bbl@beforestart\relax}
1867 \AtBeginDocument{%
1868   \@nameuse{bbl@beforestart}%
1869   \if@filesw
1870     \providecommand\babel@aux[2]{}%
1871     \immediate\write\@mainaux{%
1872       \string\providecommand\string\babel@aux[2]{}%
1873       \immediate\write\@mainaux{\string\nameuse{bbl@beforestart}}}%
1874   \fi
1875   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1876   \ifbbl@single % must go after the line above.
1877     \renewcommand\selectlanguage[1]{}%
1878     \renewcommand\foreignlanguage[2]{#2}%
1879     \global\let\babel@aux\@gobbles % Also as flag
1880   \fi
1881   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1882 \def\select@language@x#1{%
1883   \ifcase\bbl@select@type
1884     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1885   \else
1886     \select@language{#1}%
1887   \fi}
```

9.5 Shorthands

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

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

```
1888 \bbl@trace{Shorthands}
```

```

1889 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1890 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1891 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1892 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1893 \begingroup
1894 \catcode`#1\active
1895 \nfss@catcodes
1896 \ifnum\catcode`#1=\active
1897 \endgroup
1898 \bbl@add\nfss@catcodes{\@makeother#1}%
1899 \else
1900 \endgroup
1901 \fi
1902 \fi}

```

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

```

1903 \def\bbl@remove@special#1{%
1904 \begingroup
1905 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1906 \else\noexpand##1\noexpand##2\fi}%
1907 \def\do{\x\do}%
1908 \def\@makeother{\x\@makeother}%
1909 \edef\x{\endgroup
1910 \def\noexpand\dospecials{\dospecials}%
1911 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1912 \def\noexpand\@sanitize{\@sanitize}%
1913 \fi}%
1914 \x}

```

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

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

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

```

1915 \def\bbl@active@def#1#2#3#4{%
1916 \@namedef{#3#1}{%
1917 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1918 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1919 \else
1920 \bbl@afterfi\csname#2@sh@#1\endcsname
1921 \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.

```

1922 \long\@namedef{#3@arg#1}##1{%
1923   \expandafter\ifx\csname#2@sh@#1@\string##1@endcsname\relax
1924     \bbl@afterelse\csname#4#1@endcsname##1%
1925   \else
1926     \bbl@afterfi\csname#2@sh@#1@\string##1@endcsname
1927   \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.

```

1928 \def\initiate@active@char#1{%
1929   \bbl@ifunset{active@char\string#1}%
1930   {\bbl@withactive
1931     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1932   {}}

```

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).

```

1933 \def\@initiate@active@char#1#2#3{%
1934   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1935   \ifx#1\@undefined
1936     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1937   \else
1938     \bbl@csarg\let{oridef@#2}#1%
1939     \bbl@csarg\edef{oridef@#2}{%
1940       \let\noexpand#1%
1941       \expandafter\noexpand\csname bbl@oridef@#2@endcsname}%
1942   \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*").

```

1943 \ifx#1#3\relax
1944   \expandafter\let\csname normal@char#2@endcsname#3%
1945 \else
1946   \bbl@info{Making #2 an active character}%
1947   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1948   \@namedef{normal@char#2}{%
1949     \textormath{#3}{\csname bbl@oridef@#2@endcsname}}%
1950   \else
1951     \@namedef{normal@char#2}{#3}%
1952   \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).

```

1953   \bbl@restoreactive{#2}%

```

```

1954 \AtBeginDocument{%
1955   \catcode`#2\active
1956   \if@filesw
1957     \immediate\write\@mainaux{\catcode`\string#2\active}%
1958   \fi}%
1959 \expandafter\bb1@add@special\csname#2\endcsname
1960 \catcode`#2\active
1961 \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⟩`).

```

1962 \let\bb1@tempa\@firstoftwo
1963 \if\string^#2%
1964   \def\bb1@tempa{\noexpand\textormath}%
1965 \else
1966   \ifx\bb1@mathnormal\@undefined\else
1967     \let\bb1@tempa\bb1@mathnormal
1968   \fi
1969 \fi
1970 \expandafter\edef\csname active@char#2\endcsname{%
1971   \bb1@tempa
1972     {\noexpand\if@safe@actives
1973       \noexpand\expandafter
1974         \expandafter\noexpand\csname normal@char#2\endcsname
1975       \noexpand\else
1976         \noexpand\expandafter
1977         \expandafter\noexpand\csname bbl@doactive#2\endcsname
1978       \noexpand\fi}%
1979   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1980 \bb1@csarg\edef{doactive#2}{%
1981   \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!).

```

1982 \bb1@csarg\edef{active@#2}{%
1983   \noexpand\active@prefix\noexpand#1%
1984   \expandafter\noexpand\csname active@char#2\endcsname}%
1985 \bb1@csarg\edef{normal@#2}{%
1986   \noexpand\active@prefix\noexpand#1%
1987   \expandafter\noexpand\csname normal@char#2\endcsname}%
1988 \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.

```

1989 \bb1@active@def#2\user@group{user@active}{language@active}%
1990 \bb1@active@def#2\language@group{language@active}{system@active}%
1991 \bb1@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.

```
1992 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1993   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1994 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1995   {\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.

```
1996 \if\string'#2%
1997   \let\prim@s\bbl@prim@s
1998   \let\active@math@prime#1%
1999 \fi
2000 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
2001 <<{*More package options}>> ≡
2002 \DeclareOption{math=active}{}
2003 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2004 <</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.

```
2005 \@ifpackagewith{babel}{KeepShorthandsActive}%
2006   {\let\bbl@restoreactive\@gobble}%
2007   {\def\bbl@restoreactive#1{%
2008     \bbl@exp{%
2009       \\\AfterBabelLanguage\\CurrentOption
2010       {\catcode`#1=\the\catcode`#1\relax}%
2011       \\\AtEndOfPackage
2012       {\catcode`#1=\the\catcode`#1\relax}}}%
2013   \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.

```
2014 \def\bbl@sh@select#1#2{%
2015   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2016     \bbl@afterelse\bbl@scndcs
2017   \else
2018     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2019   \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.


```

2020 \begingroup
2021 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2022 {\gdef\active@prefix#1{%
2023   \ifx\protect\@typeset@protect
2024   \else
2025     \ifx\protect\@unexpandable@protect
2026       \noexpand#1%
2027     \else
2028       \protect#1%
2029     \fi
2030   \expandafter\@gobble
2031 \fi}}
2032 {\gdef\active@prefix#1{%
2033   \ifincsname
2034     \string#1%
2035   \expandafter\@gobble
2036 \else
2037   \ifx\protect\@typeset@protect
2038   \else
2039     \ifx\protect\@unexpandable@protect
2040       \noexpand#1%
2041     \else
2042       \protect#1%
2043     \fi
2044   \expandafter\expandafter\expandafter\@gobble
2045 \fi
2046 \fi}}
2047 \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⟩`.

```

2048 \newif\if@safe@actives
2049 \@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.

```

2050 \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`.

```

2051 \def\bbl@activate#1{%
2052   \bbl@withactive{\expandafter\let\expandafter}#1%
2053   \csname bbl@active@\string#1\endcsname}
2054 \def\bbl@deactivate#1{%
2055   \bbl@withactive{\expandafter\let\expandafter}#1%
2056   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs 2057 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2058 \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.

```

2059 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2060 \def\@decl@short#1#2#3\@nil#4{%
2061   \def\bbl@tempa{#3}%
2062   \ifx\bbl@tempa\@empty
2063     \expandafter\let\csname #1@sh@\string#2@sel@endcsname\bbl@scndcs
2064     \bbl@ifunset{#1@sh@\string#2@}{}%
2065     {\def\bbl@tempa{#4}%
2066      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2067      \else
2068        \bbl@info
2069          {Redefining #1 shorthand \string#2\\%
2070           in language \CurrentOption}%
2071        \fi}%
2072     \@namedef{#1@sh@\string#2@}{#4}%
2073   \else
2074     \expandafter\let\csname #1@sh@\string#2@sel@endcsname\bbl@firstcs
2075     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2076     {\def\bbl@tempa{#4}%
2077      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2078      \else
2079        \bbl@info
2080          {Redefining #1 shorthand \string#2\string#3\\%
2081           in language \CurrentOption}%
2082        \fi}%
2083     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2084   \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.

```

2085 \def\textormath{%
2086   \ifmmode
2087     \expandafter\@secondoftwo
2088   \else
2089     \expandafter\@firstoftwo
2090   \fi}

```

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

```

2091 \def\user@group{user}
2092 \def\language@group{english} % TODO. I don't like defaults
2093 \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.

```

2094 \def\useshorthands{%
2095   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2096 \def\bbl@usesh@s#1{%
2097   \bbl@usesh@x
2098   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2099   {#1}}

```

```

2100 \def\bbl@usesh@x#1#2{%
2101   \bbl@ifshorthand{#2}%
2102   {\def\user@group{user}%
2103     \initiate@active@char{#2}%
2104     #1%
2105     \bbl@activate{#2}}%
2106   {\bbl@error
2107     {Cannot declare a shorthand turned off (\string#2)}
2108     {Sorry, but you cannot use shorthands which have been\\%
2109       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.

```

2110 \def\user@language@group{user@\language@group}
2111 \def\bbl@set@user@generic#1#2{%
2112   \bbl@ifunset{user@generic@active#1}%
2113   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2114     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2115     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2116       \expandafter\noexpand\csname normal@char#1\endcsname}%
2117     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
2118       \expandafter\noexpand\csname user@active#1\endcsname}}%
2119   \@empty}
2120 \newcommand\defineshorthand[3][user]{%
2121   \edef\bbl@tempa{\zap@space#1 \@empty}%
2122   \bbl@for\bbl@tempb\bbl@tempa{%
2123     \if*\expandafter\@car\bbl@tempb\@nil
2124       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2125       \@expandtwoargs
2126       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2127     \fi
2128     \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].

```

2129 \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`.

```

2130 \def\aliasshorthand#1#2{%
2131   \bbl@ifshorthand{#2}%
2132   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2133     \ifx\document\@notprerr
2134       \@notshorthand{#2}%
2135     \else
2136       \initiate@active@char{#2}%
2137       \expandafter\let\csname active@char\string#2\endcsname
2138         \csname active@char\string#1\endcsname
2139       \expandafter\let\csname normal@char\string#2\endcsname
2140         \csname normal@char\string#1\endcsname
2141       \bbl@activate{#2}%

```

```

2142     \fi
2143   \fi}%
2144   {\bbl@error
2145     {Cannot declare a shorthand turned off (\string#2)}
2146     {Sorry, but you cannot use shorthands which have been\\%
2147       turned off in the package options}}}

```

\@notshorthand

```

2148 \def\@notshorthand#1{%
2149   \bbl@error{%
2150     The character '\string #1' should be made a shorthand character;\\%
2151     add the command \string\usesshorthands\string{#1\string} to
2152     the preamble.\\%
2153     I will ignore your instruction}%
2154   {You may proceed, but expect unexpected results}}

```

\shorthandon The first level definition of these macros just passes the argument on to \bbl@switch@sh, adding \@nil at the end to denote the end of the list of characters.

```

2155 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2156 \DeclareRobustCommand*\shorthandoff{%
2157   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2158 \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.

```

2159 \def\bbl@switch@sh#1#2{%
2160   \ifx#2\@nnil\else
2161     \bbl@ifunset{\bbl@active@\string#2}%
2162     {\bbl@error
2163       {I cannot switch '\string#2' on or off--not a shorthand}%
2164       {This character is not a shorthand. Maybe you made\\%
2165         a typing mistake? I will ignore your instruction}}}%
2166     {\ifcase#1%
2167       \catcode`#212\relax
2168       \or
2169       \catcode`#2\active
2170       \or
2171       \csname bbl@oricat@\string#2\endcsname
2172       \csname bbl@oridef@\string#2\endcsname
2173       \fi}%
2174     \bbl@afterfi\bbl@switch@sh#1%
2175   \fi}

```

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

```

2176 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2177 \def\bbl@putsh#1{%
2178   \bbl@ifunset{\bbl@active@\string#1}%
2179   {\bbl@putsh#1\@empty\@nnil}%
2180   {\csname bbl@active@\string#1\endcsname}}

```

```

2181 \def\bbl@putsh@i#1#2\@nnil{%
2182   \csname\language@group @sh\string#1@%
2183     \ifx\@empty#2\else\string#2@fi\endcsname}
2184 \ifx\bbl@opt@shorthands\@nnil\else
2185   \let\bbl@s@initiate@active@char\initiate@active@char
2186   \def\initiate@active@char#1{%
2187     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2188   \let\bbl@s@switch@sh\bbl@switch@sh
2189   \def\bbl@switch@sh#1#2{%
2190     \ifx#2\@nnil\else
2191       \bbl@afterfi
2192       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2193     \fi}
2194   \let\bbl@s@activate\bbl@activate
2195   \def\bbl@activate#1{%
2196     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2197   \let\bbl@s@deactivate\bbl@deactivate
2198   \def\bbl@deactivate#1{%
2199     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2200 \fi

```

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

```

2201 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

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

\bbl@pr@m@s

```

2202 \def\bbl@prim@s{%
2203   \prime\futurelet\@let@token\bbl@pr@m@s}
2204 \def\bbl@if@primes#1#2{%
2205   \ifx#1\@let@token
2206     \expandafter\@firstoftwo
2207   \else\ifx#2\@let@token
2208     \bbl@afterelse\expandafter\@firstoftwo
2209   \else
2210     \bbl@afterfi\expandafter\@secondoftwo
2211   \fi\fi}
2212 \begingroup
2213   \catcode`\^=7 \catcode`\*=\active \lccode`\*='^
2214   \catcode`\'=12 \catcode`\"=\active \lccode`\"=' '
2215   \lowercase{%
2216     \gdef\bbl@pr@m@s{%
2217       \bbl@if@primes" '%
2218         \pr@@@s
2219         {\bbl@if@primes*^ \pr@@@t\egroup}}}
2220 \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).

```

2221 \initiate@active@char{~}
2222 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2223 \bbl@activate{~}

```

\OT1dqpos The position of the double quote character is different for the OT1 and T1 encodings. It will
 \T1dqpos later be selected using the \f@encoding macro. Therefore we define two macros here to
 store the position of the character in these encodings.

```
2224 \expandafter\def\csname OT1dqpos\endcsname{127}
2225 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
2226 \ifx\f@encoding\undefined
2227   \def\f@encoding{OT1}
2228 \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.

```
2229 \bbl@trace{Language attributes}
2230 \newcommand\languageattribute[2]{%
2231   \def\bbl@tempc{#1}%
2232   \bbl@fixname\bbl@tempc
2233   \bbl@iflanguage\bbl@tempc{%
2234     \bbl@vforeach{#2}{%
```

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

```
2235     \ifx\bbl@known@attribs\undefined
2236       \in@false
2237     \else
2238       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2239     \fi
2240     \ifin@
2241       \bbl@warning{%
2242         You have more than once selected the attribute '##1'\%
2243         for language #1. Reported}%
2244     \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.

```
2245     \bbl@exp{%
2246       \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
2247     \edef\bbl@tempa{\bbl@tempc-##1}%
2248     \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2249     {\csname\bbl@tempc @attr##1\endcsname}%
2250     {\@attrerr{\bbl@tempc}{##1}}%
2251   \fi}}
2252 \onlypreamble\languageattribute
```

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

```
2253 \newcommand*{\@attrerr}[2]{%
2254   \bbl@error
2255   {The attribute #2 is unknown for language #1.}%
2256   {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}`.

```

2257 \def\bbl@declare@ttribute#1#2#3{%
2258   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2259   \ifin@
2260     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2261   \fi
2262   \bbl@add@list\bbl@attributes{#1-#2}%
2263   \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

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

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

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```

2264 \def\bbl@ifattributeset#1#2#3#4{%
2265   \ifx\bbl@known@attribs\@undefined
2266     \in@false
2267   \else
2268     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2269   \fi
2270   \ifin@
2271     \bbl@afterelse#3%
2272   \else
2273     \bbl@afterfi#4%
2274   \fi
2275 }

```

`\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`.

```

2276 \def\bbl@ifknown@ttrib#1#2{%
2277   \let\bbl@tempa\@secondoftwo
2278   \bbl@loopx\bbl@tempb{#2}{%
2279     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2280     \ifin@
2281       \let\bbl@tempa\@firstoftwo
2282     \else
2283       \fi}%
2284   \bbl@tempa
2285 }

```

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

```

2286 \def\bbl@clear@ttribs{%
2287   \ifx\bbl@attributes\@undefined\else
2288     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2289       \expandafter\bbl@clear@ttrib\bbl@tempa.
2290     }%
2291     \let\bbl@attributes\@undefined
2292   \fi}
2293 \def\bbl@clear@ttrib#1-#2.{%
2294   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2295 \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` 2296 `\bbl@trace{Macros for saving definitions}`
 2297 `\def\babel@beginsave{\babel@savecnt\z@}`

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

```

2298 \newcount\babel@savecnt
2299 \babel@beginsave

```

`\babel@save` The macro `\babel@save<csname>` saves the current meaning of the control sequence
`\babel@savevariable` `<csname>` to `\originalTeX`³¹. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive.

```

2300 \def\babel@save#1{%
2301   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2302   \toks@\expandafter{\originalTeX\let#1=%}
2303   \bbl@exp{%
2304     \def\\originalTeX{\the\toks@<\babel@\number\babel@savecnt>\relax}}%
2305   \advance\babel@savecnt\@ne}
2306 \def\babel@savevariable#1{%
2307   \toks@\expandafter{\originalTeX #1=%}
2308   \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.

```

2309 \def\bbl@frenchspacing{%
2310   \ifnum\the\sfcode`.=\@m
2311     \let\bbl@nonfrenchspacing\relax
2312   \else
2313     \frenchspacing
2314     \let\bbl@nonfrenchspacing\nonfrenchspacing
2315   \fi}
2316 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

³¹`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

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.

```

2317 \bbl@trace{Short tags}
2318 \def\babeltags#1{%
2319   \edef\bbl@tempa{\zap@space#1 \@empty}%
2320   \def\bbl@tempb##1=##2\@{#1}%
2321   \edef\bbl@tempc{%
2322     \noexpand\newcommand
2323     \expandafter\noexpand\csname ##1\endcsname{%
2324       \noexpand\protect
2325       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2326     \noexpand\newcommand
2327     \expandafter\noexpand\csname text##1\endcsname{%
2328       \noexpand\foreignlanguage{##2}}
2329   \bbl@tempc}%
2330 \bbl@for\bbl@tempa\bbl@tempa{%
2331   \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.

```

2332 \bbl@trace{Hyphens}
2333 \@onlypreamble\babelhyphenation
2334 \AtEndOfPackage{%
2335   \newcommand\babelhyphenation[2][\@empty]{%
2336     \ifx\bbl@hyphenation@ \relax
2337       \let\bbl@hyphenation@ \@empty
2338     \fi
2339     \ifx\bbl@hyphlist \@empty \else
2340       \bbl@warning{%
2341         You must not intermingle \string\selectlanguage\space and\%
2342         \string\babelhyphenation\space or some exceptions will not\%
2343         be taken into account. Reported}%
2344     \fi
2345     \ifx\@empty#1%
2346       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2347     \else
2348       \bbl@vforeach{#1}{%
2349         \def\bbl@tempa{##1}%
2350         \bbl@fixname\bbl@tempa
2351         \bbl@iflanguage\bbl@tempa{%
2352           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2353             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2354             \@empty
2355             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2356             #2}}}%
2357       \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`³².

³² \TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

2358 \def\bb1@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2359 \def\bb1@t@one{T1}
2360 \def\allowhyphens{\ifx\cf@encoding\bb1@t@one\else\bb1@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.

```

2361 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2362 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
2363 \def\bb1@hyphen{%
2364   \@ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i\@empty}}
2365 \def\bb1@hyphen@i#1#2{%
2366   \bb1@ifunset{\bb1@hy@#1#2\@empty}%
2367   {\csname \bb1@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2368   {\csname \bb1@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.

```

2369 \def\bb1@usehyphen#1{%
2370   \leavevmode
2371   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2372   \nobreak\hskip\z@skip}
2373 \def\bb1@usehyphen#1{%
2374   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2375 \def\bb1@hyphenchar{%
2376   \ifnum\hyphenchar\font=\m@ne
2377     \babelnullhyphen
2378   \else
2379     \char\hyphenchar\font
2380   \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 \bb1@hy@nobreak is redundant.

```

2381 \def\bb1@hy@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
2382 \def\bb1@hy@@soft{\bb1@usehyphen{\discretionary{\bb1@hyphenchar}{}}{}}
2383 \def\bb1@hy@hard{\bb1@usehyphen\bb1@hyphenchar}
2384 \def\bb1@hy@@hard{\bb1@usehyphen\bb1@hyphenchar}
2385 \def\bb1@hy@nobreak{\bb1@usehyphen{\mbox{\bb1@hyphenchar}}}
2386 \def\bb1@hy@@nobreak{\mbox{\bb1@hyphenchar}}
2387 \def\bb1@hy@repeat{%
2388   \bb1@usehyphen{%
2389     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
2390 \def\bb1@hy@@repeat{%
2391   \bb1@usehyphen{%
2392     \discretionary{\bb1@hyphenchar}{\bb1@hyphenchar}{\bb1@hyphenchar}}}
2393 \def\bb1@hy@empty{\hskip\z@skip}
2394 \def\bb1@hy@@empty{\discretionary{}{}{}}

```

\bb1@disc For some languages the macro \bb1@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.

```

2395 \def\bb1@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bb1@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.

```
2396 \bbl@trace{Multiencoding strings}
2397 \def\bbl@tglobal#1{\global\let#1#1}
2398 \def\bbl@recatcode#1{% TODO. Used only once?
2399   \@tempcnta="7F
2400   \def\bbl@tempa{%
2401     \ifnum\@tempcnta>"FF\else
2402       \catcode\@tempcnta=#1\relax
2403       \advance\@tempcnta\@ne
2404       \expandafter\bbl@tempa
2405     \fi}%
2406   \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).

```
2407 \@ifpackagewith{babel}{nocase}%
2408   {\let\bbl@patchuclc\relax}%
2409   {\def\bbl@patchuclc{%
2410     \global\let\bbl@patchuclc\relax
2411     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2412     \gdef\bbl@uclc##1{%
2413       \let\bbl@encoded\bbl@encoded@uclc
2414       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2415       {##1}%
2416       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2417         \csname\language @bbl@uclc\endcsname}%
2418         {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2419     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2420     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}
```

```
2421 <<(*More package options)>> ≡
2422 \DeclareOption{nocase}{}
2423 <</More package options>>
```

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

```
2424 <<(*More package options)>> ≡
2425 \let\bbl@opt@strings\@nnil % accept strings=value
2426 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2427 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2428 \def\BabelStringsDefault{generic}
2429 <</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.

```

2430 \@onlypreamble\StartBabelCommands
2431 \def\StartBabelCommands{%
2432   \begingroup
2433   \bbl@recatcode{11}%
2434   <⟨Macros local to BabelCommands⟩
2435   \def\bbl@provstring##1##2{%
2436     \providecommand##1{##2}%
2437     \bbl@toglobal##1}%
2438   \global\let\bbl@scafter\@empty
2439   \let\StartBabelCommands\bbl@startcmds
2440   \ifx\BabelLanguages\relax
2441     \let\BabelLanguages\CurrentOption
2442   \fi
2443   \begingroup
2444   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2445   \StartBabelCommands}
2446 \def\bbl@startcmds{%
2447   \ifx\bbl@screset\@nnil\else
2448     \bbl@usehooks{stopcommands}{}%
2449   \fi
2450   \endgroup
2451   \begingroup
2452   \@ifstar
2453     {\ifx\bbl@opt@strings\@nnil
2454       \let\bbl@opt@strings\BabelStringsDefault
2455     \fi
2456     \bbl@startcmds@i}%
2457   \bbl@startcmds@i}
2458 \def\bbl@startcmds@i#1#2{%
2459   \edef\bbl@L{\zap@space#1 \@empty}%
2460   \edef\bbl@G{\zap@space#2 \@empty}%
2461   \bbl@startcmds@ii}
2462 \let\bbl@startcmds\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.

