Babel, a multilingual package for use with LaTeX's standard document classes*

Johannes Braams
Kersengaarde 33
2723 BP Zoetermeer
The Netherlands
babel@braams.xs4all.nl

For version 3.9, Javier Bezos

This manual documents an alpha unstable release

Printed September 11, 2012

Abstract

The standard distribution of LaTeX contains a number of document classes that are meant to be used, but also serve as examples for other users to create their own document classes. These document classes have become very popular among LaTeX users. But it should be kept in mind that they were designed for American tastes and typography. At one time they contained a number of hard-wired texts. This report describes babel, a package that makes use of the new capabilities of TeX version 3 to provide an environment in which documents can be typeset in a language other than US English, or in more than one language.

Contents

*During the development ideas from Nico Poppelier, Piet van Oostrum and many others have been used. Bernd Raichle has provided many helpful suggestions.

1 The user interface

The user interface of this package is quite simple. It consists of a set of commands that switch from one language to another, and a set of commands that deal with shorthands. It is also possible to find out what the current language is.

In LaTeX2e the preamble of the document:

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

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

Another approach is making dutch and english global options in order to let other packages detect and use them:

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

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

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

Package options refer to languages in a generic way. Sometimes they are the actual language, sometimes they are file names loading a language with a different

name, sometimes they are file names loading several languages. Please, read the documentation for specific languages for further info.

2 Selecting languages

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.

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 and additional grouping, like braces `{}`.

This command can be used as environment, too.

`\begin{otherlanguage}` `{`⟨*language*⟩`}` ... `\end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\selectlanguage`, except the language change is (mostly) local to the environment. This environment is required for intermixing left-to-right typesetting with right-to-left typesetting. The language to switch to is specified as an argument to `\begin{otherlanguage}`.

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 and additional grouping, like braces `{}`.

Spaces after the environment are ignored.

`\foreignlanguage` `[`⟨*language*⟩`]{`⟨*text*⟩`}`

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first argument. 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). !!!!! The latter can be lead to unwanted results if the script is different, so a warning will be issued.

`\begin{otherlanguage*}` `{`⟨*language*⟩`}` ... `\end{otherlanguage*}`

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

`\languagename`

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

`\begin{hyphenrules}` `{`⟨*language*⟩`}` ... `\end{hyphenrules}`

The environment hyphenrules can be used to select *only* the hyphenation rules to be used. 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 characters like, say, `'` done by some languages (eg, italian, frenchb, ukraineb). To set hyphenation exceptions, use `\babelhyphenation` (see below).

2.1 Shorthands

Some notes [[!!!! to be rewritten]]:

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 at a certain level (system, language, user) there is a one-char shorthand, two-char ones starting with the same activated char are ignored.

`\useshorthands` `{`⟨*char*⟩`}`

The command `\useshorthands` initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands. However, user shorthands are not always alive, as they may be deactivated by languages (for example, if you define a `"`-shorthands and switch from `german` to `french`, it stops working. !!!!! An starred version to be added.

`\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 3.9 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>`). 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.

As an example of their applications, let's assume you want an unified set of shorthand for discretionaries (languages do not define shorthands consistently, and `"-`, `\-`, `"=` have different meanings). You could start with, say:

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

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

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

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 overriden by user ones.

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

`\aliasshorthand` `{`⟨*original*⟩`}{`⟨*alias*⟩`}`

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character `/` over `"` in typing Polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`. *Please note* the substitute character must *not* have been declared before as shorthand (in such case, `\aliashorthands` is ignored).

The following example shows how to replace a shorthand by another

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

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

`\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). 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}}
```

`\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. If a character is not known to be a shorthand character its category code will be left unchanged.

New 3.9 Note 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.

2.2 Package options

New 3.9 These package options are processed before language options, so that they are taken into account irrespective of its order.

`shorthands=` ⟨*char*⟩⟨*char*⟩... | `off`

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

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

If ' is included, `activeacute` is set; if ` is included, `activegrave` is set. Active characters (like ~) should be preceded by `\string` (otherwise they will be expanded by LaTeX before they are passed to the package and therefore they will not be recognized).

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 below.

safe= `none` | `ref` | `bib`

Some LaTeX macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from varioref and ifthen). With `safe=none` no macro is redefined. Of course, in such a case you cannot use shorthands in these macros.

config= ⟨*file*⟩

Instead of loading `bblopts.cfg`, the file ⟨*file*⟩`.cfg` is loaded.

main= ⟨*language*⟩

Sets the main language, as explained above.

headfoot= ⟨*language*⟩

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.

`strings=` (!!!! Work in progress.) Selects the encoding of strings in languages supporting this feature. Predefines values are `generic` (for traditional TeX), `unicode` (for engines like XeTeX and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (`T1`, `T2A`, `LGR`, `L7X`...), but only in languages supporting them.

`noconfig` (!!!!Not implemented in full.) Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. The key `config` still works.

For some languages babel supports the options activeacute and activegrave.

\babelshorthand `{`⟨*shorthand*⟩`}`

You can use shorthands declared in languagse file but not activated in `shorthands` with this command; for example `\babelshorthand{"u}` or `\babelshorthand{:}`. (You can conveniently define your own macros or even you own user shorthands.)

2.3 Hyphen tools

\babelhyphen *{⟨*type*⟩}
\babelhyphen *{⟨*text*⟩}

New 3.9 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 oportunity or, in TEX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking oportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking oportunity.

In TeX, `-` and `\-` forbid further breaking oportunities in the word. This is the desired behaviour very often, but not always, an therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, in Dutch, Portugese, Catalan or Danish, `"-` is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian, it is a soft hyphen. Furthermore, some of them even redefine `\-`, so that you cannot insert a soft hyphen without breaking oportunities in the rest of the word.

Therefore, some macros are provide with a set of basic "hyphens" with can be used by themselves, to define an user shorthand, or even in language files.

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.
- `\babelhyphen{double}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portugese and Spanish.
- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it.
- `\babelhyphen{empty}` inserts a break oportunity without a hyphen at all.
- `\babelhyphen{<text>}` is a hard "hyphen" using `<text>` instead. A typical case is `\babelhyphen{/}`.

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

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

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

\babelhyphenation [⟨*language*⟩,⟨*language*⟩,...]{⟨*exceptions*⟩}

New 3.9 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>` (not set in the preamble by default). Multiple `\babelhyphenation`'s are allowed. For example:

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

2.4 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 used. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

Several language definition files use their own methods to set options. For example, frenchb uses `\frenchbsetup`, magyar (1.5) uses `\magyarOptions` and spanish a set of package options (eg, `es-nolayout`). Macros settting options are also used (eg, `\ProsodicMarksOn` in latin)

2.5 Languages supported by Babel

In the following table all the languages supported by Babel are listed, together with the names of the options with which you can load babel for each language.

Language	Option(s)
Afrikaans	afrikaans
Bahasa	bahasa, indonesian, indon, bahasai, bahasam, malay, meyalu
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

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

2.6 Tips and workarounds

- If you use the document class `book` *and* you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), LaTeX will keep complaining about an undefined label. To prevent such problems, you could revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.

- Both ltxdoc and babel use `\AtBeginDocument` to change some catcodes, and babel reloads hhline 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 hhline (babel, now with the correct catcodes for | and :).

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

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

(A recent version of inputenc is required.)

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

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

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

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

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 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*⟩; 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.

- When a language definition file is loaded, it can define \l@⟨*lang*⟩ to be a dialect of \language0 when \l@⟨*lang*⟩ is undefined.

\<lang>hyphenmins

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

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

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

\providehyphenmins

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

\captions<lang>

The macro \captions⟨*lang*⟩ defines the macros that hold the texts to replace the original hard-wired texts.

\date<lang>

The macro \date⟨*lang*⟩ defines \today and

\extras<lang>

The macro \extras⟨*lang*⟩ contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – it must not used directly.

\noextras<lang>

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

\bbl@declare@ttribute

This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.

\main@language

To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.

\ProvidesLanguage

The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command \ProvidesPackage.

\LdfInit

The macro \LdfInit 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 @-sign, preventing the .ldf file from being processed twice, etc.

\ldf@quit The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.

\ldf@finish The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.

\loadlocalcfg 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 \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.

\substitutefontfamily 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 LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.1 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 LaTeX to give a character the category code 'active'. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

\bbl@activate
\bbl@deactivate The command \bbl@activate is used to change the way an active character expands. \bbl@activate 'switches on' the active behaviour 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.

\bbl@add@special
\bbl@remove@special The TeXbook 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." [1, p. 380] It is used to set text 'verbatim'. To make this work if more characters get a special category code, you have to add this character to the macro \dospecial. LaTeX adds another macro called \@sanitize representing the same character set, but without the curly braces. The macros \bbl@add@special⟨*char*⟩ and \bbl@remove@special⟨*char*⟩ add and remove the character ⟨*char*⟩ to these two sets.

3.2 Support for saving macro definitions

Language definition files may want to *re*define macros that already exist. Therefor a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this[1].

[1] This mechanism was introduced by Bernd Raichle.

\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.

3.3 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. This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`.

3.4 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).

\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.

3.5 Encoding-dependent strings

[[!!!! This is still tentative and the code is incomplete]]]

3.9 Babel 3.9 provides a way de define strings in multiple 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` (the old way to define strings still works and it's used by default). A way to select strings automatically depending on the engine is under study.

It consist is a series of blocks.

\StartBabelCommands {⟨*language-list*⟩}{⟨*selector*⟩} {⟨*group*⟩}
\StartBabelCommands *{⟨*language-list*⟩}{⟨*group*⟩}

A "selector" is a list of valid name in package option `strings` followed by (optional) extra info about the encodings to be used (spaces are ignored). The name `unicode` must be used for XeTeX and LuaTeX (the key `strings` has also two special values: `generic` and `encoded`).

Encoding info is `<` ('from') followed by a charset, which if given sets how the strings should be traslated to the internal representation used by the engine (Unicode in XeTeX an LuaTeX) – it's omitted with ascii strings. Typically, it's `utf8`. A a list of encodings which the strings are expected to work with can be given after `>` ('to'). Recommended, but not mandatory. If repeated, first??last?? ones take precedence.

The starred version is a fallback and therefore must be the last block – if no block has been selected when the starred form is reached, this one is used. If possible, it should be provided always and can be the only block (mainly LGC scripts using the LICR). It can be activated explicitly with `generic`.

group is either `captions`, `date` or `extras` (or a group of yours).