```

2463 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2464   \let\SetString\@gobbletwo
2465   \let\bbl@stringdef\@gobbletwo
2466   \let\AfterBabelCommands\@gobble
2467   \ifx\@empty#1%
2468     \def\bbl@sc@label{generic}%
2469     \def\bbl@encstring##1##2{%
2470       \ProvideTextCommandDefault##1{##2}%
2471       \bbl@toglobal##1%
2472       \expandafter\bbl@toglobal\csname\string?string##1\endcsname}%
2473     \let\bbl@sctest\in@true

```

```

2474 \else
2475 \let\bbl@sc@charset\space % <- zapped below
2476 \let\bbl@sc@fontenc\space % <- " "
2477 \def\bbl@tempa##1=##2\nil{%
2478 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
2479 \bbl@vforeach{label=#1}{\bbl@tempa##1\nil}%
2480 \def\bbl@tempa##1 ##2{% space -> comma
2481 ##1%
2482 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2483 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2484 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2485 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2486 \def\bbl@encstring##1##2{%
2487 \bbl@foreach\bbl@sc@fontenc{%
2488 \bbl@ifunset{T@###1}%
2489 {}%
2490 {\ProvideTextCommand##1{###1}{##2}%
2491 \bbl@tglobal##1%
2492 \expandafter
2493 \bbl@tglobal\csname###1\string##1\endcsname}}}%
2494 \def\bbl@sctest{%
2495 \bbl@xin@{\,\bbl@opt@strings,}{\,\bbl@sc@label,\bbl@sc@fontenc,}}%
2496 \fi
2497 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2498 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2499 \let\AfterBabelCommands\bbl@aftercmds
2500 \let\SetString\bbl@setstring
2501 \let\bbl@stringdef\bbl@encstring
2502 \else % ie, strings=value
2503 \bbl@sctest
2504 \ifin@
2505 \let\AfterBabelCommands\bbl@aftercmds
2506 \let\SetString\bbl@setstring
2507 \let\bbl@stringdef\bbl@provstring
2508 \fi\fi\fi
2509 \bbl@scswitch
2510 \ifx\bbl@G\@empty
2511 \def\SetString##1##2{%
2512 \bbl@error{Missing group for string \string##1}%
2513 {You must assign strings to some category, typically\\
2514 captions or extras, but you set none}}%
2515 \fi
2516 \ifx\@empty#1%
2517 \bbl@usehooks{defaultcommands}{}%
2518 \else
2519 \@expandtwoargs
2520 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2521 \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).

```

2522 \def\bbl@forlang#1#2{%
2523   \bbl@for#1\bbl@L{%
2524     \bbl@xin@{,#1,},{, \BabelLanguages,}%
2525     \ifin@#2\relax\fi}}
2526 \def\bbl@scswitch{%
2527   \bbl@forlang\bbl@tempa{%
2528     \ifx\bbl@G\@empty\else
2529       \ifx\SetString\@gobbleset\else
2530         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2531         \bbl@xin@{,\bbl@GL,},{,\bbl@screset,}%
2532         \ifin@else
2533           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2534           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2535         \fi
2536       \fi
2537     \fi}}
2538 \AtEndOfPackage{%
2539   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
2540   \let\bbl@scswitch\relax}
2541 \@onlypreamble\EndBabelCommands
2542 \def\EndBabelCommands{%
2543   \bbl@usehooks{stopcommands}{}%
2544   \endgroup
2545   \endgroup
2546   \bbl@scafter}
2547 \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.

```

2548 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2549   \bbl@forlang\bbl@tempa{%
2550     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2551     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2552     {\bbl@exp{%
2553       \global\@bbl@add\<\bbl@G\bbl@tempa>\@bbl@scset\#1\<\bbl@LC>}}}%
2554     {}}%
2555   \def\BabelString{#2}%
2556   \bbl@usehooks{stringprocess}{}%
2557   \expandafter\bbl@stringdef
2558   \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.

```

2559 \ifx\bbl@opt@strings\relax
2560   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2561   \bbl@patchuclc
2562   \let\bbl@encoded\relax
2563   \def\bbl@encoded@uclc#1{%
2564     \@inmathwarn#1%
2565     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2566       \expandafter\ifx\csname ?\string#1\endcsname\relax

```

```

2567      \TextSymbolUnavailable#1%
2568      \else
2569      \csname ?\string#1\endcsname
2570      \fi
2571      \else
2572      \csname\cf@encoding\string#1\endcsname
2573      \fi}
2574 \else
2575 \def\bbl@scset#1#2{\def#1{#2}}
2576 \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.

```

2577 <<*Macros local to BabelCommands>> ≡
2578 \def\SetStringLoop##1##2{%
2579   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2580   \count@\z@
2581   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2582     \advance\count@\@ne
2583     \toks@\expandafter{\bbl@tempa}%
2584     \bbl@exp{%
2585       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2586       \count@=\the\count@\relax}}}%
2587 <</Macros local to BabelCommands>>

```

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

```

2588 \def\bbl@aftercmds#1{%
2589   \toks@\expandafter{\bbl@scafter#1}%
2590   \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.

```

2591 <<*Macros local to BabelCommands>> ≡
2592 \newcommand\SetCase[3][]{%
2593   \bbl@patchuclc
2594   \bbl@forlang\bbl@tempa{%
2595     \expandafter\bbl@encstring
2596     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2597     \expandafter\bbl@encstring
2598     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2599     \expandafter\bbl@encstring
2600     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2601 <</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.

```

2602 <<*Macros local to BabelCommands>> ≡
2603 \newcommand\SetHyphenMap[1]{%
2604   \bbl@forlang\bbl@tempa{%
2605     \expandafter\bbl@stringdef
2606     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2607 <</Macros local to BabelCommands>>

```

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

```

2608 \newcommand\BabelLower[2]{% one to one.
2609   \ifnum\lccode#1=#2\else
2610     \babel@savevariable{\lccode#1}%
2611     \lccode#1=#2\relax
2612   \fi}
2613 \newcommand\BabelLowerMM[4]{% many-to-many
2614   \@tempcnta=#1\relax
2615   \@tempcntb=#4\relax
2616   \def\bb1@tempa{%
2617     \ifnum\@tempcnta>#2\else
2618       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2619       \advance\@tempcnta#3\relax
2620       \advance\@tempcntb#3\relax
2621       \expandafter\bb1@tempa
2622     \fi}%
2623   \bb1@tempa}
2624 \newcommand\BabelLowerMO[4]{% many-to-one
2625   \@tempcnta=#1\relax
2626   \def\bb1@tempa{%
2627     \ifnum\@tempcnta>#2\else
2628       \@expandtwoargs\BabelLower{\the\@tempcnta}{\#4}%
2629       \advance\@tempcnta#3
2630       \expandafter\bb1@tempa
2631     \fi}%
2632   \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2633 <<(*More package options)>> ≡
2634 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
2635 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
2636 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
2637 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
2638 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2639 <</More package options>>

```

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

```

2640 \AtEndOfPackage{%
2641   \ifx\bb1@opt@hyphenmap\undefined
2642     \bb1@xin@{,}{\bb1@language@opts}%
2643     \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
2644   \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.

```

2645 \bb1@trace{Macros related to glyphs}
2646 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2647   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2648   \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.

```

2649 \def\save@sf@q#1{\leavevmode
2650   \begingroup
2651     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2652   \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.

```
2653 \ProvideTextCommand{\quotedblbase}{OT1}{%
2654   \save@sf@q{\set@low@box{\textquotedblright\}%
2655     \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.

```
2656 \ProvideTextCommandDefault{\quotedblbase}{%
2657   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2658 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2659   \save@sf@q{\set@low@box{\textquoteright\}%
2660     \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.

```
2661 \ProvideTextCommandDefault{\quotesinglbase}{%
2662   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2663 \ProvideTextCommand{\guillemetleft}{OT1}{%
2664   \ifmmode
2665     \ll
2666   \else
2667     \save@sf@q{\nobreak
2668       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2669   \fi}
2670 \ProvideTextCommand{\guillemetright}{OT1}{%
2671   \ifmmode
2672     \gg
2673   \else
2674     \save@sf@q{\nobreak
2675       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2676   \fi}
2677 \ProvideTextCommand{\guillemotleft}{OT1}{%
2678   \ifmmode
2679     \ll
2680   \else
2681     \save@sf@q{\nobreak
2682       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2683   \fi}
2684 \ProvideTextCommand{\guillemotright}{OT1}{%
2685   \ifmmode
2686     \gg
2687   \else
2688     \save@sf@q{\nobreak
```

```

2689 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2690 \fi}

```

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

```

2691 \ProvideTextCommandDefault{\guillemetleft}{%
2692 \UseTextSymbol{OT1}{\guillemetleft}}
2693 \ProvideTextCommandDefault{\guillemetright}{%
2694 \UseTextSymbol{OT1}{\guillemetright}}
2695 \ProvideTextCommandDefault{\guillemotleft}{%
2696 \UseTextSymbol{OT1}{\guillemotleft}}
2697 \ProvideTextCommandDefault{\guillemotright}{%
2698 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2699 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2700 \ifmmode
2701 <%
2702 \else
2703 \save@sf@q{\nobreak
2704 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2705 \fi}
2706 \ProvideTextCommand{\guilsinglright}{OT1}{%
2707 \ifmmode
2708 >%
2709 \else
2710 \save@sf@q{\nobreak
2711 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2712 \fi}

```

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

```

2713 \ProvideTextCommandDefault{\guilsinglleft}{%
2714 \UseTextSymbol{OT1}{\guilsinglleft}}
2715 \ProvideTextCommandDefault{\guilsinglright}{%
2716 \UseTextSymbol{OT1}{\guilsinglright}}

```

9.12.2 Letters

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

```

2717 \DeclareTextCommand{\ij}{OT1}{%
2718 i\kern-0.02em\bbl@allowhyphens j}
2719 \DeclareTextCommand{\IJ}{OT1}{%
2720 I\kern-0.02em\bbl@allowhyphens J}
2721 \DeclareTextCommand{\ij}{T1}{\char188}
2722 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2723 \ProvideTextCommandDefault{\ij}{%
2724 \UseTextSymbol{OT1}{\ij}}
2725 \ProvideTextCommandDefault{\IJ}{%
2726 \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).

```

2727 \def\crrtic@{\hrule height0.1ex width0.3em}
2728 \def\crttic@{\hrule height0.1ex width0.33em}
2729 \def\ddj@{%
2730   \setbox0\hbox{d}\dimen@=\ht0
2731   \advance\dimen@1ex
2732   \dimen@.45\dimen@
2733   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2734   \advance\dimen@ii.5ex
2735   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2736 \def\DDJ@{%
2737   \setbox0\hbox{D}\dimen@=.55\ht0
2738   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2739   \advance\dimen@ii.15ex %      correction for the dash position
2740   \advance\dimen@ii-.15\fontdimen7\font %      correction for cmtt font
2741   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2742   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2743 %
2744 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2745 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2746 \ProvideTextCommandDefault{\dj}{%
2747   \UseTextSymbol{OT1}{\dj}}
2748 \ProvideTextCommandDefault{\DJ}{%
2749   \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.

```

2750 \DeclareTextCommand{\SS}{OT1}{\SS}
2751 \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 2752 \ProvideTextCommandDefault{\glq}{%
2753   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2754 \ProvideTextCommand{\grq}{T1}{%
2755   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2756 \ProvideTextCommand{\grq}{TU}{%
2757   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2758 \ProvideTextCommand{\grq}{OT1}{%
2759   \save@sf@q{\kern-.0125em
2760     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2761     \kern.07em\relax}}
2762 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```
\grqq 2763 \ProvideTextCommandDefault{\glqq}{%
2764 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2765 \ProvideTextCommand{\grqq}{T1}{%
2766 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2767 \ProvideTextCommand{\grqq}{TU}{%
2768 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2769 \ProvideTextCommand{\grqq}{OT1}{%
2770 \save@sf@q{\kern-.07em
2771 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2772 \kern.07em\relax}}
2773 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2774 \ProvideTextCommandDefault{\flq}{%
2775 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2776 \ProvideTextCommandDefault{\frq}{%
2777 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2778 \ProvideTextCommandDefault{\flqq}{%
2779 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2780 \ProvideTextCommandDefault{\frqq}{%
2781 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

9.12.4 Umlauts and tremas

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

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

```
2782 \def\umlauthigh{%
2783 \def\bbl@umlauta##1{\leavevmode\bggroup%
2784 \expandafter\accent\csname\fontencoding dqpos\endcsname
2785 ##1\bbl@allowhyphens\egroup}%
2786 \let\bbl@umlaute\bbl@umlauta}
2787 \def\umlautlow{%
2788 \def\bbl@umlauta{\protect\lower@umlaut}}
2789 \def\umlautelow{%
2790 \def\bbl@umlaute{\protect\lower@umlaut}}
2791 \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.

```
2792 \expandafter\ifx\csname U@D\endcsname\relax
2793 \csname newdimen\endcsname\U@D
2794 \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.

```
2795 \def\lower@umlaut#1{%
2796   \leavevmode\bggroup
2797     \U@D 1ex%
2798     {\setbox\z@\hbox{%
2799       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2800       \dimen@ -.45ex\advance\dimen@\ht\z@
2801       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2802       \expandafter\accent\csname\fontencoding dqpos\endcsname
2803       \fontdimen5\font\U@D #1%
2804     \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).

```
2805 \AtBeginDocument{%
2806   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2807   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2808   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2809   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2810   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2811   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2812   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2813   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2814   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2815   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2816   \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.

```
2817 \ifx\l@english\@undefined
2818   \chardef\l@english\z@
2819 \fi
2820 % The following is used to cancel rules in ini files (see Amharic).
2821 \ifx\l@babelnohyphens\@undefined
2822   \newlanguage\l@babelnohyphens
2823 \fi
```

9.13 Layout

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

```
2824 \bbl@trace{Bidi layout}
2825 \providecommand\IfBabelLayout[3]{#3}%
```

```

2826 \newcommand\BabelPatchSection[1]{%
2827   \@ifundefined{#1}{}{%
2828     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2829     \@namedef{#1}{%
2830       \ifstar{\bbl@presec@s{#1}}%
2831       {\@dblarg{\bbl@presec@x{#1}}}}}%
2832 \def\bbl@presec@x#1[#2]#3{%
2833   \bbl@exp{%
2834     \\select@language@x{\bbl@main@language}%
2835     \\bbl@cs{sspre@#1}%
2836     \\bbl@cs{ss@#1}%
2837     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2838     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2839     \\select@language@x{\language}}}%
2840 \def\bbl@presec@s#1#2{%
2841   \bbl@exp{%
2842     \\select@language@x{\bbl@main@language}%
2843     \\bbl@cs{sspre@#1}%
2844     \\bbl@cs{ss@#1}*%
2845     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2846     \\select@language@x{\language}}}%
2847 \IfBabelLayout{sectioning}%
2848   {\BabelPatchSection{part}%
2849    \BabelPatchSection{chapter}%
2850    \BabelPatchSection{section}%
2851    \BabelPatchSection{subsection}%
2852    \BabelPatchSection{subsubsection}%
2853    \BabelPatchSection{paragraph}%
2854    \BabelPatchSection{subparagraph}%
2855    \def\babel@toc#1{%
2856      \select@language@x{\bbl@main@language}}}%
2857 \IfBabelLayout{captions}%
2858   {\BabelPatchSection{caption}}}%

```

9.14 Load engine specific macros

```

2859 \bbl@trace{Input engine specific macros}
2860 \ifcase\bbl@engine
2861   \input txtbabel.def
2862 \or
2863   \input luababel.def
2864 \or
2865   \input xebabel.def
2866 \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.

```

2867 \bbl@trace{Creating languages and reading ini files}
2868 \newcommand\babelprovide[2][{}]{%
2869   \let\bbl@savelangname\language
2870   \edef\bbl@savelocaleid{\the\localeid}%
2871   % Set name and locale id
2872   \edef\language{#2}%
2873   % \global\@namedef{\bbl@lcname@#2}{#2}%
2874   \bbl@id@assign

```

```

2875 \let\bb1@KVP@captions\@nil
2876 \let\bb1@KVP@date\@nil
2877 \let\bb1@KVP@import\@nil
2878 \let\bb1@KVP@main\@nil
2879 \let\bb1@KVP@script\@nil
2880 \let\bb1@KVP@language\@nil
2881 \let\bb1@KVP@hyphenrules\@nil % only for provide@new
2882 \let\bb1@KVP@mapfont\@nil
2883 \let\bb1@KVP@maparabic\@nil
2884 \let\bb1@KVP@mapdigits\@nil
2885 \let\bb1@KVP@intraspace\@nil
2886 \let\bb1@KVP@intrapenalty\@nil
2887 \let\bb1@KVP@onchar\@nil
2888 \let\bb1@KVP@alph\@nil
2889 \let\bb1@KVP@Alph\@nil
2890 \let\bb1@KVP@labels\@nil
2891 \bb1@csarg\let{KVP@labels*}\@nil
2892 \bb1@forkv{#1}{% TODO - error handling
2893   \in@{/}{##1}%
2894   \ifin@
2895     \bb1@renewinikey##1\@{##2}%
2896   \else
2897     \bb1@csarg\def{KVP@##1}{##2}%
2898   \fi}%
2899 % == import, captions ==
2900 \ifx\bb1@KVP@import\@nil\else
2901   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
2902   {\ifx\bb1@initload\relax
2903     \begingroup
2904       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
2905       \bb1@input@texini{#2}%
2906     \endgroup
2907   \else
2908     \xdef\bb1@KVP@import{\bb1@initload}%
2909   \fi}%
2910 {}%
2911 \fi
2912 \ifx\bb1@KVP@captions\@nil
2913   \let\bb1@KVP@captions\bb1@KVP@import
2914 \fi
2915 % Load ini
2916 \bb1@ifunset{date#2}%
2917   {\bb1@provide@new{#2}}%
2918   {\bb1@ifblank{#1}%
2919     {\bb1@error
2920       {If you want to modify `#2' you must tell how in\\
2921       the optional argument. See the manual for the\\
2922       available options.}%
2923       {Use this macro as documented}}%
2924     {\bb1@provide@renew{#2}}}%
2925 % Post tasks
2926 \bb1@ifunset{\bb1@extracaps@#2}%
2927   {\bb1@exp{\bb1@babelensure[exclude=\\today]{#2}}%
2928   {\toks@%
2929     \expandafter\expandafter\expandafter
2930     {\csname \bb1@extracaps@#2\endcsname}%
2931     \bb1@exp{\bb1@babelensure[exclude=\\today,include=\the\toks@]{#2}}%
2932   \bb1@ifunset{\bb1@ensure@%
2933     {\bb1@exp{%
2934       \\DeclareRobustCommand\<\bb1@ensure@%

```

```

2934     \\foreignlanguage{\language}%
2935     {###1}}}%
2936     }%
2937 \bbl@exp{%
2938     \\bbl@toglobal\<bbl@ensure@\language>%
2939     \\bbl@toglobal\<bbl@ensure@\language\space>%
2940 % At this point all parameters are defined if 'import'. Now we
2941 % execute some code depending on them. But what about if nothing was
2942 % imported? We just load the very basic parameters.
2943 \bbl@load@basic{#2}%
2944 % == script, language ==
2945 % Override the values from ini or defines them
2946 \ifx\bbl@KVP@script\@nil\else
2947     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2948 \fi
2949 \ifx\bbl@KVP@language\@nil\else
2950     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2951 \fi
2952 % == onchar ==
2953 \ifx\bbl@KVP@onchar\@nil\else
2954     \bbl@luahyphenate
2955     \directlua{
2956         if Babel.locale_mapped == nil then
2957             Babel.locale_mapped = true
2958             Babel.linebreaking.add_before(Babel.locale_map)
2959             Babel.loc_to_scr = {}
2960             Babel.chr_to_loc = Babel.chr_to_loc or {}
2961         end}%
2962 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2963 \ifin@
2964     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2965         \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2966     \fi
2967     \bbl@exp{\\bbl@add\\bbl@starthyphens
2968         {\bbl@patterns@lua{\language}}}%
2969     % TODO - error/warning if no script
2970     \directlua{
2971         if Babel.script_blocks['\bbl@cl{sbc}'] then
2972             Babel.loc_to_scr[\the\localeid] =
2973                 Babel.script_blocks['\bbl@cl{sbc}']
2974             Babel.locale_props[\the\localeid].lc = \the\localeid\space
2975             Babel.locale_props[\the\localeid].lg = \the\nameuse{1l\language}\space
2976         end
2977     }%
2978 \fi
2979 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2980 \ifin@
2981     \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}}}%
2982     \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}}}%
2983     \directlua{
2984         if Babel.script_blocks['\bbl@cl{sbc}'] then
2985             Babel.loc_to_scr[\the\localeid] =
2986                 Babel.script_blocks['\bbl@cl{sbc}']
2987         end}%
2988 \ifx\bbl@mapselect\@undefined
2989     \AtBeginDocument{%
2990         \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2991         {\selectfont}}%
2992     \def\bbl@mapselect{%

```



```

2993         \let\bbbl@mapselect\relax
2994         \edef\bbbl@prefontid{\fontid\font}}%
2995     \def\bbbl@mapdir##1{%
2996         {\def\language{##1}%
2997         \let\bbbl@ifrestoring\@firstoftwo % To avoid font warning
2998         \bbbl@switchfont
2999         \directlua{
3000             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
3001             [\bbbl@prefontid'] = \fontid\font\space}}}%
3002     \fi
3003     \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
3004     \fi
3005     % TODO - catch non-valid values
3006     \fi
3007     % == mapfont ==
3008     % For bidi texts, to switch the font based on direction
3009     \ifx\bbbl@KVP@mapfont\@nil\else
3010         \bbbl@ifsamestring{\bbbl@KVP@mapfont}{direction}}{%
3011             {\bbbl@error{Option '\bbbl@KVP@mapfont' unknown for\
3012                 mapfont. Use 'direction'.%
3013                 {See the manual for details.}}}%
3014         \bbbl@ifunset{\bbbl@sys@\language}{\bbbl@provide@sys@\language}}{%
3015         \bbbl@ifunset{\bbbl@wdir@\language}{\bbbl@provide@dirs@\language}}{%
3016         \ifx\bbbl@mapselect\@undefined
3017             \AtBeginDocument{%
3018                 \expandafter\bbbl@add\csname selectfont \endcsname{\bbbl@mapselect}}%
3019                 {\selectfont}}%
3020         \def\bbbl@mapselect{%
3021             \let\bbbl@mapselect\relax
3022             \edef\bbbl@prefontid{\fontid\font}}%
3023         \def\bbbl@mapdir##1{%
3024             {\def\language{##1}%
3025             \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
3026             \bbbl@switchfont
3027             \directlua{Babel.fontmap
3028                 [\the\csname bbl@wdir@##1\endcsname]%
3029                 [\bbbl@prefontid]=\fontid\font}}}%
3030         \fi
3031         \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
3032     \fi
3033     % == intraspace, intrapenalty ==
3034     % For CJK, East Asian, Southeast Asian, if interspace in ini
3035     \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
3036         \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
3037     \fi
3038     \bbbl@provide@intraspace
3039     % == hyphenate.other.locale ==
3040     \bbbl@ifunset{\bbbl@hyotl@\language}{}%
3041     {\bbbl@csarg\bbbl@replace{\hyotl@\language}{ }{ },}%
3042     \bbbl@startcommands*{\language}{}%
3043     \bbbl@csarg\bbbl@foreach{\hyotl@\language}{%
3044         \ifcase\bbbl@engine
3045             \ifnum##1<257
3046                 \SetHyphenMap{\BabelLower{##1}{##1}}%
3047             \fi
3048             \else
3049                 \SetHyphenMap{\BabelLower{##1}{##1}}%
3050             \fi}%
3051     \bbbl@endcommands}%

```

```

3052 % == hyphenate.other.script ==
3053 \bbl@ifunset{bbl@hyots@languagename}{}%
3054 {\bbl@csarg\bbl@replace{hyots@languagename}{ }{,}%
3055 \bbl@csarg\bbl@foreach{hyots@languagename}{%
3056 \ifcase\bbl@engine
3057 \ifnum##1<257
3058 \global\lccode##1=##1\relax
3059 \fi
3060 \else
3061 \global\lccode##1=##1\relax
3062 \fi}}%
3063 % == maparabic ==
3064 % Native digits, if provided in ini (TeX level, xe and lua)
3065 \ifcase\bbl@engine\else
3066 \bbl@ifunset{bbl@dgnat@languagename}{}%
3067 {\expandafter\ifx\csname bbl@dgnat@languagename\endcsname\@empty\else
3068 \expandafter\expandafter\expandafter
3069 \bbl@setdigits\csname bbl@dgnat@languagename\endcsname
3070 \ifx\bbl@KVP@maparabic\@nil\else
3071 \ifx\bbl@latinarabic\@undefined
3072 \expandafter\let\expandafter\@arabic
3073 \csname bbl@counter@languagename\endcsname
3074 \else % ie, if layout=counters, which redefines \@arabic
3075 \expandafter\let\expandafter\bbl@latinarabic
3076 \csname bbl@counter@languagename\endcsname
3077 \fi
3078 \fi
3079 \fi}%
3080 \fi
3081 % == mapdigits ==
3082 % Native digits (lua level).
3083 \ifodd\bbl@engine
3084 \ifx\bbl@KVP@mapdigits\@nil\else
3085 \bbl@ifunset{bbl@dgnat@languagename}{}%
3086 {\RequirePackage{luatexbase}%
3087 \bbl@activate@preotf
3088 \directlua{
3089 Babel = Babel or {} %% -> presets in luababel
3090 Babel.digits_mapped = true
3091 Babel.digits = Babel.digits or {}
3092 Babel.digits[\the\localeid] =
3093 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3094 if not Babel.numbers then
3095 function Babel.numbers(head)
3096 local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3097 local GLYPH = node.id'glyph'
3098 local inmath = false
3099 for item in node.traverse(head) do
3100 if not inmath and item.id == GLYPH then
3101 local temp = node.get_attribute(item, LOCALE)
3102 if Babel.digits[temp] then
3103 local chr = item.char
3104 if chr > 47 and chr < 58 then
3105 item.char = Babel.digits[temp][chr-47]
3106 end
3107 end
3108 elseif item.id == node.id'math' then
3109 inmath = (item.subtype == 0)
3110 end

```

```

3111         end
3112     return head
3113 end
3114 end
3115 }}%
3116 \fi
3117 \fi
3118 % == alph, Alph ==
3119 % What if extras<lang> contains a \babel@save\@alph? It won't be
3120 % restored correctly when exiting the language, so we ignore
3121 % this change with the \bbl@alph@saved trick.
3122 \ifx\bbl@KVP@alph\@nil\else
3123     \toks@\expandafter\expandafter\expandafter{%
3124         \csname extras\language\endcsname}%
3125     \bbl@exp{%
3126         \def\<extras\language>{%
3127             \let\\bbl@alph@saved\\@alph
3128             \the\toks@
3129             \let\\@alph\\bbl@alph@saved
3130             \\babel@save\\@alph
3131             \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
3132     \fi
3133 \ifx\bbl@KVP@Alph\@nil\else
3134     \toks@\expandafter\expandafter\expandafter{%
3135         \csname extras\language\endcsname}%
3136     \bbl@exp{%
3137         \def\<extras\language>{%
3138             \let\\bbl@Alph@saved\\@Alph
3139             \the\toks@
3140             \let\\@Alph\\bbl@Alph@saved
3141             \\babel@save\\@Alph
3142             \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3143     \fi
3144 % == require.babel in ini ==
3145 % To load or reload the babel-*.tex, if require.babel in ini
3146 \bbl@ifunset{bbl@rqtex\language}{}%
3147 {\expandafter\ifx\csname bbl@rqtex\language\endcsname\@empty\else
3148     \let\BabelBeforeIni@gobbletwo
3149     \chardef\atcatcode=\catcode\@
3150     \catcode\@=11\relax
3151     \bbl@input@texini{\bbl@cs{rqtex\language}}%
3152     \catcode\@=\atcatcode
3153     \let\atcatcode\relax
3154     \fi}%
3155 % == main ==
3156 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3157     \let\language\bbl@savelangname
3158     \chardef\localeid\bbl@savelocaleid\relax
3159 \fi}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T_EX. Non-digits characters are kept. The first macro is the generic “localized” command.

```

3160 \def\bbl@setdigits#1#2#3#4#5{%
3161     \bbl@exp{%
3162         \def\<\language digits>####1{%          ie, \langdigits
3163             \<bbl@digits@\language>####1\\@nil}%
3164         \let\<bbl@cntr@digits@\language>\<\language digits>%
3165         \def\<\language counter>####1{%          ie, \langcounter

```

[illegible]

```

3191 \def\bbl@provide@new#1{%
3192   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3193   \@namedef{extras#1}{}%
3194   \@namedef{noextras#1}{}%
3195   \bbl@startcommands*{#1}{captions}%
3196     \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
3197       \def\bbl@tempb##1{%            elt for \bbl@captionslist
3198         \ifx##1\@empty\else
3199           \bbl@exp{%
3200             \\SetString\\##1{%
3201               \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3202             \expandafter\bbl@tempb
3203           \fi}%
3204       \expandafter\bbl@tempb\bbl@captionslist\@empty
3205     \else
3206       \ifx\bbl@initoload\relax
3207         \bbl@read@ini{\bbl@KVP@captions}0% Here letters cat = 11
3208       \else
3209         \bbl@read@ini{\bbl@initoload}0% Here all letters cat = 11
3210       \fi
3211       \bbl@after@ini
3212       \bbl@savestrings
3213     \fi
3214   \StartBabelCommands*{#1}{date}%
3215   \ifx\bbl@KVP@import\@nil
3216     \bbl@exp{%
3217       \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
3218   \else
3219     \bbl@savetoday
3220     \bbl@savedate
3221   \fi
3222   \bbl@endcommands

```

```

3223 \bbl@load@basic{#1}%
3224 \bbl@exp{%
3225   \gdef\<#1hyphenmins>{%
3226     {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3227     {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3228 \bbl@provide@hyphens{#1}%
3229 \ifx\bbl@KVP@main\@nil\else
3230   \expandafter\main@language\expandafter{#1}%
3231 \fi}
3232 \def\bbl@provide@renew#1{%
3233   \ifx\bbl@KVP@captions\@nil\else
3234     \StartBabelCommands*{#1}{captions}%
3235     \bbl@read@ini{\bbl@KVP@captions}0%   Here all letters cat = 11
3236     \bbl@after@ini
3237     \bbl@savestrings
3238     \EndBabelCommands
3239 \fi
3240 \ifx\bbl@KVP@import\@nil\else
3241   \StartBabelCommands*{#1}{date}%
3242   \bbl@savetoday
3243   \bbl@savestate
3244   \EndBabelCommands
3245 \fi
3246 % == hyphenrules ==
3247 \bbl@provide@hyphens{#1}}
3248 % Load the basic parameters (ids, typography, counters, and a few
3249 % more), while captions and dates are left out. But it may happen some
3250 % data has been loaded before automatically, so we first discard the
3251 % saved values.
3252 \def\bbl@load@basic#1{%
3253   \bbl@ifunset{\bbl@inidata@\language\name}{}%
3254   {\getlocaleproperty\bbl@tempa{\language\name}{identification/load.level}%
3255   \ifcase\bbl@tempa\else
3256     \bbl@csarg\let{lname@\language\name}\relax
3257   \fi}%
3258   \bbl@ifunset{\bbl@lname@#1}%
3259   {\def\BabelBeforeIni##1##2{%
3260     \begingroup
3261       \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3262       \catcode`\;=12 \catcode`\|=12 \catcode`\%=14
3263       \let\bbl@ini@captions@aux\@gobbletwo
3264       \def\bbl@inidata ###1.###2.###3.###4\relax ###5###6{}%
3265       \bbl@read@ini{##1}0%
3266       \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3267       \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3268       \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3269       \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3270       \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3271       \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3272       \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3273       \bbl@exportkey{intsp}{typography.intraspace}{}%
3274       \bbl@exportkey{chnrg}{characters.ranges}{}%
3275       \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3276       \ifx\bbl@initoload\relax\endinput\fi
3277     \endgroup}%
3278   \begingroup % boxed, to avoid extra spaces:
3279   \ifx\bbl@initoload\relax
3280     \bbl@input@texini{#1}%
3281   \else

```

```

3282      \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3283      \fi
3284      \endgroup}%
3285      {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

3286 \def\bbl@provide@hyphens#1{%
3287   \let\bbl@tempa\relax
3288   \ifx\bbl@KVP@hyphenrules\@nil\else
3289     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3290     \bbl@foreach\bbl@KVP@hyphenrules{%
3291       \ifx\bbl@tempa\relax % if not yet found
3292         \bbl@ifsamestring{##1}{+}%
3293         {{\bbl@exp{\addlanguage\<l@##1>}}}%
3294         {}%
3295         \bbl@ifunset{l@##1}%
3296         {}%
3297         {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3298       \fi}%
3299   \fi
3300   \ifx\bbl@tempa\relax % if no opt or no language in opt found
3301     \ifx\bbl@KVP@import\@nil
3302       \ifx\bbl@initoload\relax\else
3303         \bbl@exp{% and hyphenrules is not empty
3304           \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3305           {}%
3306           {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3307       \fi
3308       \else % if importing
3309         \bbl@exp{% and hyphenrules is not empty
3310           \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3311           {}%
3312           {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3313       \fi
3314     \fi
3315     \bbl@ifunset{bbl@tempa}% ie, relax or undefined
3316     {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
3317       {\bbl@exp{\adddialect\<l@#1>\language}}%
3318       {}% so, l@<lang> is ok - nothing to do
3319       {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
3320

```

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.

```

3321 \ifx\bbl@readstream\@undefined
3322   \csname newread\endcsname\bbl@readstream
3323 \fi
3324 \def\bbl@input@texini#1{%
3325   \bbl@bsphack
3326   \bbl@exp{%
3327     \catcode\%%=14
3328     \lowercase{\InputIfFileExists{babel-#1.tex}}{}%
3329     \catcode\%%=\the\catcode\%\relax}%
3330   \bbl@esphack}
3331 \def\bbl@inipreread#1=#2\@{%
3332   \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3333   \bbl@trim\toks@{#2}%
3334   % Move trims here ??
3335   \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%

```

```

3336 {\bbl@exp{%
3337     \\g@addto@macro\\bbl@inidata{%
3338         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3339     \expandafter\bbl@inireader\bbl@tempa=#2\@@}%
3340     }%}%
3341 \def\bbl@fetch@ini#1#2{%
3342     \bbl@exp{\def\\bbl@inidata{%
3343         \\bbl@elt{identification}{tag.ini}{#1}%
3344         \\bbl@elt{identification}{load.level}{#2}}}%
3345     \openin\bbl@readstream=babel-#1.ini
3346     \ifeof\bbl@readstream
3347         \bbl@error
3348         {There is no ini file for the requested language\\%
3349         (#1). Perhaps you misspelled it or your installation\\%
3350         is not complete.}%
3351         {Fix the name or reinstall babel.}%
3352     \else
3353         \bbl@info{Importing
3354             \ifcase#2 \or font and identification \or basic \fi
3355             data for \language\\%
3356             from babel-#1.ini. Reported}%
3357     \loop
3358     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3359     \endlinechar\m@ne
3360     \read\bbl@readstream to \bbl@line
3361     \endlinechar\^^M
3362     \ifx\bbl@line\empty\else
3363         \expandafter\bbl@iniline\bbl@line\bbl@iniline
3364     \fi
3365     \repeat
3366 \fi}
3367 \def\bbl@read@ini#1#2{%
3368     \bbl@csarg\edef{lini@\language}{#1}%
3369     \let\bbl@section\empty
3370     \let\bbl@savestrings\empty
3371     \let\bbl@savetoday\empty
3372     \let\bbl@savestate\empty
3373     \let\bbl@inireader\bbl@iniskip
3374     \bbl@fetch@ini{#1}{#2}%
3375     \bbl@foreach\bbl@renewlist{%
3376         \bbl@ifunset{\bbl@renew@##1}{\bbl@inisec[##1]\@@}%
3377     \global\let\bbl@renewlist\empty
3378     % Ends last section. See \bbl@inisec
3379     \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
3380     \bbl@cs{renew@\bbl@section}%
3381     \global\bbl@csarg\let{renew@\bbl@section}\relax
3382     \bbl@cs{secpost@\bbl@section}%
3383     \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3384     \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
3385     \bbl@tglobal\bbl@ini@loaded}
3386 \def\bbl@iniline#1\bbl@iniline{%
3387     \@ifnextchar[\bbl@inisec{\@ifnextchar\bbl@iniskip\bbl@inipreread}#1\@@}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start. By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

3388 \def\bbl@iniskip#1\@@{%          if starts with ;
3389 \def\bbl@inisec[#1]#2\@@{%      if starts with opening bracket

```

```

3390 \def\bb1@elt##1##2{%
3391   \expandafter\toks@\expandafter{%
3392     \expandafter{\bb1@section}{##1}{##2}}%
3393   \bb1@exp{%
3394     \g@addto@macro\bb1@inidata{\bb1@elt\the\toks@}}%
3395   \bb1@inireader##1=##2\@}%
3396 \bb1@cs{renew\bb1@section}%
3397 \global\bb1@csarg\let{renew\bb1@section}\relax
3398 \bb1@cs{secpost\bb1@section}%
3399 % The previous code belongs to the previous section.
3400 % -----
3401 % Now start the current one.
3402 \in@{=date.}{#1}%
3403 \ifin@
3404   \lowercase{\def\bb1@tempa{#1}}%
3405   \bb1@replace\bb1@tempa{=date.gregorian.}{}%
3406   \bb1@replace\bb1@tempa{=date.}{}%
3407   \in@{.licr=}{#1}%
3408   \ifin@
3409     \ifcase\bb1@engine
3410       \bb1@replace\bb1@tempa{.licr=}{}%
3411     \else
3412       \let\bb1@tempa\relax
3413     \fi
3414   \fi
3415   \ifx\bb1@tempa\relax\else
3416     \bb1@replace\bb1@tempa{=}{}%
3417     \bb1@exp{%
3418       \def\<bb1@inikv@#1>####1=####2\\@{%
3419         \bb1@inidata####1...\relax{####2}{\bb1@tempa}}}%
3420     \fi
3421   \fi
3422 \def\bb1@section{#1}%
3423 \def\bb1@elt##1##2{%
3424   \@namedef{bb1@KVP@#1/#1}{}}%
3425 \bb1@cs{renew@#1}%
3426 \bb1@cs{secpre@#1}% pre-section `hook'
3427 \bb1@ifunset{bb1@inikv@#1}%
3428   {\let\bb1@inireader\bb1@iniskip}%
3429   {\bb1@exp{\let\bb1@inireader\<bb1@inikv@#1>}}
3430 \let\bb1@renewlist\@empty
3431 \def\bb1@renewinikv#1/#2\@#3{%
3432   \bb1@ifunset{bb1@renew@#1}%
3433     {\bb1@add@list\bb1@renewlist{#1}}%
3434     {}%
3435   \bb1@csarg\bb1@add{renew@#1}{\bb1@elt{#2}{#3}}

```

Reads a key=val line and stores the trimmed val in \bb1@@kv@<section>.<key>.

```

3436 \def\bb1@inikv#1=#2\@{%      key=value
3437   \bb1@trim\def\bb1@tempa{#1}%
3438   \bb1@trim\toks@{#2}%
3439   \bb1@csarg\edef{@kv@\bb1@section.\bb1@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.

```

3440 \def\bb1@exportkey#1#2#3{%
3441   \bb1@ifunset{bb1@@kv@#2}%
3442     {\bb1@csarg\gdef{#1\language}\@#3}%
3443     {\expandafter\ifx\csname bb1@@kv@#2\endcsname\@empty

```



```

3444     \bbl@csarg\gdef{#1@\language}\{#3}%
3445     \else
3446     \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
3447     \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.

```

3448 \def\bbl@iniwarning#1{%
3449   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3450   {\bbl@warning{%
3451     From babel-\bbl@cs{lini@\language}.ini:\%
3452     \bbl@cs{@kv@identification.warning#1}\%
3453     Reported }}}
3454 \let\bbl@inikv@identification\bbl@inikv
3455 \def\bbl@secpost@identification{%
3456   \bbl@iniwarning}%
3457   \ifcase\bbl@engine
3458     \bbl@iniwarning{.pdflatex}%
3459   \or
3460     \bbl@iniwarning{.lualatex}%
3461   \or
3462     \bbl@iniwarning{.xelatex}%
3463   \fi%
3464   \bbl@exportkey{elname}{identification.name.english}{}%
3465   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3466     {\csname bbl@elname@\language\endcsname}}%
3467   \bbl@exportkey{lbcpl}{identification.tag.bcp47}{}% TODO
3468   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3469   \bbl@exportkey{esname}{identification.script.name}{}%
3470   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3471     {\csname bbl@esname@\language\endcsname}}%
3472   \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3473   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3474   \ifbbl@bcptoname
3475     \bbl@csarg\xdef{bcp@map@bbl@cl{lbcpl}}{\language}%
3476   \fi}
3477 \let\bbl@inikv@typography\bbl@inikv
3478 \let\bbl@inikv@characters\bbl@inikv
3479 \let\bbl@inikv@numbers\bbl@inikv
3480 \def\bbl@inikv@counters#1=#2\@{%
3481   \bbl@ifsamestring{#1}{digits}%
3482   {\bbl@error{The counter name 'digits' is reserved for mapping\%
3483     decimal digits}%
3484     {Use another name.}}%
3485   {}}%
3486   \def\bbl@tempc{#1}%
3487   \bbl@trim@def{\bbl@tempb*}{#2}%
3488   \in@{.1$}{#1$}%
3489   \ifin@
3490     \bbl@replace\bbl@tempc{.1}{}%
3491     \bbl@csarg\protected@xdef{cntr@bbl@tempc @\language}{%
3492       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3493   \fi
3494   \in@{.F.}{#1}%
3495   \ifin@else\in@{.S.}{#1}\fi
3496   \ifin@
3497     \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%

```

```

3498 \else
3499 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3500 \expandafter\bbl@buildifcase\bbl@tempb* \ % Space after \
3501 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3502 \fi}
3503 \def\bbl@after@ini{%
3504 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3505 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3506 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3507 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3508 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3509 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3510 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3511 \bbl@exportkey{intsp}{typography.intraspace}{}%
3512 \bbl@exportkey{jstfy}{typography.justify}{w}%
3513 \bbl@exportkey{chrng}{characters.ranges}{}%
3514 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3515 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3516 \bbl@toglobal\bbl@savetoday
3517 \bbl@toglobal\bbl@savestate}

```

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

```

3518 \ifcase\bbl@engine
3519 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3520 \bbl@ini@captions@aux{#1}{#2}}
3521 \else
3522 \def\bbl@inikv@captions#1=#2\@@{%
3523 \bbl@ini@captions@aux{#1}{#2}}
3524 \fi

```

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

```

3525 \def\bbl@ini@captions@aux#1#2{%
3526 \bbl@trim\def\bbl@tempa{#1}%
3527 \bbl@xin@{.template}{\bbl@tempa}%
3528 \ifin@
3529 \bbl@replace\bbl@tempa{.template}{}%
3530 \def\bbl@toreplace{#2}%
3531 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3532 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3533 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3534 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}%
3535 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}%
3536 \bbl@xin@{,\bbl@tempa,}{,chapter,}%
3537 \ifin@
3538 \bbl@patchchapter
3539 \global\bbl@csarg\let{chapfmt@\language}\bbl@toreplace
3540 \fi
3541 \bbl@xin@{,\bbl@tempa,}{,appendix,}%
3542 \ifin@
3543 \bbl@patchchapter
3544 \global\bbl@csarg\let{appxfmt@\language}\bbl@toreplace
3545 \fi
3546 \bbl@xin@{,\bbl@tempa,}{,part,}%
3547 \ifin@
3548 \bbl@patchpart
3549 \global\bbl@csarg\let{partfmt@\language}\bbl@toreplace
3550 \fi

```

```

3551 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3552 \ifin@
3553 \toks@ \expandafter{\bbl@toreplace}%
3554 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3555 \fi
3556 \else
3557 \bbl@ifblank{#2}%
3558 {\bbl@exp{%
3559 \toks@{\bbl@nocaption{\bbl@tempa}{\language name\bbl@tempa name}}}%
3560 {\bbl@trim\toks@{#2}}}%
3561 \bbl@exp{%
3562 \bbl@add\bbl@savestrings{%
3563 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3564 \toks@ \expandafter{\bbl@captionslist}%
3565 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3566 \ifin@ \else
3567 \bbl@exp{%
3568 \bbl@add\<\bbl@extracaps@\language name>{\<\bbl@tempa name>}%
3569 \bbl@to global\<\bbl@extracaps@\language name>}%
3570 \fi
3571 \fi}

```

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