```
\StartBabelCommands\CurrentOption{unicode < utf8 > EU1,EU2}{captions}
\SetBabelString{\chapternname}{utf8-string}

\StartBabelCommands*\CurrentOption{captions}
\SetBabelString{\chapternname}{ascii-maybe-LICR-string}

\EndBabelCommands
```

Selection and strings are separated. No need of `\addto`. If the language is german, just redefine `\germanchaptername`.

\SetBabelString {⟨*macro-name*⟩}{⟨*code*⟩}

Adds `<macro-name>` so the current group, and defines `<lang-macro-name>` to `<code>` (after applying the transformation corresponding to the current "selector").

\EndBabelCommands

Marks the end of the series of blocks.

\SetBabelUppercase {⟨*code*⟩}

Sets code to be executed at `\MakeUppercase`.

\SetBabelLowercase {⟨*code*⟩}

Sets code to be executed at `\MakeLowercase`. !!!! Or just a single `\SetBabelCase` for both. ???? This should be independent from `strings=`...

4 Compatibility with `german.sty`

The file `german.sty` has been one of the sources of inspiration for the babel system. Because of this I wanted to include `german.sty` in the babel system. To be able to do that I had to allow for one incompatibility: in the definition of the macro `\selectlanguage` in `german.sty` the argument is used as the ⟨*number*⟩ for an `\ifcase`. So in this case a call to `\selectlanguage` might look like `\selectlanguage{\german}`.

In the definition of the macro `\selectlanguage` in `babel.def` the argument is used as a part of other macronames, so a call to `\selectlanguage` now looks like `\selectlanguage{german}`. Notice the absence of the escape character. As of version 3.1a of babel both syntaxes are allowed.

All other features of the original `german.sty` have been copied into a new file, called `germanb.sty`[2].

Although the babel system was developed to be used with LaTeX, some of the features implemented in the language definition files might be needed by plain TeX users. Care has been taken that all files in the system can be processed by plain TeX.

5 Compatibility with `ngerman.sty`

When used with the options ngerman or naustrian, babel will provide all features of the package `ngerman`. There is however one exception: The commands for special hyphenation of double consonants (`"ff` etc.) and ck (`"ck`), which are no longer required with the new German orthography, are undefined. With the `ngerman` package, however, these commands will generate appropriate warning messages only.

6 Compatibility with the `french` package

It has been reported to me that the package `french` by Bernard Gaulle (`gaulle@idris.fr`) works together with babel. On the other hand, it seems *not* to work well together with a lot of other packages. Therefore I have decided to no longer load `french.ldf` by default. Instead, when you want to use the package by Bernard Gaulle, you will have to request it specifically, by passing either frenchle or frenchpro as an option to babel.

7 Changes in **Babel** version 3.7

In Babel version 3.7 a number of bugs that were found in version 3.6 are fixed. Also a number of changes and additions have occurred:

[2] The 'b' is added to the name to distinguish the file from Partls' file.

- Shorthands are expandable again. The disadvantage is that one has to type `'{}a` when the acute accent is used as a shorthand character. The advantage is that a number of other problems (such as the breaking of ligatures, etc.) have vanished.
- Two new commands, `\shorthandon` and `\shorthandoff` have been introduced to enable to temporarily switch off one or more shorthands.
- Support for typesetting Greek has been enhanced. Code from the `kdgreek` package (suggested by the author) was added and `\greeknumeral` has been added.
- Support for typesetting Basque is now available thanks to Juan Aguirregabiria.
- Support for typesetting Serbian with Latin script is now available thanks to Dejan Muhamedagić and Jankovic Slobodan.
- Support for typesetting Hebrew (and potential support for typesetting other right-to-left written languages) is now available thanks to Rama Porrat and Boris Lavva.
- Support for typesetting Bulgarian is now available thanks to Georgi Boshnakov.
- Support for typesetting Latin is now available, thanks to Claudio Beccari and Krzysztof Konrad Żelechowski.
- Support for typesetting North Sami is now available, thanks to Regnor Jernsletten.
- The options canadian, canadien and acadien have been added for Canadian English and French use.
- A language attribute has been added to the `\mark...` commands in order to make sure that a Greek header line comes out right on the last page before a language switch.
- Hyphenation pattern files are now read *inside a group*; therefore any changes a pattern file needs to make to lowercase codes, uppercase codes, and category codes are kept local to that group. If they are needed for the language, these changes will need to be repeated and stored in `\extras...`