```

3572 \def\bbl@list@the{%
3573 part,chapter,section,subsection,subsubsection,paragraph,%
3574 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3575 table,page,footnote,mpfootnote,mpfn}
3576 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3577 \bbl@ifunset{\bbl@map@#1@\language name}%
3578 {\@nameuse{#1}}%
3579 {\@nameuse{\bbl@map@#1@\language name}}}
3580 \def\bbl@inikv@labels#1=#2\@{%
3581 \in@{.map}{#1}%
3582 \ifin@
3583 \ifx\bbl@KVP@labels\@nil\else
3584 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3585 \ifin@
3586 \def\bbl@tempc{#1}%
3587 \bbl@replace\bbl@tempc{.map}{}%
3588 \in@{,#2,}{,arabic,roman,Roman,alpha,Alph,fnsymbol,}%
3589 \bbl@exp{%
3590 \gdef\<\bbl@map@\bbl@tempc @\language name>%
3591 {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3592 \bbl@foreach\bbl@list@the{%
3593 \bbl@ifunset{the##1}{}%
3594 {\bbl@exp{\let\bbl@tempd\<the##1>}%
3595 \bbl@exp{%
3596 \bbl@sreplace\<the##1>%
3597 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3598 \bbl@sreplace\<the##1>%
3599 {\<\@empty @\bbl@tempc>\<c##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3600 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3601 \toks@ \expandafter\expandafter\expandafter{%
3602 \csname the##1\endcsname}%
3603 \expandafter\edef\csname the##1\endcsname{\the\toks@}%
3604 \fi}}%
3605 \fi
3606 \fi

```

```

3607 %
3608 \else
3609 %
3610 % The following code is still under study. You can test it and make
3611 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3612 % language dependent.
3613 \in@{enumerate.}{#1}%
3614 \ifin@
3615   \def\bbl@tempa{#1}%
3616   \bbl@replace\bbl@tempa{enumerate.}{}%
3617   \def\bbl@toreplace{#2}%
3618   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3619   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3620   \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}%
3621   \toks@ \expandafter{\bbl@toreplace}%
3622   \bbl@exp{%
3623     \\bbl@add<extras\language>{%
3624       \\babel@save<labelenum\romannumeral\bbl@tempa>%
3625       \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3626       \\bbl@tglobal\<extras\language>}%
3627   \fi
3628 \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.

```

3629 \def\bbl@chapttype{chap}
3630 \ifx\@makechapterhead\undefined
3631   \let\bbl@patchchapter\relax
3632 \else\ifx\thechapter\undefined
3633   \let\bbl@patchchapter\relax
3634 \else\ifx\ps@headings\undefined
3635   \let\bbl@patchchapter\relax
3636 \else
3637   \def\bbl@patchchapter{%
3638     \global\let\bbl@patchchapter\relax
3639     \bbl@add\appendix{\def\bbl@chapttype{appx}}% Not harmful, I hope
3640     \bbl@tglobal\appendix
3641     \bbl@sreplace\ps@headings
3642       {\@chapapp\ thechapter}%
3643       {\bbl@chapterformat}%
3644     \bbl@tglobal\ps@headings
3645     \bbl@sreplace\chaptermark
3646       {\@chapapp\ thechapter}%
3647       {\bbl@chapterformat}%
3648     \bbl@tglobal\chaptermark
3649     \bbl@sreplace\@makechapterhead
3650       {\@chapapp\space\thechapter}%
3651       {\bbl@chapterformat}%
3652     \bbl@tglobal\@makechapterhead
3653     \gdef\bbl@chapterformat{%
3654       \bbl@ifunset{\bbl@\bbl@chapttype fmt@\language}%
3655       {\@chapapp\space\thechapter}
3656       {\@nameuse{\bbl@\bbl@chapttype fmt@\language}}}%
3657   \fi\fi\fi
3658 \ifx\@part\undefined
3659   \let\bbl@patchpart\relax
3660 \else

```

```

3661 \def\bbl@patchpart{%
3662   \global\let\bbl@patchpart\relax
3663   \bbl@sreplace\@part
3664   {\partname\nobreakspace\thepart}%
3665   {\bbl@partformat}%
3666   \bbl@tglobal\@part
3667   \gdef\bbl@partformat{%
3668     \bbl@ifunset\bbl@partfmt@\languagename}%
3669     {\partname\nobreakspace\thepart}
3670     {\@nameuse\bbl@partfmt@\languagename}}}%
3671 \fi

```

Date. TODO. Document

```

3672 % Arguments are _not_ protected.
3673 \let\bbl@calendar\@empty
3674 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3675 \def\bbl@cased{% TODO. Move
3676   \ifx\oe\OE
3677     \expandafter\in@\expandafter
3678     {\expandafter\OE\expandafter}\expandafter{\oe}%
3679     \ifin@
3680     \bbl@afterelse\expandafter\MakeUppercase
3681     \else
3682     \bbl@afterfi\expandafter\MakeLowercase
3683     \fi
3684   \else
3685     \expandafter\@firstofone
3686     \fi}
3687 \def\bbl@localedate#1#2#3#4{%
3688   \begingroup
3689   \ifx\@empty#1\@empty\else
3690     \let\bbl@ld@calendar\@empty
3691     \let\bbl@ld@variant\@empty
3692     \edef\bbl@tempa{\zap@space#1 \@empty}%
3693     \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld##1}{##2}}%
3694     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3695     \edef\bbl@calendar{%
3696       \bbl@ld@calendar
3697       \ifx\bbl@ld@variant\@empty\else
3698       .\bbl@ld@variant
3699       \fi}%
3700     \bbl@replace\bbl@calendar{gregorian}}}%
3701   \fi
3702   \bbl@cased
3703   {\@nameuse\bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3704 \endgroup}
3705 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3706 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3707   \bbl@trim\def\bbl@tempa{#1.#2}%
3708   \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3709   {\bbl@trim\def\bbl@tempa{#3}%
3710     \bbl@trim\toks@{#5}%
3711     \@temptokena\expandafter{\bbl@savedate}%
3712     \bbl@exp{% Reverse order - in ini last wins
3713       \def\\bbl@savedate{%
3714         \\SetString\<month\romannumeral\bbl@tempa#6name>\the\toks@}%
3715         \the\@temptokena}}}%
3716   {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3717     {\lowercase{\def\bbl@tempb{#6}}}%

```

```

3718 \bbl@trim@def\bbl@toreplace{#5}%
3719 \bbl@TG@date
3720 \bbl@ifunset\bbl@date@\language @}%
3721 {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3722 % TODO. Move to a better place.
3723 \bbl@exp{%
3724 \gdef\<\language date>{\protect\<\language date >}%
3725 \gdef\<\language date >####1####2####3{%
3726 \\\bbl@usedategroupttrue
3727 \<bbl@ensure@\language>{%
3728 \\\localedate{####1}{####2}{####3}}}%
3729 \\\bbl@add\\bbl@savetoday{%
3730 \\\SetString\\today{%
3731 \<\language date>%
3732 {\the\year}{\the\month}{\the\day}}}%
3733 }%
3734 \ifx\bbl@tempb\empty\else
3735 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3736 \fi}%
3737 {}%

```

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.

```

3738 \let\bbl@calendar\@empty
3739 \newcommand\BabelDateSpace{\nobreakspace}
3740 \newcommand\BabelDateDot{. \@} % TODO. \let instead of repeating
3741 \newcommand\BabelDated[1]{\number#1}
3742 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3743 \newcommand\BabelDateM[1]{\number#1}
3744 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3745 \newcommand\BabelDateMMM[1]{%
3746 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3747 \newcommand\BabelDatey[1]{\number#1}%
3748 \newcommand\BabelDateyy[1]{%
3749 \ifnum#1<10 0\number#1 %
3750 \else\ifnum#1<100 \number#1 %
3751 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3752 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3753 \else
3754 \bbl@error
3755 {Currently two-digit years are restricted to the\
3756 range 0-9999.}%
3757 {There is little you can do. Sorry.}%
3758 \fi\fi\fi\fi}}
3759 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3760 \def\bbl@replace@finish@iii#1{%
3761 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3762 \def\bbl@TG@date{%
3763 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3764 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3765 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3766 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3767 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3768 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3769 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3770 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3771 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3772 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%

```

```

3773 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[###1]}%
3774 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[###2]}%
3775 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[###3]}%
3776 % Note after \bbl@replace \toks@ contains the resulting string.
3777 % TODO - Using this implicit behavior doesn't seem a good idea.
3778 \bbl@replace@finish@iii\bbl@toreplace}
3779 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3780 \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.

```

3781 \def\bbl@provide@lsys#1{%
3782 \bbl@ifunset{bbl@lname@#1}%
3783 {\bbl@ini@basic{#1}}%
3784 }%
3785 \bbl@csarg\let{lsys@#1}\@empty
3786 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3787 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3788 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3789 \bbl@ifunset{bbl@lname@#1}{%
3790 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3791 \ifcase\bbl@engine\or\or
3792 \bbl@ifunset{bbl@prehc@#1}{}%
3793 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3794 }%
3795 {\ifx\bbl@xenohyph\@undefined
3796 \let\bbl@xenohyph\bbl@xenohyph@d
3797 \ifx\AtBeginDocument\@notprerr
3798 \expandafter\@secondoftwo % to execute right now
3799 \fi
3800 \AtBeginDocument{%
3801 \expandafter\bbl@add
3802 \csname selectfont \endcsname{\bbl@xenohyph}%
3803 \expandafter\selectlanguage\expandafter{\language}%
3804 \expandafter\bbl@tglobal\csname selectfont \endcsname}%
3805 \fi}}%
3806 \fi
3807 \bbl@csarg\bbl@tglobal{lsys@#1}}
3808 \def\bbl@ifset#1#2#3{% TODO. Move to the correct place.
3809 \bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{#1}}{#3}{#2}}}
3810 \def\bbl@xenohyph@d{%
3811 \bbl@ifset{bbl@prehc@\language}%
3812 {\ifnum\hyphenchar\font=\defaultthyphenchar
3813 \iffontchar\font\bbl@cl{prehc}\relax
3814 \hyphenchar\font\bbl@cl{prehc}\relax
3815 \else\iffontchar\font"200B
3816 \hyphenchar\font"200B
3817 \else
3818 \bbl@warning
3819 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3820 in the current font, and therefore the hyphen\\%
3821 will be printed. Try changing the fontspec's\\%
3822 'HyphenChar' to another value, but be aware\\%
3823 this setting is not safe (see the manual)}%
3824 \hyphenchar\font\defaultthyphenchar
3825 \fi\fi
3826 \fi}%
3827 {\hyphenchar\font\defaultthyphenchar}}
3828 % \fi}

```

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

```

3829 \def\bbl@ini@basic#1{%
3830   \def\BabelBeforeIni##1##2{%
3831     \begingroup
3832     \bbl@add\bbl@secpst@identification{\closein\bbl@readstream }%
3833     \catcode`\[=12 \catcode`\]=12 \catcode`\=12
3834     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14
3835     \bbl@read@ini{##1}%
3836     \endinput          % babel- .tex may contain onlypreamble's
3837     \endgroup}%        boxed, to avoid extra spaces:
3838   {\bbl@input@texini{##1}}}
```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3839 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3840   \ifx\#1%           % \ before, in case #1 is multiletter
3841     \bbl@exp{%
3842       \def\#1\bbl@tempa####1{%
3843         \ifcase>####1\space\the\toks@<else>\@ctrerr<fi>}}%
3844     \else
3845       \toks@>\expandafter{\the\toks@>\or #1}%
3846       \expandafter\bbl@buildifcase
3847     \fi}
```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3848 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@<language>}{#2}}
3849 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3850 \newcommand\localecounter[2]{%
3851   \expandafter\bbl@localecntr
3852   \expandafter{\number\csname c@#2\endcsname}{#1}}
3853 \def\bbl@alphnumeral#1#2{%
3854   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3855 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3856   \ifcase\car#8@nil\or % Currenty <10000, but prepared for bigger
3857     \bbl@alphnumeral@ii{#9}000000#1\or
3858     \bbl@alphnumeral@ii{#9}000000#1#2\or
3859     \bbl@alphnumeral@ii{#9}000000#1#2#3\or
3860     \bbl@alphnumeral@ii{#9}000000#1#2#3#4\else
3861     \bbl@alphnum@invalid{>9999}%
3862   \fi}
3863 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3864   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@<language>}%
3865   {\bbl@cs{cntr@#1.4@<language>}{#5}
3866     \bbl@cs{cntr@#1.3@<language>}{#6}
3867     \bbl@cs{cntr@#1.2@<language>}{#7}
3868     \bbl@cs{cntr@#1.1@<language>}{#8}
3869     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3870     \bbl@ifunset{bbl@cntr@#1.S.321@<language>}{}%
3871     {\bbl@cs{cntr@#1.S.321@<language>}}%
3872   \fi}%

```



```

3873      {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}
3874 \def\bbl@alphnum@invalid#1{%
3875   \bbl@error{Alphabetic numeral too large (#1)}%
3876   {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.

```

3877 \newcommand\localeinfo[1]{%
3878   \bbl@ifunset{\bbl@csname \bbl@info@#1\endcsname @\language}%
3879   {\bbl@error{I've found no info for the current locale.\%
3880     The corresponding ini file has not been loaded\%
3881     Perhaps it doesn't exist}%
3882     {See the manual for details.}}%
3883   {\bbl@cs{csname \bbl@info@#1\endcsname @\language}}}%
3884 % \@namedef{\bbl@info@name.locale}{lname}
3885 \@namedef{\bbl@info@tag.ini}{lini}
3886 \@namedef{\bbl@info@name.english}{elname}
3887 \@namedef{\bbl@info@name.opentype}{lname}
3888 \@namedef{\bbl@info@tag.bcp47}{lbcpr} % TODO
3889 \@namedef{\bbl@info@tag.opentype}{lotf}
3890 \@namedef{\bbl@info@script.name}{esname}
3891 \@namedef{\bbl@info@script.name.opentype}{sname}
3892 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
3893 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3894 \let\bbl@ensureinfo\@gobble
3895 \newcommand\BabelEnsureInfo{%
3896   \ifx\InputIfFileExists\undefined\else
3897     \def\bbl@ensureinfo##1{%
3898       \bbl@ifunset{\bbl@lname@##1}{\bbl@ini@basic{##1}}}%
3899   \fi
3900   \bbl@foreach\bbl@loaded{%
3901     \def\language{##1}%
3902     \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`.

```

3903 \newcommand\getlocaleproperty{%
3904   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3905 \def\bbl@getproperty@s#1#2#3{%
3906   \let#1\relax
3907   \def\bbl@elt##1##2##3{%
3908     \bbl@ifsamestring{##1/##2}{##3}%
3909     {\providecommand#1{##3}%
3910     \def\bbl@elt####1####2####3{}}}%
3911   {}}%
3912   \bbl@cs{inidata@#2}}%
3913 \def\bbl@getproperty@x#1#2#3{%
3914   \bbl@getproperty@s{#1}{#2}{#3}%
3915   \ifx#1\relax
3916     \bbl@error
3917     {Unknown key for locale '#2':\%
3918     #3\%
3919     \string#1 will be set to \relax}%
3920     {Perhaps you misspelled it.}%
3921   \fi}
3922 \let\bbl@ini@loaded\@empty
3923 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

10 Adjusting the Babel bahavior

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

```
3924 \newcommand\babeladjust[1]{% TODO. Error handling.
3925   \bbl@forkv{#1}{%
3926     \bbl@ifunset{bbl@ADJ@##1@##2}%
3927     {\bbl@cs{ADJ@##1}{##2}}%
3928     {\bbl@cs{ADJ@##1@##2}}}
3929 %
3930 \def\bbl@adjust@lua#1#2{%
3931   \ifvmode
3932     \ifnum\currentgrouplevel=\z@
3933       \directlua{ Babel.#2 }%
3934       \expandafter\expandafter\expandafter\@gobble
3935     \fi
3936   \fi
3937   {\bbl@error   % The error is gobbled if everything went ok.
3938     {Currently, #1 related features can be adjusted only\\%
3939       in the main vertical list.}%
3940     {Maybe things change in the future, but this is what it is.}}}
3941 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3942   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3943 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3944   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3945 \@namedef{bbl@ADJ@bidi.text@on}{%
3946   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3947 \@namedef{bbl@ADJ@bidi.text@off}{%
3948   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3949 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3950   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3951 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3952   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3953 %
3954 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3955   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3956 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3957   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3958 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3959   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3960 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3961   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3962 %
3963 \def\bbl@adjust@layout#1{%
3964   \ifvmode
3965     #1%
3966     \expandafter\@gobble
3967   \fi
3968   {\bbl@error   % The error is gobbled if everything went ok.
3969     {Currently, layout related features can be adjusted only\\%
3970       in vertical mode.}%
3971     {Maybe things change in the future, but this is what it is.}}}
3972 \@namedef{bbl@ADJ@layout.tabular@on}{%
3973   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3974 \@namedef{bbl@ADJ@layout.tabular@off}{%
3975   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3976 \@namedef{bbl@ADJ@layout.lists@on}{%
3977   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3978 \@namedef{bbl@ADJ@layout.lists@on}{%
```

```

3979 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3980 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3981 \bbl@activateposthyphen}
3982 %
3983 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3984 \bbl@bcpallowedtrue}
3985 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3986 \bbl@bcpallowedfalse}
3987 \@namedef{bbl@ADJ@autoload.bcp47.prefix}{#1}%
3988 \def\bbl@bcp@prefix{#1}}
3989 \def\bbl@bcp@prefix{bcp47-}
3990 \@namedef{bbl@ADJ@autoload.options}{#1}%
3991 \def\bbl@autoload@options{#1}}
3992 \let\bbl@autoload@bcptoptions\@empty
3993 \@namedef{bbl@ADJ@autoload.bcp47.options}{#1}%
3994 \def\bbl@autoload@bcptoptions{#1}}
3995 \newif\ifbbl@bcptoname
3996 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3997 \bbl@bcptonametrue
3998 \BabelEnsureInfo}
3999 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4000 \bbl@bcptonamefalse}
4001 % TODO: use babel name, override
4002 %
4003 % As the final task, load the code for lua.
4004 %
4005 \ifx\directlua\@undefined\else
4006 \ifx\bbl@luapatterns\@undefined
4007 \input luababel.def
4008 \fi
4009 \fi
4010 </core>

A proxy file for switch.def

4011 <*kernel>
4012 \let\bbl@onlyswitch\@empty
4013 \input babel.def
4014 \let\bbl@onlyswitch\@undefined
4015 </kernel>
4016 <*patterns>

```

11 Loading hyphenation patterns

The following code is meant to be read by \LaTeX because it should instruct \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 \LaTeX 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.

```

4017 <<Make sure ProvidesFile is defined>>
4018 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
4019 \xdef\bbl@format{\jobname}

```

```

4020 \def\bbl@version{<<version>>}
4021 \def\bbl@date{<<date>>}
4022 \ifx\AtBeginDocument\@undefined
4023   \def\@empty{}
4024   \let\orig@dump\dump
4025   \def\dump{%
4026     \ifx\@ztryfc\@undefined
4027     \else
4028       \toks0=\expandafter{\@preamblecmds}%
4029       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4030       \def\@begindocumenthook{}%
4031     \fi
4032     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4033 \fi
4034 <<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.

```

4035 \def\process@line#1#2 #3 #4 {%
4036   \ifx=#1%
4037     \process@synonym{#2}%
4038   \else
4039     \process@language{#1#2}{#3}{#4}%
4040   \fi
4041   \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.

```

4042 \toks@{}
4043 \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.

```

4044 \def\process@synonym#1{%
4045   \ifnum\last@language=\m@ne
4046     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4047   \else
4048     \expandafter\chardef\csname l@#1\endcsname\last@language
4049     \wlog{\string\l@#1=\string\language\the\last@language}%
4050     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4051       \csname\language\hyphenmins\endcsname
4052     \let\bbl@elt\relax
4053     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4054   \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 `\langle lang \rangle hyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. 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.

```

4055 \def\process@language#1#2#3{%
4056   \expandafter\addlanguage\csname l@#1\endcsname
4057   \expandafter\language\csname l@#1\endcsname
4058   \edef\language#1#2#3%
4059   \bbl@hook@everylanguage{#1}%
4060   % > luatex
4061   \bbl@get@enc#1::@@@
4062   \begingroup
4063     \lefthyphenmin@m@ne
4064     \bbl@hook@loadpatterns{#2}%
4065     % > luatex
4066     \ifnum\lefthyphenmin=\m@ne
4067       \else
4068         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4069           \the\lefthyphenmin\the\righthyphenmin}%
4070       \fi
4071   \endgroup
4072   \def\bbl@tempa{#3}%
4073   \ifx\bbl@tempa\@empty\else
4074     \bbl@hook@loadexceptions{#3}%
4075     % > luatex
4076   \fi
4077   \let\bbl@elt\relax
4078   \edef\bbl@languages{%
4079     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4080   \ifnum\the\language=\z@
4081     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4082       \set@hyphenmins\tw@\thr@@\relax
4083     \else
4084       \expandafter\expandafter\expandafter\set@hyphenmins
4085       \csname #1hyphenmins\endcsname
4086     \fi

```

```

4087 \the\toks@
4088 \toks@{}%
4089 \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.

```

4090 \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.

```

4091 \def\bbl@hook@everylanguage#1{}
4092 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4093 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4094 \def\bbl@hook@loadkernel#1{%
4095   \def\addlanguage{\csname newlanguage\endcsname}%
4096   \def\adddialect##1##2{%
4097     \global\chardef##1##2\relax
4098     \wlog{\string##1 = a dialect from \string\language##2}}%
4099   \def\iflanguage##1{%
4100     \expandafter\ifx\csname l@##1\endcsname\relax
4101       \nolannerr{##1}%
4102     \else
4103       \ifnum\csname l@##1\endcsname=\language
4104         \expandafter\expandafter\expandafter\@firstoftwo
4105       \else
4106         \expandafter\expandafter\expandafter\@secondoftwo
4107       \fi
4108     \fi}%
4109   \def\providehyphenmins##1##2{%
4110     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4111       \@namedef{##1hyphenmins}{##2}%
4112     \fi}%
4113   \def\set@hyphenmins##1##2{%
4114     \lefthyphenmin##1\relax
4115     \righthyphenmin##2\relax}%
4116   \def\selectlanguage{%
4117     \errhelp{Selecting a language requires a package supporting it}%
4118     \errmessage{Not loaded}}%
4119   \let\foreignlanguage\selectlanguage
4120   \let\otherlanguage\selectlanguage
4121   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4122   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4123     \def\setlocale{%
4124       \errhelp{Find an armchair, sit down and wait}%
4125       \errmessage{Not yet available}}%
4126     \let\uselocale\setlocale
4127     \let\locale\setlocale
4128     \let\selectlocale\setlocale
4129     \let\localename\setlocale
4130     \let\textlocale\setlocale
4131     \let\textlanguage\setlocale
4132     \let\languagetext\setlocale}
4133   \begingroup
4134     \def\AddBabelHook#1#2{%
4135       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4136         \def\next{\toks1}%
4137       \else

```

```

4138     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4139     \fi
4140     \next}
4141     \ifx\directlua\@undefined
4142     \ifx\XeTeXinputencoding\@undefined\else
4143     \input xebabel.def
4144     \fi
4145     \else
4146     \input luababel.def
4147     \fi
4148     \openin1 = babel-\bbl@format.cfg
4149     \ifeof1
4150     \else
4151     \input babel-\bbl@format.cfg\relax
4152     \fi
4153     \closein1
4154 \endgroup
4155 \bbl@hook@loadkernel{switch.def}

```

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

```

4156 \openin1 = language.dat

```

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

```

4157 \def\language{english}%
4158 \ifeof1
4159 \message{I couldn't find the file language.dat,\space
4160         I will try the file hyphen.tex}
4161 \input hyphen.tex\relax
4162 \chardef\l@english\z@
4163 \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.