- The concept of language attributes is introduced. It is intended to give the user some control over the features a language-definition file provides. Its first use is for the Greek language, where the user can choose the *πολυτονικό* ("Polutoniko" or multi-accented) Greek way of typesetting texts. These attributes will possibly find wider use in future releases.
- The environment hyphenrules is introduced.

- The syntax of the file `language.dat` has been extended to allow (optionally) specifying the font encoding to be used while processing the patterns file.
- The command `\providehyphenmins` should now be used in language definition files in order to be able to keep any settings provided by the pattern file.

8 Changes in Babel version 3.6

In Babel version 3.6 a number of bugs that were found in version 3.5 are fixed. Also a number of changes and additions have occurred:

- A new environment otherlanguage* is introduced. it only switches the 'specials', but leaves the 'captions' untouched.
- The shorthands are no longer fully expandable. Some problems could only be solved by peeking at the token following an active character. The advantage is that `'{}a` works as expected for languages that have the `'` active.
- Support for typesetting french texts is much enhanced; the file `francais.ldf` is now replaced by `frenchb.ldf` which is maintained by Daniel Flipo.
- Support for typesetting the russian language is again available. The language definition file was originally developed by Olga Lapko from CyrTUG. The fonts needed to typeset the russian language are now part of the babel distribution. The support is not yet up to the level which is needed according to Olga, but this is a start.
- Support for typesetting greek texts is now also available. What is offered in this release is a first attempt; it will be enhanced later on by Yannis Haralambous.
- in babel 3.6j some hooks have been added for the development of support for Hebrew typesetting.
- Support for typesetting texts in Afrikaans (a variant of Dutch, spoken in South Africa) has been added to `dutch.ldf`.
- Support for typesetting Welsh texts is now available.
- A new command `\aliasshorthand` is introduced. It seems that in Poland various conventions are used to type the necessary Polish letters. It is now possible to use the character `/` as a shorthand character instead of the character `"`, by issuing the command `\aliasshorthand{"}{/}`.
- The shorthand mechanism now deals correctly with characters that are already active.
- Shorthand characters are made active at `\begin{document}`, not earlier. This is to prevent problems with other packages.

- A *preambleonly* command `\substitutefontfamily` has been added to create `.fd` files on the fly when the font families of the Latin text differ from the families used for the Cyrillic or Greek parts of the text.
- Three new commands `\LdfInit`, `\ldf@quit` and `\ldf@finish` are introduced that perform a number of standard tasks.
- In babel 3.6k the language Ukrainian has been added and the support for Russian typesetting has been adapted to the package 'cyrillic' to be released with the December 1998 release of LaTeX 2_ε.

9 Changes in Babel version 3.5

In Babel version 3.5 a lot of changes have been made when compared with the previous release. Here is a list of the most important ones:

- the selection of the language is delayed until `\begin{document}`, which means you must add appropriate `\selectlanguage` commands if you include `\hyphenation` lists in the preamble of your document.
- babel now has a language environment and a new command `\foreignlanguage`;
- the way active characters are dealt with is completely changed. They are called 'shorthands'; one can have three levels of shorthands: on the user level, the language level, and on 'system level'. A consequence of the new way of handling active characters is that they are now written to auxiliary files 'verbatim';
- A language change now also writes information in the `.aux` file, as the change might also affect typesetting the table of contents. The consequence is that an .aux file generated by a LaTeX format with babel preloaded gives errors when read with a LaTeX format without babel; but I think this probably doesn't occur;
- babel is now compatible with the `inputenc` and `fontenc` packages;
- the language definition files now have a new extension, `ldf`;
- the syntax of the file `language.dat` is extended to be compatible with the `french` package by Bernard Gaulle;
- each language definition file looks for a configuration file which has the same name, but the extension `.cfg`. It can contain any valid LaTeX code.

10 Identification

The file babel.sty[3] is meant for LaTeX 2_ε, therefor we make sure that the format file used is the right one.

\ProvidesLanguage

The identification code for each file is something that was introduced in LaTeX 2_ε. 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.