```

4164 \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.

```

4165 \loop
4166 \endlinechar\m@ne
4167 \read1 to \bbl@line
4168 \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.

```

4169 \if T\ifeof1F\fi T\relax
4170 \ifx\bbl@line\@empty\else
4171     \edef\bbl@line{\bbl@line\space\space\space}%
4172     \expandafter\process@line\bbl@line\relax
4173 \fi
4174 \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.

```

4175 \begingroup
4176   \def\bbl@elt#1#2#3#4{%
4177     \global\language=#2\relax
4178     \gdef\language#1}%
4179   \def\bbl@elt##1##2##3##4{}}%
4180   \bbl@languages
4181 \endgroup
4182 \fi
4183 \closein1

```

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

```

4184 \if/\the\toks@/\else
4185   \errhelp{language.dat loads no language, only synonyms}
4186   \errmessage{Orphan language synonym}
4187 \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.

```

4188 \let\bbl@line\@undefined
4189 \let\process@line\@undefined
4190 \let\process@synonym\@undefined
4191 \let\process@language\@undefined
4192 \let\bbl@get@enc\@undefined
4193 \let\bbl@hyph@enc\@undefined
4194 \let\bbl@tempa\@undefined
4195 \let\bbl@hook@loadkernel\@undefined
4196 \let\bbl@hook@everylanguage\@undefined
4197 \let\bbl@hook@loadpatterns\@undefined
4198 \let\bbl@hook@loadexceptions\@undefined
4199 \</patterns>

```

Here the code for `iniTeX` 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].

```

4200 <<{*More package options}>> ≡
4201 \chardef\bbl@bidimode\z@
4202 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4203 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4204 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4205 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4206 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4207 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4208 <</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.


```

4209 <<*Font selection>> ≡
4210 \bbl@trace{Font handling with fontspec}
4211 \ifx\ExplSyntaxOn\@undefined\else
4212   \ExplSyntaxOn
4213   \catcode`\ =10
4214   \def\bbl@loadfontspec{%
4215     \usepackage{fontspec}%
4216     \expandafter
4217     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4218       Font '\l_fontspec_fontname_tl' is using the\\%
4219       default features for language '##1'.\\%
4220       That's usually fine, because many languages\\%
4221       require no specific features, but if the output is\\%
4222       not as expected, consider selecting another font.}
4223     \expandafter
4224     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4225       Font '\l_fontspec_fontname_tl' is using the\\%
4226       default features for script '##2'.\\%
4227       That's not always wrong, but if the output is\\%
4228       not as expected, consider selecting another font.}}
4229   \ExplSyntaxOff
4230 \fi
4231 \@onlypreamble\babelfont
4232 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4233   \bbl@foreach{#1}{%
4234     \expandafter\ifx\csname date##1\endcsname\relax
4235     \IfFileExists{babel-##1.tex}%
4236       {\babelprovide{##1}}%
4237     }%
4238   \fi}%
4239 \edef\bbl@tempa{#1}%
4240 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4241 \ifx\fontspec\@undefined
4242   \bbl@loadfontspec
4243 \fi
4244 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4245 \bbl@bblfont}
4246 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4247   \bbl@ifunset{\bbl@tempb family}%
4248   {\bbl@providedefam{\bbl@tempb}}%
4249   {\bbl@exp{%
4250     \\\bbl@sreplace\<\bbl@tempb family >%
4251     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4252   % For the default font, just in case:
4253   \bbl@ifunset{\bbl@lsys\@languagename}{\bbl@provide@lsys{\@languagename}}}%
4254   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4255   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4256     \bbl@exp{%
4257       \let\<\bbl@tempb dflt@\@languagename>\<\bbl@tempb dflt@>%
4258       \\\bbl@font@set\<\bbl@tempb dflt@\@languagename>%
4259       \<\bbl@tempb default>\<\bbl@tempb family>}}}%
4260   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4261     \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:

```

4262 \def\bbl@providedefam#1{%
4263   \bbl@exp{%
4264     \\\newcommand\<#1default>{}% Just define it
4265     \\\bbl@add@list\<\bbl@font@fams{#1}%

```

```

4266   \\\DeclareRobustCommand\<#1family>{%
4267     \\\not@math@alphabet\<#1family>\relax
4268     \\\fontfamily\<#1default>\\selectfont}%
4269   \\\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.

```

4270 \def\bbbl@nostdfont#1{%
4271   \bbbl@ifunset{bbbl@WFF@f@family}%
4272   {\bbbl@csarg\gdef{WFF@f@family}}}% Flag, to avoid dupl warns
4273   \bbbl@infowarn{The current font is not a babel standard family:\\%
4274     #1%
4275     \fontname\font\\%
4276     There is nothing intrinsically wrong with this warning, and\\%
4277     you can ignore it altogether if you do not need these\\%
4278     families. But if they are used in the document, you should be\\%
4279     aware 'babel' will no set Script and Language for them, so\\%
4280     you may consider defining a new family with \string\babelfont.\\%
4281     See the manual for further details about \string\babelfont.\\%
4282     Reported}}
4283   {}}%
4284 \gdef\bbbl@switchfont{%
4285   \bbbl@ifunset{bbbl@lsys@\language name}{\bbbl@provide@lsys{\language name}}}%
4286   \bbbl@exp{% eg Arabic -> arabic
4287     \lowercase{\edef\\bbbl@tempa{\bbbl@cl{sname}}}}}%
4288   \bbbl@foreach\bbbl@font@fams{%
4289     \bbbl@ifunset{bbbl@##1dflt@\language name}% (1) language?
4290     {\bbbl@ifunset{bbbl@##1dflt*@\bbbl@tempa}% (2) from script?
4291       {\bbbl@ifunset{bbbl@##1dflt@}% 2=F - (3) from generic?
4292         {}}% 123=F - nothing!
4293         {\bbbl@exp{% 3=T - from generic
4294           \global\let\<bbbl@##1dflt@\language name>%
4295             \<bbbl@##1dflt@>}}}%
4296         {\bbbl@exp{% 2=T - from script
4297           \global\let\<bbbl@##1dflt@\language name>%
4298             \<bbbl@##1dflt*@\bbbl@tempa>}}}%
4299         {}}% 1=T - language, already defined
4300   \def\bbbl@tempa{\bbbl@nostdfont}}}%
4301   \bbbl@foreach\bbbl@font@fams{% don't gather with prev for
4302     \bbbl@ifunset{bbbl@##1dflt@\language name}%
4303     {\bbbl@cs{famrst@##1}%
4304       \global\bbbl@csarg\let{famrst@##1}\relax}%
4305     {\bbbl@exp{% order is relevant
4306       \\\bbbl@add\\originalTeX{%
4307         \\\bbbl@font@rst{\bbbl@cl{##1dflt}}}%
4308         \<##1default>\<##1family>{##1}}}%
4309       \\\bbbl@font@set\<bbbl@##1dflt@\language name>% the main part!
4310       \<##1default>\<##1family>}}}%
4311   \bbbl@ifrestoring{{\bbbl@tempa}}}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

4312 \ifx\fontfamily\undefined\else % if latex
4313   \ifcase\bbbl@engine % if pdftex
4314     \let\bbbl@ckeckstdfonts\relax
4315   \else
4316     \def\bbbl@ckeckstdfonts{%
4317       \begingroup
4318       \global\let\bbbl@ckeckstdfonts\relax

```

```

4319 \let\bbl@tempa\@empty
4320 \bbl@foreach\bbl@font@fams{%
4321 \bbl@ifunset\bbl@##1dflt@}%
4322 {\nameuse{##1family}%
4323 \bbl@csarg\gdef{WFF@f@family}{}}% Flag
4324 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= f@family\\%
4325 \space\space\fontname\font\\}%
4326 \bbl@csarg\xdef{##1dflt@}{f@family}%
4327 \expandafter\xdef\csname ##1default\endcsname{f@family}}%
4328 {}}%
4329 \ifx\bbl@tempa\@empty\else
4330 \bbl@infowarn{The following font families will use the default\\%
4331 settings for all or some languages:\\%
4332 \bbl@tempa
4333 There is nothing intrinsically wrong with it, but\\%
4334 'babel' will no set Script and Language, which could\\%
4335 be relevant in some languages. If your document uses\\%
4336 these families, consider redefining them with \string\babelfont.\\%
4337 Reported}%
4338 \fi
4339 \endgroup}
4340 \fi
4341 \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.

```

4342 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4343 \bbl@xin@{<>}{#1}%
4344 \ifin@
4345 \bbl@exp{\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4346 \fi
4347 \bbl@exp{%
4348 \def\\#2#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4349 \\bbl@ifsamestring{#2}{f@family}{\\#3\let\\bbl@tempa\relax}}%
4350 % TODO - next should be global?, but even local does its job. I'm
4351 % still not sure -- must investigate:
4352 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4353 \let\bbl@tempe\bbl@mapselect
4354 \let\bbl@mapselect\relax
4355 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4356 \let#4\@empty % Make sure \renewfontfamily is valid
4357 \bbl@exp{%
4358 \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4359 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4360 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4361 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4362 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4363 \\renewfontfamily\\#4%
4364 [\bbl@cs{lsys@languagename},#2]}{#3}% ie \bbl@exp{.}{#3}
4365 \begingroup
4366 #4%
4367 \xdef#1{f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4368 \endgroup
4369 \let#4\bbl@temp@fam
4370 \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4371 \let\bbl@mapselect\bbl@tempe}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```
4372 \def\bbl@font@rst#1#2#3#4{%
4373   \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.

```
4374 \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 :-).

```
4375 \newcommand\babelFSstore[2][]{%
4376   \bbl@ifblank{#1}%
4377   {\bbl@csarg\def{sname@#2}{Latin}}%
4378   {\bbl@csarg\def{sname@#2}{#1}}%
4379   \bbl@provide@dirs{#2}%
4380   \bbl@csarg\ifnum{wdir@#2}>\z@
4381     \let\bbl@beforeforeign\leavevmode
4382     \EnableBabelHook{babel-bidi}%
4383   \fi
4384   \bbl@foreach{#2}{%
4385     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4386     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4387     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4388 \def\bbl@FSstore#1#2#3#4{%
4389   \bbl@csarg\edef{#2default#1}{#3}%
4390   \expandafter\addto\csname extras#1\endcsname{%
4391     \let#4#3%
4392     \ifx#3\f@family
4393       \edef#3{\csname bbl@#2default#1\endcsname}%
4394       \fontfamily{#3}\selectfont
4395     \else
4396       \edef#3{\csname bbl@#2default#1\endcsname}%
4397     \fi}%
4398   \expandafter\addto\csname noextras#1\endcsname{%
4399     \ifx#3\f@family
4400       \fontfamily{#4}\selectfont
4401     \fi
4402     \let#3#4}}
4403 \let\bbl@langfeatures\@empty
4404 \def\babelFSfeatures{% make sure \fontspec is redefined once
4405   \let\bbl@ori@fontspec\fontspec
4406   \renewcommand\fontspec[1][]{%
4407     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4408   \let\babelFSfeatures\bbl@FSfeatures
4409   \babelFSfeatures}
4410 \def\bbl@FSfeatures#1#2{%
4411   \expandafter\addto\csname extras#1\endcsname{%
4412     \babel@save\bbl@langfeatures
4413     \edef\bbl@langfeatures{#2,}}
4414 <</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.

```
4415 <<{*Footnote changes}>> ≡
4416 \bbl@trace{Bidi footnotes}
4417 \ifnum\bbl@bidimode>\z@
4418   \def\bbl@footnote#1#2#3{%
4419     \ifnextchar[%
4420       {\bbl@footnote@o{#1}{#2}{#3}}%
4421       {\bbl@footnote@x{#1}{#2}{#3}}}
4422   \def\bbl@footnote@x#1#2#3#4{%
4423     \bgroup
4424     \select@language@x{\bbl@main@language}%
4425     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4426     \egroup}
4427   \def\bbl@footnote@o#1#2#3[#4]#5{%
4428     \bgroup
4429     \select@language@x{\bbl@main@language}%
4430     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4431     \egroup}
4432   \def\bbl@footnotetext#1#2#3{%
4433     \ifnextchar[%
4434       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4435       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4436   \def\bbl@footnotetext@x#1#2#3#4{%
4437     \bgroup
4438     \select@language@x{\bbl@main@language}%
4439     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4440     \egroup}
4441   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4442     \bgroup
4443     \select@language@x{\bbl@main@language}%
4444     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4445     \egroup}
4446   \def\BabelFootnote#1#2#3#4{%
4447     \ifx\bbl@fn@footnote\@undefined
4448       \let\bbl@fn@footnote\footnote
4449     \fi
4450     \ifx\bbl@fn@footnotetext\@undefined
4451       \let\bbl@fn@footnotetext\footnotetext
4452     \fi
4453     \bbl@ifblank{#2}%
4454     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4455      \@namedef{\bbl@stripslash#1text}%
4456      {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4457     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4458      \@namedef{\bbl@stripslash#1text}%
4459      {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4460   \fi
4461 <</Footnote changes>>
```

Now, the code.

```
4462 <{*xetex}>
4463 \def\BabelStringsDefault{unicode}
4464 \let\xebbl@stop\relax
```

```

4465 \AddBabelHook{xetex}{encodedcommands}{%
4466   \def\bbl@tempa{#1}%
4467   \ifx\bbl@tempa@empty
4468     \XeTeXinputencoding"bytes"%
4469   \else
4470     \XeTeXinputencoding"#1"%
4471   \fi
4472   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4473 \AddBabelHook{xetex}{stopcommands}{%
4474   \xebbl@stop
4475   \let\xebbl@stop\relax}
4476 \def\bbl@intraspace#1 #2 #3@@{%
4477   \bbl@csarg\gdef{xeisp@\language}%
4478   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4479 \def\bbl@intrapenalty#1@@{%
4480   \bbl@csarg\gdef{xeipn@\language}%
4481   {\XeTeXlinebreakpenalty #1\relax}}
4482 \def\bbl@provide@intraspace{%
4483   \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4484   \ifin@else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4485   \ifin@
4486     \bbl@ifunset{\bbl@intsp@\language}{}%
4487     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4488       \ifx\bbl@KVP@intraspace\@nil
4489         \bbl@exp{%
4490           \\bbl@intraspace\bbl@cl{intsp}\\\\@}%
4491         \fi
4492         \ifx\bbl@KVP@intrapenalty\@nil
4493           \bbl@intrapenalty0@@
4494         \fi
4495       \fi
4496       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4497         \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4498       \fi
4499       \ifx\bbl@KVP@intrapenalty\@nil\else
4500         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4501       \fi
4502       \bbl@exp{%
4503         \\bbl@add\<extras\language>{%
4504           \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
4505           \<bbl@xeisp@\language>%
4506           \<bbl@xeipn@\language>%
4507           \\bbl@toglobal\<extras\language>%
4508           \\bbl@add\<noextras\language>{%
4509             \XeTeXlinebreaklocale "en"%
4510             \\bbl@toglobal\<noextras\language>}}%
4511       \ifx\bbl@ispace\@undefined
4512         \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4513       \ifx\AtBeginDocument\@notprerr
4514         \expandafter\@secondoftwo % to execute right now
4515       \fi
4516       \AtBeginDocument{%
4517         \expandafter\bbl@add
4518         \csname selectfont \endcsname{\bbl@ispace}%
4519         \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4520       \fi}%
4521   \fi}
4522 \ifx\DisableBabelHook\@undefined\endinput\fi
4523 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}

```

```

4524 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4525 \DisableBabelHook{babel-fontspec}
4526 <<Font selection>>
4527 \input txtbabel.def
4528 </xetex>

```

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titles, and geometry.

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

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

```

4529 (*texxet)
4530 \providecommand\bbl@provide@intraspace{}
4531 \bbl@trace{Redefinitions for bidi layout}
4532 \def\bbl@sspre@caption{%
4533   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4534 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4535 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4536 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4537 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4538   \def\hangfrom#1{%
4539     \setbox\@tempboxa\hbox{#1}%
4540     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4541     \noindent\box\@tempboxa}
4542 \def\raggedright{%
4543   \let\@centercr
4544   \bbl@startskip\z@skip
4545   \@rightskip\@flushglue
4546   \bbl@endskip\@rightskip
4547   \parindent\z@
4548   \parfillskip\bbl@startskip}
4549 \def\raggedleft{%
4550   \let\@centercr
4551   \bbl@startskip\@flushglue
4552   \bbl@endskip\z@skip
4553   \parindent\z@
4554   \parfillskip\bbl@endskip}
4555 \fi
4556 \IfBabelLayout{lists}
4557   {\bbl@sreplace\list
4558     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4559     \def\bbl@listleftmargin{%
4560       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4561     \ifcase\bbl@engine
4562       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4563       \def\p@enumiii{\p@enumii}\theenumii{}
4564     \fi
4565     \bbl@sreplace\@verbatim
4566       {\leftskip\@totalleftmargin}%
4567       {\bbl@startskip\textwidth
4568         \advance\bbl@startskip-\linewidth}%
4569     \bbl@sreplace\@verbatim

```

```

4570      {\rightskip\z@skip}%
4571      {\bbl@endskip\z@skip}}%
4572  {}
4573 \IfBabelLayout{contents}
4574  {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4575   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4576  {}
4577 \IfBabelLayout{columns}
4578  {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4579   \def\bbl@outputbox#1{%
4580     \hb@xt@\textwidth{%
4581       \hskip\columnwidth
4582       \hfil
4583       {\normalcolor\vrule \@width\columnseprule}%
4584       \hfil
4585       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4586       \hskip-\textwidth
4587       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4588       \hskip\columnsep
4589       \hskip\columnwidth}}}%
4590  {}
4591 <<Footnote changes>>
4592 \IfBabelLayout{footnotes}%
4593  {\BabelFootnote\footnote\language{}{}}%
4594   \BabelFootnote\localfootnote\language{}{}}%
4595   \BabelFootnote\mainfootnote{}{}{}}
4596  {}

```

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

```

4597 \IfBabelLayout{counters}%
4598  {\let\bbl@latinarabic=\@arabic
4599   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4600   \let\bbl@asciroman=\@roman
4601   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4602   \let\bbl@asciiRoman=\@Roman
4603   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
4604 </texet>

```

13.3 LuaTeX

The loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

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

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

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

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

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4605 (*luatex)
4606 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4607 \bbl@trace{Read language.dat}
4608 \ifx\bbl@readstream\undefined
4609   \csname newread\endcsname\bbl@readstream
4610 \fi
4611 \begingroup
4612   \toks@{}
4613   \count@ \z@ % 0=start, 1=0th, 2=normal
4614   \def\bbl@process@line#1#2 #3 #4 {%
4615     \ifx=#1%
4616       \bbl@process@synonym{#2}%
4617     \else
4618       \bbl@process@language{#1#2}{#3}{#4}%
4619     \fi
4620     \ignorespaces}
4621   \def\bbl@manylang{%
4622     \ifnum\bbl@last>\@ne
4623       \bbl@info{Non-standard hyphenation setup}%
4624     \fi
4625     \let\bbl@manylang\relax}
4626   \def\bbl@process@language#1#2#3{%
4627     \ifcase\count@
4628       \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4629     \or
4630       \count@\tw@
4631     \fi
4632     \ifnum\count@=\tw@
4633       \expandafter\addlanguage\csname l@#1\endcsname
4634       \language\allocationnumber
4635       \chardef\bbl@last\allocationnumber
4636       \bbl@manylang
4637       \let\bbl@elt\relax
4638       \xdef\bbl@languages{%
4639         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4640     \fi
4641     \the\toks@
4642     \toks@{}

```

```

4643 \def\bbl@process@synonym@aux#1#2{%
4644   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4645   \let\bbl@elt\relax
4646   \xdef\bbl@languages{%
4647     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4648 \def\bbl@process@synonym#1{%
4649   \ifcase\count@
4650     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4651   \or
4652     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4653   \else
4654     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4655   \fi}
4656 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4657   \chardef\l@english\z@
4658   \chardef\l@USenglish\z@
4659   \chardef\bbl@last\z@
4660   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4661   \gdef\bbl@languages{%
4662     \bbl@elt{english}{0}{hyphen.tex}}%
4663     \bbl@elt{USenglish}{0}{}}
4664 \else
4665   \global\let\bbl@languages@format\bbl@languages
4666   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4667     \ifnum#2>\z@\else
4668       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4669     \fi}%
4670   \xdef\bbl@languages{\bbl@languages}%
4671 \fi
4672 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4673 \bbl@languages
4674 \openin\bbl@readstream=language.dat
4675 \ifeof\bbl@readstream
4676   \bbl@warning{I couldn't find language.dat. No additional\\%
4677     patterns loaded. Reported}%
4678 \else
4679   \loop
4680     \endlinechar\m@ne
4681     \read\bbl@readstream to \bbl@line
4682     \endlinechar\^^M
4683     \if T\ifeof\bbl@readstream F\fi T\relax
4684     \ifx\bbl@line\@empty\else
4685       \edef\bbl@line{\bbl@line\space\space\space}%
4686       \expandafter\bbl@process@line\bbl@line\relax
4687     \fi
4688   \repeat
4689 \fi
4690 \endgroup
4691 \bbl@trace{Macros for reading patterns files}
4692 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4693 \ifx\babelcatcodetablenum\@undefined
4694   \ifx\newcatcodetable\@undefined
4695     \def\babelcatcodetablenum{5211}
4696     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4697   \else
4698     \newcatcodetable\babelcatcodetablenum
4699     \newcatcodetable\bbl@pattcodes
4700 \fi
4701 \else

```

```

4702 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4703 \fi
4704 \def\bbl@luapatterns#1#2{%
4705   \bbl@get@enc#1::\@@@
4706   \setbox\z@\hbox\bgroup
4707     \begingroup
4708       \savecatcodetable\babelcatcodetablenum\relax
4709       \initcatcodetable\bbl@pattcodes\relax
4710       \catcodetable\bbl@pattcodes\relax
4711       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4712       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4713       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4714       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4715       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4716       \catcode`\'=12 \catcode`\'=12 \catcode`\`=12
4717       \input #1\relax
4718       \catcodetable\babelcatcodetablenum\relax
4719     \endgroup
4720   \def\bbl@tempa{#2}%
4721   \ifx\bbl@tempa@empty\else
4722     \input #2\relax
4723   \fi
4724 \egroup}%
4725 \def\bbl@patterns@lua#1{%
4726   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4727     \csname l@#1\endcsname
4728     \edef\bbl@tempa{#1}%
4729   \else
4730     \csname l@#1:\f@encoding\endcsname
4731     \edef\bbl@tempa{#1:\f@encoding}%
4732   \fi\relax
4733   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4734   \@ifundefined{bbl@hyphendata@the\language}%
4735     {\def\bbl@elt##1##2##3##4{%
4736       \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4737       \def\bbl@tempb{##3}%
4738       \ifx\bbl@tempb@empty\else % if not a synonymous
4739         \def\bbl@tempc{##3}{##4}}%
4740       \fi
4741       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4742     \fi}%
4743   \bbl@languages
4744   \@ifundefined{bbl@hyphendata@the\language}%
4745     {\bbl@info{No hyphenation patterns were set for\%
4746       language '\bbl@tempa'. Reported}}%
4747     {\expandafter\expandafter\expandafter\bbl@luapatterns
4748       \csname bbl@hyphendata@the\language\endcsname}}}%
4749 \endinput\fi
4750 % Here ends \ifx\AddBabelHook\undefined
4751 % A few lines are only read by hyphen.cfg
4752 \ifx\DisableBabelHook\undefined
4753   \AddBabelHook{luatex}{everylanguage}{%
4754     \def\process@language##1##2##3{%
4755       \def\process@line####1####2 ####3 ####4 {}}}
4756   \AddBabelHook{luatex}{loadpatterns}{%
4757     \input #1\relax
4758     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4759       {#1}{}}}
4760   \AddBabelHook{luatex}{loadexceptions}{%

```

```

4761 \input #1\relax
4762 \def\bbl@tempb##1##2{{##1}{##2}}%
4763 \expandafter\edef\csname bbl@hyphendata@the\language\endcsname
4764 {\expandafter\expandafter\expandafter\bbl@tempb
4765 \csname bbl@hyphendata@the\language\endcsname}}
4766 \endinput\fi
4767 % Here stops reading code for hyphen.cfg
4768 % The following is read the 2nd time it's loaded
4769 \begingroup
4770 \catcode`\%=12
4771 \catcode`\'=12
4772 \catcode`\%=12
4773 \catcode`\:=12
4774 \directlua{
4775   Babel = Babel or {}
4776   function Babel.bytes(line)
4777     return line:gsub(".",
4778       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4779   end
4780   function Babel.begin_process_input()
4781     if luatexbase and luatexbase.add_to_callback then
4782       luatexbase.add_to_callback('process_input_buffer',
4783         Babel.bytes, 'Babel.bytes')
4784     else
4785       Babel.callback = callback.find('process_input_buffer')
4786       callback.register('process_input_buffer', Babel.bytes)
4787     end
4788   end
4789   function Babel.end_process_input ()
4790     if luatexbase and luatexbase.remove_from_callback then
4791       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4792     else
4793       callback.register('process_input_buffer', Babel.callback)
4794     end
4795   end
4796   function Babel.addpatterns(pp, lg)
4797     local lg = lang.new(lg)
4798     local pats = lang.patterns(lg) or ''
4799     lang.clear_patterns(lg)
4800     for p in pp:gmatch('[^%s]+') do
4801       ss = ''
4802       for i in string.utfcharacters(p:gsub('%d', '')) do
4803         ss = ss .. '%d?' .. i
4804       end
4805       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4806       ss = ss:gsub('%.%%d%?$', '%%.')
4807       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4808       if n == 0 then
4809         tex.sprint(
4810           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4811           .. p .. [[]])
4812         pats = pats .. ' ' .. p
4813       else
4814         tex.sprint(
4815           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4816           .. p .. [[]])
4817       end
4818     end
4819     lang.patterns(lg, pats)

```

```

4820 end
4821 }
4822 \endgroup
4823 \ifx\newattribute\undefined\else
4824 \newattribute\bbbl@attr@locale
4825 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbbl@attr@locale'}
4826 \AddBabelHook{luatex}{beforeextras}{%
4827 \setattribute\bbbl@attr@locale\localeid}
4828 \fi
4829 \def\BabelStringsDefault{unicode}
4830 \let\luabbbl@stop\relax
4831 \AddBabelHook{luatex}{encodedcommands}{%
4832 \def\bbbl@tempa{utf8}\def\bbbl@tempb{#1}%
4833 \ifx\bbbl@tempa\bbbl@tempb\else
4834 \directlua{Babel.begin_process_input()}%
4835 \def\luabbbl@stop{%
4836 \directlua{Babel.end_process_input()}}%
4837 \fi}%
4838 \AddBabelHook{luatex}{stopcommands}{%
4839 \luabbbl@stop
4840 \let\luabbbl@stop\relax}
4841 \AddBabelHook{luatex}{patterns}{%
4842 \@ifundefined{bbbl@hyphendata@the\language}%
4843 {\def\bbbl@elt##1##2##3##4{%
4844 \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4845 \def\bbbl@tempb{##3}%
4846 \ifx\bbbl@tempb\@empty\else % if not a synonymous
4847 \def\bbbl@tempc{##3}{##4}%
4848 \fi
4849 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
4850 \fi}%
4851 \bbbl@languages
4852 \@ifundefined{bbbl@hyphendata@the\language}%
4853 {\bbbl@info{No hyphenation patterns were set for\%
4854 language '#2'. Reported}}%
4855 {\expandafter\expandafter\expandafter\bbbl@luapatterns
4856 \csname bbl@hyphendata@the\language\endcsname}}}%
4857 \@ifundefined{bbbl@patterns@}{}%
4858 \begingroup
4859 \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
4860 \ifin\else
4861 \ifx\bbbl@patterns@\@empty\else
4862 \directlua{ Babel.addpatterns(
4863 [[\bbbl@patterns@]], \number\language) }%
4864 \fi
4865 \@ifundefined{bbbl@patterns@#1}%
4866 \@empty
4867 {\directlua{ Babel.addpatterns(
4868 [[\space\csname bbl@patterns@#1\endcsname]],
4869 \number\language) }}%
4870 \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
4871 \fi
4872 \endgroup}%
4873 \bbbl@exp{%
4874 \bbbl@ifunset{bbbl@prehc\languagename}{}%
4875 {\bbbl@ifblank{\bbbl@cs{prehc\languagename}}}%
4876 {\prehyphenchar=\bbbl@c1{prehc}\relax}}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbbl@patterns@` for the

global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4877 \@onlypreamble\babelpatterns
4878 \AtEndOfPackage{%
4879   \newcommand\babelpatterns[2][\@empty]{%
4880     \ifx\bbl@patterns@relax
4881       \let\bbl@patterns@\@empty
4882     \fi
4883     \ifx\bbl@pttnlist\@empty\else
4884       \bbl@warning{%
4885         You must not intermingle \string\selectlanguage\space and\%
4886         \string\babelpatterns\space or some patterns will not\%
4887         be taken into account. Reported}%
4888     \fi
4889     \ifx\@empty#1%
4890       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4891     \else
4892       \edef\bbl@tempb{\zap@space#1 \@empty}%
4893       \bbl@for\bbl@tempa\bbl@tempb{%
4894         \bbl@fixname\bbl@tempa
4895         \bbl@iflanguage\bbl@tempa{%
4896           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4897             \ifundefined{bbl@patterns@\bbl@tempa}%
4898               \@empty
4899             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4900             #2}}}%
4901     \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).

```

4902 \directlua{
4903   Babel = Babel or {}
4904   Babel.linebreaking = Babel.linebreaking or {}
4905   Babel.linebreaking.before = {}
4906   Babel.linebreaking.after = {}
4907   Babel.locale = {} % Free to use, indexed with \localeid
4908   function Babel.linebreaking.add_before(func)
4909     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4910     table.insert(Babel.linebreaking.before , func)
4911   end
4912   function Babel.linebreaking.add_after(func)
4913     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4914     table.insert(Babel.linebreaking.after, func)
4915   end
4916 }
4917 \def\bbl@intraspace#1 #2 #3\@@{%
4918   \directlua{
4919     Babel = Babel or {}
4920     Babel.intraspaces = Babel.intraspaces or {}
4921     Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
4922       {b = #1, p = #2, m = #3}
4923     Babel.locale_props[\the\localeid].intraspace = %

```

```

4924      {b = #1, p = #2, m = #3}
4925  }}
4926 \def\bbl@intrapenalty#1\@@{%
4927   \directlua{
4928     Babel = Babel or {}
4929     Babel.intrapenalties = Babel.intrapenalties or {}
4930     Babel.intrapenalties['\csname bbl@sbc@p\language\endcsname'] = #1
4931     Babel.locale_props[\the\localeid].intrapenalty = #1
4932   }}
4933 \begingroup
4934 \catcode`\%=12
4935 \catcode`\^=14
4936 \catcode`\'=12
4937 \catcode`\~=12
4938 \gdef\bbl@seaintraspace{^
4939   \let\bbl@seaintraspace\relax
4940   \directlua{
4941     Babel = Babel or {}
4942     Babel.sea_enabled = true
4943     Babel.sea_ranges = Babel.sea_ranges or {}
4944     function Babel.set_chranges (script, chrng)
4945       local c = 0
4946       for s, e in string.gmatch(chrng..' ', '(-)%%.(-)%s') do
4947         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4948         c = c + 1
4949       end
4950     end
4951     function Babel.sea_disc_to_space (head)
4952       local sea_ranges = Babel.sea_ranges
4953       local last_char = nil
4954       local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4955       for item in node.traverse(head) do
4956         local i = item.id
4957         if i == node.id'glyph' then
4958           last_char = item
4959         elseif i == 7 and item.subtype == 3 and last_char
4960           and last_char.char > 0x0C99 then
4961           quad = font.getfont(last_char.font).size
4962           for lg, rg in pairs(sea_ranges) do
4963             if last_char.char > rg[1] and last_char.char < rg[2] then
4964               lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyl1
4965               local intraspace = Babel.intraspaces[lg]
4966               local intrapenalty = Babel.intrapenalties[lg]
4967               local n
4968               if intrapenalty ~= 0 then
4969                 n = node.new(14, 0) ^^ penalty
4970                 n.penalty = intrapenalty
4971                 node.insert_before(head, item, n)
4972               end
4973               n = node.new(12, 13) ^^ (glue, spaceskip)
4974               node.setglue(n, intraspace.b * quad,
4975                 intraspace.p * quad,
4976                 intraspace.m * quad)
4977               node.insert_before(head, item, n)
4978               node.remove(head, item)
4979             end
4980           end
4981         end
4982       end

```

```

4983     end
4984 }^^
4985 \bbl@luahyphenate}
4986 \catcode`\%=14
4987 \gdef\bbl@cjkintraspac{%
4988   \let\bbl@cjkintraspac\relax
4989   \directlua{
4990     Babel = Babel or {}
4991     require'babel-data-cjk.lua'
4992     Babel.cjk_enabled = true
4993     function Babel.cjk_linebreak(head)
4994       local GLYPH = node.id'glyph'
4995       local last_char = nil
4996       local quad = 655360      % 10 pt = 655360 = 10 * 65536
4997       local last_class = nil
4998       local last_lang = nil
4999
5000       for item in node.traverse(head) do
5001         if item.id == GLYPH then
5002
5003           local lang = item.lang
5004
5005           local LOCALE = node.get_attribute(item,
5006             luatexbase.registernumber'bbl@attr@locale')
5007           local props = Babel.locale_props[LOCALE]
5008
5009           local class = Babel.cjk_class[item.char].c
5010
5011           if class == 'cp' then class = 'cl' end % )] as CL
5012           if class == 'id' then class = 'I' end
5013
5014           local br = 0
5015           if class and last_class and Babel.cjk_breaks[last_class][class] then
5016             br = Babel.cjk_breaks[last_class][class]
5017           end
5018
5019           if br == 1 and props.linebreak == 'c' and
5020             lang ~= \the\l@nohyphenation\space and
5021             last_lang ~= \the\l@nohyphenation then
5022             local intrapenalty = props.intrapenalty
5023             if intrapenalty ~= 0 then
5024               local n = node.new(14, 0)      % penalty
5025               n.penalty = intrapenalty
5026               node.insert_before(head, item, n)
5027             end
5028             local intraspac = props.intraspac
5029             local n = node.new(12, 13)      % (glue, spaceskip)
5030             node.setglue(n, intraspac.b * quad,
5031               intraspac.p * quad,
5032               intraspac.m * quad)
5033             node.insert_before(head, item, n)
5034           end
5035
5036           quad = font.getfont(item.font).size
5037           last_class = class
5038           last_lang = lang
5039         else % if penalty, glue or anything else
5040           last_class = nil
5041         end

```



```

5042     end
5043     lang.hyphenate(head)
5044 end
5045 }%
5046 \bbl@luahyphenate}
5047 \gdef\bbl@luahyphenate{%
5048   \let\bbl@luahyphenate\relax
5049   \directlua{
5050     luatexbase.add_to_callback('hyphenate',
5051     function (head, tail)
5052       if Babel.linebreaking.before then
5053         for k, func in ipairs(Babel.linebreaking.before) do
5054           func(head)
5055         end
5056       end
5057       if Babel.cjk_enabled then
5058         Babel.cjk_linebreak(head)
5059       end
5060       lang.hyphenate(head)
5061       if Babel.linebreaking.after then
5062         for k, func in ipairs(Babel.linebreaking.after) do
5063           func(head)
5064         end
5065       end
5066       if Babel.sea_enabled then
5067         Babel.sea_disc_to_space(head)
5068       end
5069     end,
5070     'Babel.hyphenate')
5071   }
5072 }
5073 \endgroup
5074 \def\bbl@provide@intraspace{%
5075   \bbl@ifunset{\bbl@intsp@language}{}%
5076   {\expandafter\ifx\csname\bbl@intsp@language\endcsname\@empty\else
5077     \bbl@xin@{\bbl@cl{lbrk}}{c}%
5078     \ifin@           % cjk
5079     \bbl@cjk_intraspace
5080     \directlua{
5081       Babel = Babel or {}
5082       Babel.locale_props = Babel.locale_props or {}
5083       Babel.locale_props[\the\localeid].linebreak = 'c'
5084     }%
5085     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5086     \ifx\bbl@KVP@intrapenalty\@nil
5087       \bbl@intrapenalty0\@
5088     \fi
5089   \else           % sea
5090     \bbl@sea_intraspace
5091     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5092     \directlua{
5093       Babel = Babel or {}
5094       Babel.sea_ranges = Babel.sea_ranges or {}
5095       Babel.set_chranges('\bbl@cl{sbc} ',
5096       '\bbl@cl{chrng}')
5097     }%
5098     \ifx\bbl@KVP@intrapenalty\@nil
5099       \bbl@intrapenalty0\@
5100     \fi

```

```

5101      \fi
5102      \fi
5103      \ifx\babel@KVP@intrapenalty\@nil\else
5104      \expandafter\babel@intrapenalty\babel@KVP@intrapenalty\@@
5105      \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.

```

5106 \AddBabelHook{babel-fontspec}{afterextras}{\babel@switchfont}
5107 \AddBabelHook{babel-fontspec}{beforestart}{\babel@ckeckstdfonts}
5108 \DisableBabelHook{babel-fontspec}
5109 <<Font selection>>

```

13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5110 \directlua{
5111 Babel.script_blocks = {
5112   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5113              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5114   ['Armn'] = {{0x0530, 0x058F}},
5115   ['Beng'] = {{0x0980, 0x09FF}},
5116   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5117   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5118   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5119              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5120   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5121   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5122              {0xAB00, 0xAB2F}},
5123   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5124   % Don't follow strictly Unicode, which places some Coptic letters in
5125   % the 'Greek and Coptic' block
5126   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5127   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5128              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5129              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5130              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5131              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5132              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5133   ['Hebr'] = {{0x0590, 0x05FF}},

```

```

5134 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5135           {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5136 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5137 ['Knda'] = {{0x0C80, 0x0CFF}},
5138 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5139           {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5140           {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5141 ['Laoo'] = {{0x0E80, 0x0EFF}},
5142 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5143           {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5144           {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5145 ['Mahj'] = {{0x11150, 0x1117F}},
5146 ['Mlym'] = {{0x0D00, 0x0D7F}},
5147 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5148 ['Orya'] = {{0x0B00, 0x0B7F}},
5149 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5150 ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5151 ['Taml'] = {{0x0B80, 0x0BFF}},
5152 ['Telu'] = {{0x0C00, 0x0C7F}},
5153 ['Tfng'] = {{0x2D30, 0x2D7F}},
5154 ['Thai'] = {{0x0E00, 0x0E7F}},
5155 ['Tibt'] = {{0x0F00, 0x0FFF}},
5156 ['Vaii'] = {{0xA500, 0xA63F}},
5157 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5158 }
5159
5160 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5161 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5162 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5163
5164 function Babel.locale_map(head)
5165   if not Babel.locale_mapped then return head end
5166
5167   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5168   local GLYPH = node.id('glyph')
5169   local inmath = false
5170   local toloc_save
5171   for item in node.traverse(head) do
5172     local toloc
5173     if not inmath and item.id == GLYPH then
5174       % Optimization: build a table with the chars found
5175       if Babel.chr_to_loc[item.char] then
5176         toloc = Babel.chr_to_loc[item.char]
5177       else
5178         for lc, maps in pairs(Babel.loc_to_scr) do
5179           for _, rg in pairs(maps) do
5180             if item.char >= rg[1] and item.char <= rg[2] then
5181               Babel.chr_to_loc[item.char] = lc
5182               toloc = lc
5183               break
5184             end
5185           end
5186         end
5187       end
5188       % Now, take action, but treat composite chars in a different
5189       % fashion, because they 'inherit' the previous locale. Not yet
5190       % optimized.
5191       if not toloc and
5192         (item.char >= 0x0300 and item.char <= 0x036F) or

```

```

5193         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5194         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5195             toloc = toloc_save
5196         end
5197         if toloc and toloc > -1 then
5198             if Babel.locale_props[toloc].lg then
5199                 item.lang = Babel.locale_props[toloc].lg
5200                 node.set_attribute(item, LOCALE, toloc)
5201             end
5202             if Babel.locale_props[toloc]['/'..item.font] then
5203                 item.font = Babel.locale_props[toloc]['/'..item.font]
5204             end
5205             toloc_save = toloc
5206         end
5207         elseif not inmath and item.id == 7 then
5208             item.replace = item.replace and Babel.locale_map(item.replace)
5209             item.pre      = item.pre and Babel.locale_map(item.pre)
5210             item.post     = item.post and Babel.locale_map(item.post)
5211         elseif item.id == node.id'math' then
5212             inmath = (item.subtype == 0)
5213         end
5214     end
5215     return head
5216 end
5217 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

5218 \newcommand\babelcharproperty[1]{%
5219   \count@=#1\relax
5220   \ifvmode
5221     \expandafter\bbl@chprop
5222   \else
5223     \bbl@error{\string\babelcharproperty\space can be used only in\%
5224       vertical mode (preamble or between paragraphs)}%
5225     {See the manual for futher info}%
5226   \fi}
5227 \newcommand\bbl@chprop[3][\the\count@]{%
5228   \@tempcnta=#1\relax
5229   \bbl@ifunset{\bbl@chprop@#2}%
5230   {\bbl@error{No property named '#2'. Allowed values are\%
5231     direction (bc), mirror (bmg), and linebreak (lb)}%
5232     {See the manual for futher info}}%
5233   }%
5234   \loop
5235     \bbl@cs{chprop@#2}{#3}%
5236   \ifnum\count@<\@tempcnta
5237     \advance\count@\@ne
5238   \repeat}
5239 \def\bbl@chprop@direction#1{%
5240   \directlua{
5241     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5242     Babel.characters[\the\count@]['d'] = '#1'
5243   }}
5244 \let\bbl@chprop@bc\bbl@chprop@direction
5245 \def\bbl@chprop@mirror#1{%
5246   \directlua{
5247     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5248     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

5249 }}
5250 \let\bbl@chprop@bmg\bbl@chprop@mirror
5251 \def\bbl@chprop@linebreak#1{%
5252   \directlua{
5253     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5254     Babel.cjk_characters[\the\count@]['c'] = '#1'
5255   }}
5256 \let\bbl@chprop@lb\bbl@chprop@linebreak
5257 \def\bbl@chprop@locale#1{%
5258   \directlua{
5259     Babel.chr_to_loc = Babel.chr_to_loc or {}
5260     Babel.chr_to_loc[\the\count@] =
5261       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5262   }}

```

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.

```

5263 \begingroup
5264 \catcode`\#=12
5265 \catcode`\%=12
5266 \catcode`\&=14
5267 \directlua{
5268   Babel.linebreaking.post_replacements = {}
5269   Babel.linebreaking.pre_replacements = {}
5270
5271   function Babel.str_to_nodes(fn, matches, base)
5272     local n, head, last
5273     if fn == nil then return nil end
5274     for s in string.utfvalues(fn(matches)) do
5275       if base.id == 7 then
5276         base = base.replace
5277       end
5278       n = node.copy(base)
5279       n.char = s
5280       if not head then
5281         head = n
5282       else
5283         last.next = n
5284       end
5285       last = n
5286     end
5287     return head
5288   end
5289
5290   function Babel.fetch_word(head, funct)
5291     local word_string = ''
5292     local word_nodes = {}

```

```

5293     local lang
5294     local item = head
5295     local inmath = false
5296
5297     while item do
5298
5299         if item.id == 29
5300             and not(item.char == 124) && ie, not |
5301             and not(item.char == 61) && ie, not =
5302             and not inmath
5303             and (item.lang == lang or lang == nil) then
5304             lang = lang or item.lang
5305             word_string = word_string .. unicode.utf8.char(item.char)
5306             word_nodes[#word_nodes+1] = item
5307
5308         elseif item.id == 7 and item.subtype == 2 and not inmath then
5309             word_string = word_string .. '='
5310             word_nodes[#word_nodes+1] = item
5311
5312         elseif item.id == 7 and item.subtype == 3 and not inmath then
5313             word_string = word_string .. '|'
5314             word_nodes[#word_nodes+1] = item
5315
5316         elseif item.id == 11 and item.subtype == 0 then
5317             inmath = true
5318
5319         elseif word_string == '' then
5320             && pass
5321
5322         else
5323             return word_string, word_nodes, item, lang
5324         end
5325
5326         item = item.next
5327     end
5328 end
5329
5330 function Babel.post_hyphenate_replace(head)
5331     local u = unicode.utf8
5332     local lbkr = Babel.linebreaking.post_replacements
5333     local word_head = head
5334
5335     while true do
5336         local w, wn, nw, lang = Babel.fetch_word(word_head)
5337         if not lang then return head end
5338
5339         if not lbkr[lang] then
5340             break
5341         end
5342
5343         for k=1, #lbkr[lang] do
5344             local p = lbkr[lang][k].pattern
5345             local r = lbkr[lang][k].replace
5346
5347             while true do
5348                 local matches = { u.match(w, p) }
5349                 if #matches < 2 then break end
5350
5351                 local first = table.remove(matches, 1)

```

```

5352         local last = table.remove(matches, #matches)
5353
5354         %% Fix offsets, from bytes to unicode.
5355         first = u.len(w:sub(1, first-1)) + 1
5356         last = u.len(w:sub(1, last-1))
5357
5358         local new %% used when inserting and removing nodes
5359         local changed = 0
5360
5361         %% This loop traverses the replace list and takes the
5362         %% corresponding actions
5363         for q = first, last do
5364             local crep = r[q-first+1]
5365             local char_node = wn[q]
5366             local char_base = char_node
5367
5368             if crep and crep.data then
5369                 char_base = wn[crep.data+first-1]
5370             end
5371
5372             if crep == {} then
5373                 break
5374             elseif crep == nil then
5375                 changed = changed + 1
5376                 node.remove(head, char_node)
5377             elseif crep and (crep.pre or crep.no or crep.post) then
5378                 changed = changed + 1
5379                 d = node.new(7, 0) %% (disc, discretionary)
5380                 d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5381                 d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5382                 d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5383                 d.attr = char_base.attr
5384                 if crep.pre == nil then %% TeXbook p96
5385                     d.penalty = crep.penalty or tex.hyphenpenalty
5386                 else
5387                     d.penalty = crep.penalty or tex.exhyphenpenalty
5388                 end
5389                 head, new = node.insert_before(head, char_node, d)
5390                 node.remove(head, char_node)
5391                 if q == 1 then
5392                     word_head = new
5393                 end
5394             elseif crep and crep.string then
5395                 changed = changed + 1
5396                 local str = crep.string(matches)
5397                 if str == '' then
5398                     if q == 1 then
5399                         word_head = char_node.next
5400                     end
5401                     head, new = node.remove(head, char_node)
5402                 elseif char_node.id == 29 and u.len(str) == 1 then
5403                     char_node.char = string.utfvalue(str)
5404                 else
5405                     local n
5406                     for s in string.utfvalues(str) do
5407                         if char_node.id == 7 then
5408                             log('Automatic hyphens cannot be replaced, just removed.')
5409                         else
5410                             n = node.copy(char_base)

```

```

5411         end
5412         n.char = s
5413         if q == 1 then
5414             head, new = node.insert_before(head, char_node, n)
5415             word_head = new
5416         else
5417             node.insert_before(head, char_node, n)
5418         end
5419     end
5420
5421     node.remove(head, char_node)
5422     end %% string length
5423     end %% if char and char.string
5424     end %% for char in match
5425     if changed > 20 then
5426         texio.write('Too many changes. Ignoring the rest.')
5427     elseif changed > 0 then
5428         w, wn, nw = Babel.fetch_word(word_head)
5429     end
5430
5431     end %% for match
5432     end %% for patterns
5433     word_head = nw
5434     end %% for words
5435     return head
5436 end
5437
5438 %%%
5439 %% Preliminary code for \babelprehyphenation
5440 %% TODO. Copy paste pattern. Merge with fetch_word
5441 function Babel.fetch_subtext(head, funct)
5442     local word_string = ''
5443     local word_nodes = {}
5444     local lang
5445     local item = head
5446     local inmath = false
5447
5448     while item do
5449
5450         if item.id == 29 then
5451             local locale = node.get_attribute(item, Babel.attr_locale)
5452
5453             if not(item.char == 124) %% ie, not | = space
5454                 and not inmath
5455                 and (locale == lang or lang == nil) then
5456                 lang = lang or locale
5457                 word_string = word_string .. unicode.utf8.char(item.char)
5458                 word_nodes[#word_nodes+1] = item
5459             end
5460
5461             if item == node.tail(head) then
5462                 item = nil
5463                 return word_string, word_nodes, item, lang
5464             end
5465
5466             elseif item.id == 12 and item.subtype == 13 and not inmath then
5467                 word_string = word_string .. '|'
5468                 word_nodes[#word_nodes+1] = item
5469

```



```

5470         if item == node.tail(head) then
5471             item = nil
5472             return word_string, word_nodes, item, lang
5473         end
5474
5475         elseif item.id == 11 and item.subtype == 0 then
5476             inmath = true
5477
5478         elseif word_string == '' then
5479             %% pass
5480
5481         else
5482             return word_string, word_nodes, item, lang
5483         end
5484
5485         item = item.next
5486     end
5487 end
5488
5489 %% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5490 function Babel.pre_hyphenate_replace(head)
5491     local u = unicode.utf8
5492     local lbkr = Babel.linebreaking.pre_replacements
5493     local word_head = head
5494
5495     while true do
5496         local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5497         if not lang then return head end
5498
5499         if not lbkr[lang] then
5500             break
5501         end
5502
5503         for k=1, #lbkr[lang] do
5504             local p = lbkr[lang][k].pattern
5505             local r = lbkr[lang][k].replace
5506
5507             while true do
5508                 local matches = { u.match(w, p) }
5509                 if #matches < 2 then break end
5510
5511                 local first = table.remove(matches, 1)
5512                 local last = table.remove(matches, #matches)
5513
5514                 %% Fix offsets, from bytes to unicode.
5515                 first = u.len(w:sub(1, first-1)) + 1
5516                 last = u.len(w:sub(1, last-1))
5517
5518                 local new %% used when inserting and removing nodes
5519                 local changed = 0
5520
5521                 %% This loop traverses the replace list and takes the
5522                 %% corresponding actions
5523                 for q = first, last do
5524                     local crep = r[q-first+1]
5525                     local char_node = wn[q]
5526                     local char_base = char_node
5527
5528                     if crep and crep.data then

```

```

5529         char_base = wn[crep.data+first-1]
5530     end
5531
5532     if crep == {} then
5533         break
5534     elseif crep == nil then
5535         changed = changed + 1
5536         node.remove(head, char_node)
5537     elseif crep and crep.string then
5538         changed = changed + 1
5539         local str = crep.string(matches)
5540         if str == '' then
5541             if q == 1 then
5542                 word_head = char_node.next
5543             end
5544             head, new = node.remove(head, char_node)
5545         elseif char_node.id == 29 and u.len(str) == 1 then
5546             char_node.char = string.utfvalue(str)
5547         else
5548             local n
5549             for s in string.utfvalues(str) do
5550                 if char_node.id == 7 then
5551                     log('Automatic hyphens cannot be replaced, just removed.')
5552                 else
5553                     n = node.copy(char_base)
5554                 end
5555                 n.char = s
5556                 if q == 1 then
5557                     head, new = node.insert_before(head, char_node, n)
5558                     word_head = new
5559                 else
5560                     node.insert_before(head, char_node, n)
5561                 end
5562             end
5563
5564             node.remove(head, char_node)
5565         end &% string length
5566     end &% if char and char.string
5567 end &% for char in match
5568 if changed > 20 then
5569     texio.write('Too many changes. Ignoring the rest.')
5570 elseif changed > 0 then
5571     &% For one-to-one can we modify directly the
5572     &% values without re-fetching? Very likely.
5573     w, wn, nw = Babel.fetch_subtext(word_head)
5574 end
5575
5576     end &% for match
5577 end &% for patterns
5578 word_head = nw
5579 end &% for words
5580 return head
5581 end
5582 &%% end of preliminary code for \babelprehyphenation
5583
5584 &% The following functions belong to the next macro
5585
5586 &% This table stores capture maps, numbered consecutively
5587 Babel.capture_maps = {}

```

```

5588
5589 function Babel.capture_func(key, cap)
5590   local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
5591   ret = ret:gsub('{{([0-9])|([^\]|+)|(.-)}', Babel.capture_func_map)
5592   ret = ret:gsub("%[%[%]%%.%.%", '')
5593   ret = ret:gsub("%.%.%.%.%", '')
5594   return key .. "[=function(m) return ]] .. ret .. [[ end]]
5595 end
5596
5597 function Babel.capt_map(from, mapno)
5598   return Babel.capture_maps[mapno][from] or from
5599 end
5600
5601 &% Handle the {n|abc|ABC} syntax in captures
5602 function Babel.capture_func_map(capno, from, to)
5603   local froms = {}
5604   for s in string.utfcharacters(from) do
5605     table.insert(froms, s)
5606   end
5607   local cnt = 1
5608   table.insert(Babel.capture_maps, {})
5609   local mlen = table.getn(Babel.capture_maps)
5610   for s in string.utfcharacters(to) do
5611     Babel.capture_maps[mlen][froms[cnt]] = s
5612     cnt = cnt + 1
5613   end
5614   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5615     (mlen) .. ").." .. "["
5616 end
5617 }

```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the {*n*} syntax. For example, `pre={1}{1}`- becomes `function(m) return m[1]..m[1]..'-' end`, where *m* are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to *m*[1]. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a T_EX 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).

```

5618 \catcode`\#=6
5619 \gdef\babelposthyphenation#1#2#3{&%
5620   \bbl@activateposthyphen
5621   \begin{group}
5622     \def\babeltempa{\bbl@add@list\babeltempb}&%
5623     \let\babeltempb\@empty
5624     \bbl@foreach{#3}{&%
5625       \bbl@ifsamestring{##1}{remove}&%
5626       {\bbl@add@list\babeltempb{nil}}&%
5627       {\directlua{
5628         local rep = [[##1]]
5629         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5630         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5631         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5632         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5633         tex.print([[string\babeltempa{[]] .. rep .. [[]]])
5634       }}}&%

```

```

5635 \directlua{
5636     local lbkr = Babel.linebreaking.post_replacements
5637     local u = unicode.utf8
5638     &% Convert pattern:
5639     local patt = string.gsub(#[#2]=, '%s', '')
5640     if not u.find(patt, '()', nil, true) then
5641         patt = '()' .. patt .. '()'
5642     end
5643     patt = string.gsub(patt, '%(%)^', '^()')
5644     patt = string.gsub(patt, '%$(%)', '()$')
5645     texio.write('*****' .. patt)
5646     patt = u.gsub(patt, '{(.)}',
5647         function (n)
5648             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5649         end)
5650     lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5651     table.insert(lbkr[\the\csname l@#1\endcsname],
5652         { pattern = patt, replace = { \babeltempb } })
5653 }&%
5654 \endgroup}
5655 % TODO. Working !!! Copypaste pattern.
5656 \gdef\babelprehyphenation#1#2#3{&%
5657 \bbl@activateprehyphen
5658 \beginngroup
5659 \def\babeltempa{\bbl@add@list\babeltempb}&%
5660 \let\babeltempb\@empty
5661 \bbl@foreach{#3}{&%
5662 \bbl@ifsamestring{##1}{remove}&%
5663 {\bbl@add@list\babeltempb{nil}}&%
5664 {\directlua{
5665     local rep = [[#1]]
5666     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5667     tex.print([[string\babeltempa{}}] .. rep .. [[}}]])
5668 }}}&%
5669 \directlua{
5670     local lbkr = Babel.linebreaking.pre_replacements
5671     local u = unicode.utf8
5672     &% Convert pattern:
5673     local patt = string.gsub(#[#2]=, '%s', '')
5674     if not u.find(patt, '()', nil, true) then
5675         patt = '()' .. patt .. '()'
5676     end
5677     patt = u.gsub(patt, '{(.)}',
5678         function (n)
5679             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5680         end)
5681     lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5682     table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5683         { pattern = patt, replace = { \babeltempb } })
5684 }&%
5685 \endgroup}
5686 \endgroup
5687 \def\bbl@activateposthyphen{%
5688 \let\bbl@activateposthyphen\relax
5689 \directlua{
5690     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5691 }}
5692 % TODO. Working !!!
5693 \def\bbl@activateprehyphen{%

```

```

5694 \let\bbl@activateprehyphen\relax
5695 \directlua{
5696     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5697 }}

```

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`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5698 \bbl@trace{Redefinitions for bidi layout}
5699 \ifx\@eqnnum\undefined\else
5700   \ifx\bbl@attr@dir\undefined\else
5701     \edef\@eqnnum{%
5702       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5703       \unexpanded\expandafter{\@eqnnum}}
5704   \fi
5705 \fi
5706 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5707 \ifnum\bbl@bidimode>\z@
5708   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5709     \bbl@exp{%
5710       \mathdir\the\bodydir
5711       #1%           Once entered in math, set boxes to restore values
5712       \<ifmmode>%
5713         \everyvbox{%
5714           \the\everyvbox
5715           \bodydir\the\bodydir
5716           \mathdir\the\mathdir
5717           \everyhbox{\the\everyhbox}%
5718           \everyvbox{\the\everyvbox}}%
5719         \everyhbox{%
5720           \the\everyhbox
5721           \bodydir\the\bodydir
5722           \mathdir\the\mathdir
5723           \everyhbox{\the\everyhbox}%
5724           \everyvbox{\the\everyvbox}}%
5725       \<fi>}}%
5726   \def\@hangfrom#1{%
5727     \setbox\@tempboxa\hbox{#1}%
5728     \hangindent\wd\@tempboxa
5729     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5730       \shapemode\@ne
5731     \fi
5732     \noindent\box\@tempboxa}
5733 \fi
5734 \IfBabelLayout{tabular}
5735 {\let\bbl@OL@tabular\@tabular

```

```

5736 \bbl@replace\@tabular{$}\bbl@nextfake$}%
5737 \let\bbl@NL@tabular\@tabular
5738 \AtBeginDocument{%
5739   \ifx\bbl@NL@tabular\@tabular\else
5740     \bbl@replace\@tabular{$}\bbl@nextfake$}%
5741     \let\bbl@NL@tabular\@tabular
5742   \fi}}
5743 {}
5744 \IfBabelLayout{lists}
5745 {\let\bbl@OL@list\list
5746  \bbl@sreplace\list{\parshape}\bbl@listparshape}%
5747  \let\bbl@NL@list\list
5748  \def\bbl@listparshape#1#2#3{%
5749    \parshape #1 #2 #3 %
5750    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5751      \shapemode\tw@
5752    \fi}}
5753 {}
5754 \IfBabelLayout{graphics}
5755 {\let\bbl@pictresetdir\relax
5756  \def\bbl@pictsetdir{%
5757    \ifcase\bbl@thetextdir
5758      \let\bbl@pictresetdir\relax
5759    \else
5760      \textdir TLT\relax
5761      \def\bbl@pictresetdir{\textdir TRT\relax}%
5762    \fi}%
5763  \let\bbl@OL@picture\@picture
5764  \let\bbl@OL@put\put
5765  \bbl@sreplace\@picture{\hskip-}\bbl@pictsetdir\hskip-}%
5766  \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5767    \@killglue
5768    \raise#2\unitlength
5769    \hb@xt@#1\z@{\kern#1\unitlength\bbl@pictresetdir#3}\hss}}%
5770  \AtBeginDocument
5771    {\ifx\tikz@atbegin@node\undefined\else
5772      \let\bbl@OL@pgfpicture\pgfpicture
5773      \bbl@sreplace\pgfpicture{\pgfpicturetrue}\bbl@pictsetdir\pgfpicturetrue}%
5774      \bbl@add\pgfsys@beginpicture\bbl@pictsetdir}%
5775      \bbl@add\tikz@atbegin@node\bbl@pictresetdir}%
5776    \fi}}
5777 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

```

5778 \IfBabelLayout{counters}%
5779 {\let\bbl@OL@textsuperscript\textsuperscript
5780  \bbl@sreplace\textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5781  \let\bbl@latinarabic=\@arabic
5782  \let\bbl@OL@arabic\@arabic
5783  \def\@arabic#1{\babelsublr\bbl@latinarabic#1}}%
5784  \@ifpackagewith{babel}{bidi=default}%
5785    {\let\bbl@asciroman=\@roman
5786     \let\bbl@OL@roman\@roman
5787     \def\@roman#1{\babelsublr\ensureascii\bbl@asciroman#1}}}%
5788  \let\bbl@asciiRoman=\@Roman
5789  \let\bbl@OL@roman\@Roman
5790  \def\@Roman#1{\babelsublr\ensureascii\bbl@asciiRoman#1}}}%

```

```

5791 \let\bbl@OL@labelenumii\labelenumii
5792 \def\labelenumii{}\theenumii{}%
5793 \let\bbl@OL@p@enumiii\p@enumiii
5794 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
5795 <<Footnote changes>>
5796 \IfBabelLayout{footnotes}%
5797 {\let\bbl@OL@footnote\footnote
5798 \BabelFootnote\footnote\languagename{}\}\}%
5799 \BabelFootnote\localfootnote\languagename{}\}\}%
5800 \BabelFootnote\mainfootnote{}\}\}\}\}
5801 {}

```

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

```

5802 \IfBabelLayout{extras}%
5803 {\let\bbl@OL@underline\underline
5804 \bbl@sreplace\underline{\$@@@underline}\bbl@nextfake$@@@underline}%
5805 \let\bbl@OL@LaTeX2e\LaTeX2e
5806 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5807 \if b\expandafter\@car\@f@series\@nil\boldmath\fi
5808 \babelsublr}%
5809 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
5810 {}
5811 </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.

```

5812 <*basic-r>
5813 Babel = Babel or {}
5814
5815 Babel.bidi_enabled = true
5816
5817 require('babel-data-bidi.lua')
5818
5819 local characters = Babel.characters
5820 local ranges = Babel.ranges
5821
5822 local DIR = node.id("dir")
5823
5824 local function dir_mark(head, from, to, outer)
5825   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5826   local d = node.new(DIR)
5827   d.dir = '+' .. dir
5828   node.insert_before(head, from, d)
5829   d = node.new(DIR)
5830   d.dir = '-' .. dir
5831   node.insert_after(head, to, d)
5832 end
5833
5834 function Babel.bidi(head, ispar)
5835   local first_n, last_n          -- first and last char with nums
5836   local last_es                  -- an auxiliary 'last' used with nums
5837   local first_d, last_d          -- first and last char in L/R block
5838   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):

```

5839   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5840   local strong_lr = (strong == 'l') and 'l' or 'r'
5841   local outer = strong
5842
5843   local new_dir = false
5844   local first_dir = false
5845   local inmath = false
5846
5847   local last_lr
5848
5849   local type_n = ''
5850
5851   for item in node.traverse(head) do
5852
5853     -- three cases: glyph, dir, otherwise

```



```

5854   if item.id == node.id'glyph'
5855     or (item.id == 7 and item.subtype == 2) then
5856
5857     local itemchar
5858     if item.id == 7 and item.subtype == 2 then
5859       itemchar = item.replace.char
5860     else
5861       itemchar = item.char
5862     end
5863     local chardata = characters[itemchar]
5864     dir = chardata and chardata.d or nil
5865     if not dir then
5866       for nn, et in ipairs(ranges) do
5867         if itemchar < et[1] then
5868           break
5869         elseif itemchar <= et[2] then
5870           dir = et[3]
5871           break
5872         end
5873       end
5874     end
5875     dir = dir or 'l'
5876     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).

```

5877   if new_dir then
5878     attr_dir = 0
5879     for at in node.traverse(item.attr) do
5880       if at.number == luatexbase.registernumber'bbl@attr@dir' then
5881         attr_dir = at.value % 3
5882       end
5883     end
5884     if attr_dir == 1 then
5885       strong = 'r'
5886     elseif attr_dir == 2 then
5887       strong = 'al'
5888     else
5889       strong = 'l'
5890     end
5891     strong_lr = (strong == 'l') and 'l' or 'r'
5892     outer = strong_lr
5893     new_dir = false
5894   end
5895
5896   if dir == 'nsm' then dir = strong end -- W1

```

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

```

5897   dir_real = dir -- We need dir_real to set strong below
5898   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:

```

5899   if strong == 'al' then
5900     if dir == 'en' then dir = 'an' end -- W2
5901     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6

```

```

5902         strong_lr = 'r'                                -- W3
5903     end

```

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

```

5904     elseif item.id == node.id'dir' and not inmath then
5905         new_dir = true
5906         dir = nil
5907     elseif item.id == node.id'math' then
5908         inmath = (item.subtype == 0)
5909     else
5910         dir = nil          -- Not a char
5911     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>.

```

5912     if dir == 'en' or dir == 'an' or dir == 'et' then
5913         if dir ~= 'et' then
5914             type_n = dir
5915         end
5916         first_n = first_n or item
5917         last_n = last_es or item
5918         last_es = nil
5919     elseif dir == 'es' and last_n then -- W3+W6
5920         last_es = item
5921     elseif dir == 'cs' then          -- it's right - do nothing
5922     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5923         if strong_lr == 'r' and type_n ~= '' then
5924             dir_mark(head, first_n, last_n, 'r')
5925         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5926             dir_mark(head, first_n, last_n, 'r')
5927             dir_mark(head, first_d, last_d, outer)
5928             first_d, last_d = nil, nil
5929         elseif strong_lr == 'l' and type_n ~= '' then
5930             last_d = last_n
5931         end
5932         type_n = ''
5933         first_n, last_n = nil, nil
5934     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:

```

5935     if dir == 'l' or dir == 'r' then
5936         if dir ~= outer then
5937             first_d = first_d or item
5938             last_d = item
5939         elseif first_d and dir ~= strong_lr then
5940             dir_mark(head, first_d, last_d, outer)
5941             first_d, last_d = nil, nil
5942         end
5943     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.

```

5944   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5945       item.char = characters[item.char] and
5946           characters[item.char].m or item.char
5947   elseif (dir or new_dir) and last_lr ~= item then
5948       local mir = outer .. strong_lr .. (dir or outer)
5949       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5950           for ch in node.traverse(node.next(last_lr)) do
5951               if ch == item then break end
5952               if ch.id == node.id'glyph' and characters[ch.char] then
5953                   ch.char = characters[ch.char].m or ch.char
5954               end
5955           end
5956       end
5957   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).

```

5958   if dir == 'l' or dir == 'r' then
5959       last_lr = item
5960       strong = dir_real          -- Don't search back - best save now
5961       strong_lr = (strong == 'l') and 'l' or 'r'
5962   elseif new_dir then
5963       last_lr = nil
5964   end
5965 end

```

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

```

5966   if last_lr and outer == 'r' then
5967       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5968           if characters[ch.char] then
5969               ch.char = characters[ch.char].m or ch.char
5970           end
5971       end
5972   end
5973   if first_n then
5974       dir_mark(head, first_n, last_n, outer)
5975   end
5976   if first_d then
5977       dir_mark(head, first_d, last_d, outer)
5978   end

```

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

```

5979   return node.prev(head) or head
5980 end
5981 </basic-r>

```

And here the Lua code for bidi=basic:

```

5982 <*basic>
5983 Babel = Babel or {}
5984
5985 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5986

```

```

5987 Babel.fontmap = Babel.fontmap or {}
5988 Babel.fontmap[0] = {}      -- l
5989 Babel.fontmap[1] = {}      -- r
5990 Babel.fontmap[2] = {}      -- al/an
5991
5992 Babel.bidi_enabled = true
5993 Babel.mirroring_enabled = true
5994
5995 require('babel-data-bidi.lua')
5996
5997 local characters = Babel.characters
5998 local ranges = Babel.ranges
5999
6000 local DIR = node.id('dir')
6001 local GLYPH = node.id('glyph')
6002
6003 local function insert_implicit(head, state, outer)
6004   local new_state = state
6005   if state.sim and state.eim and state.sim ~= state.eim then
6006     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6007     local d = node.new(DIR)
6008     d.dir = '+' .. dir
6009     node.insert_before(head, state.sim, d)
6010     local d = node.new(DIR)
6011     d.dir = '-' .. dir
6012     node.insert_after(head, state.eim, d)
6013   end
6014   new_state.sim, new_state.eim = nil, nil
6015   return head, new_state
6016 end
6017
6018 local function insert_numeric(head, state)
6019   local new
6020   local new_state = state
6021   if state.san and state.ean and state.san ~= state.ean then
6022     local d = node.new(DIR)
6023     d.dir = '+TLT'
6024     _, new = node.insert_before(head, state.san, d)
6025     if state.san == state.sim then state.sim = new end
6026     local d = node.new(DIR)
6027     d.dir = '-TLT'
6028     _, new = node.insert_after(head, state.ean, d)
6029     if state.ean == state.eim then state.eim = new end
6030   end
6031   new_state.san, new_state.ean = nil, nil
6032   return head, new_state
6033 end
6034
6035 -- TODO - \hbox with an explicit dir can lead to wrong results
6036 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6037 -- was s made to improve the situation, but the problem is the 3-dir
6038 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6039 -- well.
6040
6041 function Babel.bidi(head, ispar, hdir)
6042   local d -- d is used mainly for computations in a loop
6043   local prev_d = ''
6044   local new_d = false
6045

```

```

6046 local nodes = {}
6047 local outer_first = nil
6048 local inmath = false
6049
6050 local glue_d = nil
6051 local glue_i = nil
6052
6053 local has_en = false
6054 local first_et = nil
6055
6056 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6057
6058 local save_outer
6059 local temp = node.get_attribute(head, ATDIR)
6060 if temp then
6061     temp = temp % 3
6062     save_outer = (temp == 0 and 'l') or
6063                 (temp == 1 and 'r') or
6064                 (temp == 2 and 'al')
6065 elseif ispar then -- Or error? Shouldn't happen
6066     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6067 else -- Or error? Shouldn't happen
6068     save_outer = ('TRT' == hdir) and 'r' or 'l'
6069 end
6070 -- when the callback is called, we are just _after_ the box,
6071 -- and the textdir is that of the surrounding text
6072 -- if not ispar and hdir ~= tex.textdir then
6073 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6074 -- end
6075 local outer = save_outer
6076 local last = outer
6077 -- 'al' is only taken into account in the first, current loop
6078 if save_outer == 'al' then save_outer = 'r' end
6079
6080 local fontmap = Babel.fontmap
6081
6082 for item in node.traverse(head) do
6083
6084     -- In what follows, #node is the last (previous) node, because the
6085     -- current one is not added until we start processing the neutrals.
6086
6087     -- three cases: glyph, dir, otherwise
6088     if item.id == GLYPH
6089         or (item.id == 7 and item.subtype == 2) then
6090
6091         local d_font = nil
6092         local item_r
6093         if item.id == 7 and item.subtype == 2 then
6094             item_r = item.replace -- automatic discs have just 1 glyph
6095         else
6096             item_r = item
6097         end
6098         local chardata = characters[item_r.char]
6099         d = chardata and chardata.d or nil
6100         if not d or d == 'nsm' then
6101             for nn, et in ipairs(ranges) do
6102                 if item_r.char < et[1] then
6103                     break
6104                 elseif item_r.char <= et[2] then

```

```

6105         if not d then d = et[3]
6106         elseif d == 'nsm' then d_font = et[3]
6107         end
6108         break
6109     end
6110 end
6111 end
6112 d = d or 'l'
6113
6114 -- A short 'pause' in bidi for mapfont
6115 d_font = d_font or d
6116 d_font = (d_font == 'l' and 0) or
6117           (d_font == 'nsm' and 0) or
6118           (d_font == 'r' and 1) or
6119           (d_font == 'al' and 2) or
6120           (d_font == 'an' and 2) or nil
6121 if d_font and fontmap and fontmap[d_font][item_r.font] then
6122     item_r.font = fontmap[d_font][item_r.font]
6123 end
6124
6125 if new_d then
6126     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6127     if inmath then
6128         attr_d = 0
6129     else
6130         attr_d = node.get_attribute(item, ATDIR)
6131         attr_d = attr_d % 3
6132     end
6133     if attr_d == 1 then
6134         outer_first = 'r'
6135         last = 'r'
6136     elseif attr_d == 2 then
6137         outer_first = 'r'
6138         last = 'al'
6139     else
6140         outer_first = 'l'
6141         last = 'l'
6142     end
6143     outer = last
6144     has_en = false
6145     first_et = nil
6146     new_d = false
6147 end
6148
6149 if glue_d then
6150     if (d == 'l' and 'l' or 'r') ~= glue_d then
6151         table.insert(nodes, {glue_i, 'on', nil})
6152     end
6153     glue_d = nil
6154     glue_i = nil
6155 end
6156
6157 elseif item.id == DIR then
6158     d = nil
6159     new_d = true
6160
6161 elseif item.id == node.id'glue' and item.subtype == 13 then
6162     glue_d = d
6163     glue_i = item

```

```

6164     d = nil
6165
6166     elseif item.id == node.id'math' then
6167         inmath = (item.subtype == 0)
6168
6169     else
6170         d = nil
6171     end
6172
6173     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6174     if last == 'al' and d == 'en' then
6175         d = 'an'          -- W3
6176     elseif last == 'al' and (d == 'et' or d == 'es') then
6177         d = 'on'          -- W6
6178     end
6179
6180     -- EN + CS/ES + EN      -- W4
6181     if d == 'en' and #nodes >= 2 then
6182         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6183             and nodes[#nodes-1][2] == 'en' then
6184             nodes[#nodes][2] = 'en'
6185         end
6186     end
6187
6188     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6189     if d == 'an' and #nodes >= 2 then
6190         if (nodes[#nodes][2] == 'cs')
6191             and nodes[#nodes-1][2] == 'an' then
6192             nodes[#nodes][2] = 'an'
6193         end
6194     end
6195
6196     -- ET/EN                -- W5 + W7->1 / W6->on
6197     if d == 'et' then
6198         first_et = first_et or (#nodes + 1)
6199     elseif d == 'en' then
6200         has_en = true
6201         first_et = first_et or (#nodes + 1)
6202     elseif first_et then    -- d may be nil here !
6203         if has_en then
6204             if last == 'l' then
6205                 temp = 'l'    -- W7
6206             else
6207                 temp = 'en'   -- W5
6208             end
6209         else
6210             temp = 'on'       -- W6
6211         end
6212         for e = first_et, #nodes do
6213             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6214         end
6215         first_et = nil
6216         has_en = false
6217     end
6218
6219     if d then
6220         if d == 'al' then
6221             d = 'r'
6222             last = 'al'

```

```

6223     elseif d == 'l' or d == 'r' then
6224         last = d
6225     end
6226     prev_d = d
6227     table.insert(nodes, {item, d, outer_first})
6228 end
6229
6230 outer_first = nil
6231
6232 end
6233
6234 -- TODO -- repeated here in case EN/ET is the last node. Find a
6235 -- better way of doing things:
6236 if first_et then      -- dir may be nil here !
6237     if has_en then
6238         if last == 'l' then
6239             temp = 'l'    -- W7
6240         else
6241             temp = 'en'   -- W5
6242         end
6243     else
6244         temp = 'on'      -- W6
6245     end
6246     for e = first_et, #nodes do
6247         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6248     end
6249 end
6250
6251 -- dummy node, to close things
6252 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6253
6254 ----- NEUTRAL -----
6255
6256 outer = save_outer
6257 last = outer
6258
6259 local first_on = nil
6260
6261 for q = 1, #nodes do
6262     local item
6263
6264     local outer_first = nodes[q][3]
6265     outer = outer_first or outer
6266     last = outer_first or last
6267
6268     local d = nodes[q][2]
6269     if d == 'an' or d == 'en' then d = 'r' end
6270     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6271
6272     if d == 'on' then
6273         first_on = first_on or q
6274     elseif first_on then
6275         if last == d then
6276             temp = d
6277         else
6278             temp = outer
6279         end
6280         for r = first_on, q - 1 do
6281             nodes[r][2] = temp

```



```

6282         item = nodes[r][1]    -- MIRRORING
6283         if Babel.mirroring_enabled and item.id == GLYPH
6284             and temp == 'r' and characters[item.char] then
6285             local font_mode = font.fonts[item.font].properties.mode
6286             if font_mode ~= 'harf' and font_mode ~= 'plug' then
6287                 item.char = characters[item.char].m or item.char
6288             end
6289         end
6290     end
6291     first_on = nil
6292 end
6293
6294     if d == 'r' or d == 'l' then last = d end
6295 end
6296
6297 ----- IMPLICIT, REORDER -----
6298
6299 outer = save_outer
6300 last = outer
6301
6302 local state = {}
6303 state.has_r = false
6304
6305 for q = 1, #nodes do
6306     local item = nodes[q][1]
6307
6308     outer = nodes[q][3] or outer
6309
6310     local d = nodes[q][2]
6311
6312     if d == 'nsm' then d = last end          -- W1
6313     if d == 'en' then d = 'an' end
6314     local isdir = (d == 'r' or d == 'l')
6315
6316     if outer == 'l' and d == 'an' then
6317         state.san = state.san or item
6318         state.ean = item
6319     elseif state.san then
6320         head, state = insert_numeric(head, state)
6321     end
6322
6323     if outer == 'l' then
6324         if d == 'an' or d == 'r' then      -- im -> implicit
6325             if d == 'r' then state.has_r = true end
6326             state.sim = state.sim or item
6327             state.eim = item
6328         elseif d == 'l' and state.sim and state.has_r then
6329             head, state = insert_implicit(head, state, outer)
6330         elseif d == 'l' then
6331             state.sim, state.eim, state.has_r = nil, nil, false
6332         end
6333     else
6334         if d == 'an' or d == 'l' then
6335             if nodes[q][3] then -- nil except after an explicit dir
6336                 state.sim = item -- so we move sim 'inside' the group
6337             else
6338                 state.sim = state.sim or item
6339             end
6340         end

```

```

6341     state.eim = item
6342     elseif d == 'r' and state.sim then
6343         head, state = insert_implicit(head, state, outer)
6344     elseif d == 'r' then
6345         state.sim, state.eim = nil, nil
6346     end
6347 end
6348
6349 if isdir then
6350     last = d          -- Don't search back - best save now
6351 elseif d == 'on' and state.san then
6352     state.san = state.san or item
6353     state.ean = item
6354 end
6355
6356 end
6357
6358 return node.prev(head) or head
6359 end
6360 </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.

```

6361 <*nil>
6362 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6363 \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.

```

6364 \ifx\l@nil\@undefined
6365   \newlanguage\l@nil
6366   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
6367   \let\bbl@elt\relax
6368   \edef\bbl@languages{% Add it to the list of languages
6369     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}
6370 \fi

```

6371 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```
\captionnil
\datenil 6372 \let\captionsnil\@empty
6373 \let\datenil\@empty
```

```
6374 \ldf@finish{nil}
6375 \</nil>
```

```

6387 \let\@undefined
6388 }
6389 \fi
6390 </bplain | bplain>

```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```

6391 <bplain>\a plain.tex
6392 <bplain>\a lplain.tex

```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```

6393 <bplain>\def\fmtname{babel-plain}
6394 <bplain>\def\fmtname{babel-lplain}

```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

16.2 Emulating some \LaTeX features

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

```

6395 <<*Emulate LaTeX>> ≡
6396 % == Code for plain ==
6397 \def\@empty{}
6398 \def\loadlocalcfg#1{%
6399   \openin0#1.cfg
6400   \ifeof0
6401     \closein0
6402   \else
6403     \closein0
6404     {\immediate\write16{*****}%
6405      \immediate\write16{* Local config file #1.cfg used}%
6406      \immediate\write16{*}%
6407     }
6408     \input #1.cfg\relax
6409   \fi
6410 \endofldef}

```

16.3 General tools

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

```

6411 \long\def\@firstofone#1{#1}
6412 \long\def\@firstoftwo#1#2{#1}
6413 \long\def\@secondoftwo#1#2{#2}
6414 \def\@nnil{\@nil}
6415 \def\@gobbletwo#1#2{}
6416 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6417 \def\@star@or@long#1{%
6418   \@ifstar
6419   {\let\l@ngrel@x\relax#1}%
6420   {\let\l@ngrel@x\long#1}}
6421 \let\l@ngrel@x\relax
6422 \def\@car#1#2\@nil{#1}
6423 \def\@cdr#1#2\@nil{#2}
6424 \let\@typeset@protect\relax
6425 \let\protected@edef\edef
6426 \long\def\@gobble#1{}

```

```

6427 \edef\@backslashchar{\expandafter\@gobble\string\}
6428 \def\strip@prefix#1>{}
6429 \def\g@addto@macro#1#2{%
6430     \toks@\expandafter{#1#2}%
6431     \xdef#1{\the\toks@}}
6432 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6433 \def\@nameuse#1{\csname #1\endcsname}
6434 \def\@ifundefined#1{%
6435     \expandafter\ifx\csname#1\endcsname\relax
6436     \expandafter\@firstoftwo
6437     \else
6438     \expandafter\@secondoftwo
6439     \fi}
6440 \def\@expandtwoargs#1#2#3{%
6441     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6442 \def\zap@space#1 #2{%
6443     #1%
6444     \ifx#2\@empty\else\expandafter\zap@space\fi
6445     #2}
6446 \let\bbl@trace\@gobble

```

\LaTeX 2_ε has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

6447 \ifx\@preamblecmds\@undefined
6448     \def\@preamblecmds{}
6449 \fi
6450 \def\@onlypreamble#1{%
6451     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6452         \@preamblecmds\do#1}}
6453 \@onlypreamble\@onlypreamble

```

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

```

6454 \def\begindocument{%
6455     \@begindocumenthook
6456     \global\let\@begindocumenthook\@undefined
6457     \def\do##1{\global\let##1\@undefined}%
6458     \@preamblecmds
6459     \global\let\do\noexpand}
6460 \ifx\@begindocumenthook\@undefined
6461     \def\@begindocumenthook{}
6462 \fi
6463 \@onlypreamble\@begindocumenthook
6464 \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 `\@endoflfd`.

```

6465 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
6466 \@onlypreamble\AtEndOfPackage
6467 \def\@endoflfd{}
6468 \@onlypreamble\@endoflfd
6469 \let\bbl@afterlang\@empty
6470 \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.

```

6471 \catcode`\&=\z@

```

```

6472 \ifx&if@filesw\@undefined
6473 \expandafter\let\csname if@filesw\expandafter\endcsname
6474 \csname iffalse\endcsname
6475 \fi
6476 \catcode\&=4

Mimick LATEX's commands to define control sequences.

6477 \def\newcommand{\@star@or@long\new@command}
6478 \def\new@command#1{%
6479 \@testopt{\@newcommand#1}0}
6480 \def\@newcommand#1[#2]{%
6481 \@ifnextchar [{\@xargdef#1[#2]}%
6482 {\@argdef#1[#2]}}
6483 \long\def\@argdef#1[#2]#3{%
6484 \@yargdef#1\@ne{#2}{#3}}
6485 \long\def\@xargdef#1[#2][#3]#4{%
6486 \expandafter\def\expandafter#1\expandafter{%
6487 \expandafter\@protected@testopt\expandafter #1%
6488 \csname\string#1\expandafter\endcsname{#3}}%
6489 \expandafter\@yargdef \csname\string#1\endcsname
6490 \tw@{#2}{#4}}
6491 \long\def\@yargdef#1#2#3{%
6492 \@tempcnta#3\relax
6493 \advance \@tempcnta \@ne
6494 \let\@hash@\relax
6495 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6496 \@tempcntb #2%
6497 \@whilenum\@tempcntb <\@tempcnta
6498 \do{%
6499 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6500 \advance\@tempcntb \@ne}%
6501 \let\@hash@###
6502 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6503 \def\providecommand{\@star@or@long\provide@command}
6504 \def\provide@command#1{%
6505 \begingroup
6506 \escapechar\m@ne\def\@gtempa{{\string#1}}%
6507 \endgroup
6508 \expandafter\@ifundefined\@gtempa
6509 {\def\reserved@a{\new@command#1}}%
6510 {\let\reserved@a\relax
6511 \def\reserved@a{\new@command\reserved@a}}%
6512 \reserved@a}%

6513 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6514 \def\declare@robustcommand#1{%
6515 \edef\reserved@a{\string#1}%
6516 \def\reserved@b{#1}%
6517 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6518 \edef#1{%
6519 \ifx\reserved@a\reserved@b
6520 \noexpand\x@protect
6521 \noexpand#1%
6522 \fi
6523 \noexpand\protect
6524 \expandafter\noexpand\csname
6525 \expandafter\@gobble\string#1 \endcsname
6526 }%
6527 \expandafter\new@command\csname
6528 \expandafter\@gobble\string#1 \endcsname

```

```

6529 }
6530 \def\x@protect#1{%
6531   \ifx\protect\@typeset@protect\else
6532     \@x@protect#1%
6533   \fi
6534 }
6535 \catcode`\&=\z@ % Trick to hide conditionals
6536 \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`.

```

6537 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6538 \catcode`\&=4
6539 \ifx\in@\undefined
6540   \def\in@#1#2{%
6541     \def\in@##1#1##2##3\in@{%
6542       \ifx\in@##2\in@false\else\in@true\fi}%
6543     \in@@#2#1\in@\in@@}
6544 \else
6545   \let\bbl@tempa\empty
6546 \fi
6547 \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).

```

6548 \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.

```

6549 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their \LaTeX 2_ε versions; just enough to make things work in plain \TeX environments.

```

6550 \ifx\@tempcnta\undefined
6551   \csname newcount\endcsname\@tempcnta\relax
6552 \fi
6553 \ifx\@tempcntb\undefined
6554   \csname newcount\endcsname\@tempcntb\relax
6555 \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`).

```

6556 \ifx\bye\undefined
6557   \advance\count10 by -2\relax
6558 \fi
6559 \ifx\@ifnextchar\undefined
6560   \def\@ifnextchar#1#2#3{%
6561     \let\reserved@d=#1%
6562     \def\reserved@a{#2}\def\reserved@b{#3}%
6563     \futurelet\@let@token\@ifnch}
6564   \def\@ifnch{%

```

```

6565 \ifx\@let@token\@sptoken
6566 \let\reserved@c\@xifnch
6567 \else
6568 \ifx\@let@token\reserved@
6569 \let\reserved@c\reserved@a
6570 \else
6571 \let\reserved@c\reserved@b
6572 \fi
6573 \fi
6574 \reserved@c}
6575 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6576 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6577 \fi
6578 \def\@testopt#1#2{%
6579 \@ifnextchar[{\#1}{\#1[#2]}}
6580 \def\@protected@testopt#1{%
6581 \ifx\protect\@typeset@protect
6582 \expandafter\@testopt
6583 \else
6584 \@x@protect#1%
6585 \fi}
6586 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6587 #2\relax}\fi}
6588 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6589 \else\expandafter\@gobble\fi{#1}}

```

16.4 Encoding related macros

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

```

6590 \def\DeclareTextCommand{%
6591 \@dec@text@cmd\providecommand
6592 }
6593 \def\ProvideTextCommand{%
6594 \@dec@text@cmd\providecommand
6595 }
6596 \def\DeclareTextSymbol#1#2#3{%
6597 \@dec@text@cmd\chardef#1{#2}#3\relax
6598 }
6599 \def\@dec@text@cmd#1#2#3{%
6600 \expandafter\def\expandafter#2%
6601 \expandafter{%
6602 \csname#3-cmd\expandafter\endcsname
6603 \expandafter#2%
6604 \csname#3\string#2\endcsname
6605 }%
6606 % \let\@ifdefinable\@rc@ifdefinable
6607 \expandafter#1\csname#3\string#2\endcsname
6608 }
6609 \def\@current@cmd#1{%
6610 \ifx\protect\@typeset@protect\else
6611 \noexpand#1\expandafter\@gobble
6612 \fi
6613 }
6614 \def\@changed@cmd#1#2{%
6615 \ifx\protect\@typeset@protect
6616 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6617 \expandafter\ifx\csname ?\string#1\endcsname\relax
6618 \expandafter\def\csname ?\string#1\endcsname{%

```



```

6619         \@changed@x@err{#1}%
6620     }%
6621 \fi
6622 \global\expandafter\let
6623 \csname\cf@encoding\string#1\expandafter\endcsname
6624 \csname ?\string#1\endcsname
6625 \fi
6626 \csname\cf@encoding\string#1%
6627 \expandafter\endcsname
6628 \else
6629 \noexpand#1%
6630 \fi
6631 }
6632 \def\@changed@x@err#1{%
6633 \errhelp{Your command will be ignored, type <return> to proceed}%
6634 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6635 \def\DeclareTextCommandDefault#1{%
6636 \DeclareTextCommand#1?%
6637 }
6638 \def\ProvideTextCommandDefault#1{%
6639 \ProvideTextCommand#1?%
6640 }
6641 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6642 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6643 \def\DeclareTextAccent#1#2#3{%
6644 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6645 }
6646 \def\DeclareTextCompositeCommand#1#2#3#4{%
6647 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6648 \edef\reserved@b{\string#1}%
6649 \edef\reserved@c{%
6650 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6651 \ifx\reserved@b\reserved@c
6652 \expandafter\expandafter\expandafter\ifx
6653 \expandafter\@car\reserved@a\relax\relax\@nil
6654 \@text@composite
6655 \else
6656 \edef\reserved@b##1{%
6657 \def\expandafter\noexpand
6658 \csname#2\string#1\endcsname####1{%
6659 \noexpand\@text@composite
6660 \expandafter\noexpand\csname#2\string#1\endcsname
6661 ####1\noexpand\@empty\noexpand\@text@composite
6662 {##1}%
6663 }%
6664 }%
6665 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6666 \fi
6667 \expandafter\def\csname\expandafter\string\csname
6668 #2\endcsname\string#1-\string#3\endcsname{#4}
6669 \else
6670 \errhelp{Your command will be ignored, type <return> to proceed}%
6671 \errmessage{\string\DeclareTextCompositeCommand\space used on
6672 inappropriate command \protect#1}
6673 \fi
6674 }
6675 \def\@text@composite#1#2#3\@text@composite{%
6676 \expandafter\@text@composite@x
6677 \csname\string#1-\string#2\endcsname

```

```

6678 }
6679 \def\@text@composite@x#1#2{%
6680   \ifx#1\relax
6681     #2%
6682   \else
6683     #1%
6684   \fi
6685 }
6686 %
6687 \def\@strip@args#1:#2-#3\@strip@args{#2}
6688 \def\DeclareTextComposite#1#2#3#4{%
6689   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6690   \bgroup
6691     \lccode`\@=#4%
6692     \lowercase{%
6693   \egroup
6694     \reserved@a @%
6695   }%
6696 }
6697 %
6698 \def\UseTextSymbol#1#2{#2}
6699 \def\UseTextAccent#1#2#3{}
6700 \def\@use@text@encoding#1{}
6701 \def\DeclareTextSymbolDefault#1#2{%
6702   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6703 }
6704 \def\DeclareTextAccentDefault#1#2{%
6705   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6706 }
6707 \def\cf@encoding{OT1}

```

Currently we only use the \LaTeX 2_ϵ method for accents for those that are known to be made active in *some* language definition file.

```

6708 \DeclareTextAccent{"}{OT1}{127}
6709 \DeclareTextAccent{'}{OT1}{19}
6710 \DeclareTextAccent{^}{OT1}{94}
6711 \DeclareTextAccent{`}{OT1}{18}
6712 \DeclareTextAccent{~}{OT1}{126}

```

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

```

6713 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6714 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
6715 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
6716 \DeclareTextSymbol{\textquoteright}{OT1}{`\' }
6717 \DeclareTextSymbol{\i}{OT1}{16}
6718 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

6719 \ifx\scriptsize\@undefined
6720   \let\scriptsize\sevenrm
6721 \fi
6722 % End of code for plain
6723 <</Emulate LaTeX>>

```

A proxy file:

```

6724 <*plain>
6725 \input babel.def
6726 </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).