```
⟨*!package⟩
\ifx\ProvidesFile\@undefined
  \def\ProvidesFile#1[#2 #3 #4]{%
    \wlog{File: #1 #4 #3 <#2>}%
⟨*kernel & patterns⟩
    \toks8{Babel <#3> and hyphenation patterns for }%
⟨/kernel & patterns⟩
    \let\ProvidesFile\@undefined
    }
```

As an alternative for \ProvidesFile we define \ProvidesLanguage here to be used in the language definition files.

```
⟨*kernel⟩
  \def\ProvidesLanguage#1[#2 #3 #4]{%
    \wlog{Language: #1 #4 #3 <#2>}%
    }
\else
```

In this case we save the original definition of \ProvidesFile in \bbl@tempa and restore it after we have stored the version of the file in \toks8.

```
⟨*kernel & patterns⟩
  \let\bbl@tempa\ProvidesFile
  \def\ProvidesFile#1[#2 #3 #4]{%
    \toks8{Babel <#3> and hyphenation patterns for }%
    \bbl@tempa#1[#2 #3 #4]%
    \let\ProvidesFile\bbl@tempa}
⟨/kernel & patterns⟩
```

When \ProvidesFile is defined we give \ProvidesLanguage a similar definition.

```
  \def\ProvidesLanguage#1{%
    \begingroup
      \catcode`\ 10 %
      \@makeother\/%
      \@ifnextchar[%]
        {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
  \def\@provideslanguage#1[#2]{%
    \wlog{Language: #1 #2}%
    \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
    \endgroup}
```

[3]The file described in this section is called babel.dtx, has version number v3.9a-alpha-5 and was last revised on 2012/09/11.

```
⟨/kernel⟩
\fi
⟨/!package⟩
```

Identify each file that is produced from this source file.

```
⟨package⟩\ProvidesPackage{babel}
⟨core⟩\ProvidesFile{babel.def}
⟨kernel & patterns⟩\ProvidesFile{hyphen.cfg}
⟨kernel&!patterns⟩\ProvidesFile{switch.def}
⟨driver&!user⟩\ProvidesFile{babel.drv}
⟨driver & user⟩\ProvidesFile{user.drv}
                 [2012/09/11 v3.9a alpha-5 %
⟨package⟩     The Babel package]
⟨core⟩         Babel common definitions]
⟨kernel⟩      Babel language switching mechanism]
⟨driver⟩]
```

11 The Package File

In order to make use of the features of LaTeX 2_ε, the babel system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options whose name is different from that of the `.ldf` file (like variant spellings). It also takes care of a number of compatibility issues with other packages an defines a few aditional package options.

11.1 key=value options

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

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

```
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
```

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.

```
\DeclareOption{KeepShorthandsActive}{}
```

```
\DeclareOption{noconfig}{}
% \DeclareOption{nomarks}{} %%% ????
% \DeclareOption{delay}{}  %%% ????
```

Handling of package options is done in three passes. [!!! Not very happy with the idea, anyway.] The first one processes options which 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 options with a nil value.

```
⟨*package⟩
\let\bbl@opt@shorthands\@nnil
\let\bbl@opt@config\@nnil
\let\bbl@opt@main\@nnil
\let\bbl@opt@strings\@nnil
\let\bbl@opt@headfoot\@nnil
\let\bbl@opt@safe\@nnil
```

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

```
\def\bbl@a#1=#2\bbl@a{%
  \expandafter\ifx\csname bbl@opt@#1\endcsname\@nnil
    \expandafter\edef\csname bbl@opt@#1\endcsname{#2}%
  \else
    \PackageError{babel}{%
      Bad option ‘#1=#2’. Either you have misspelled the\MessageBreak
      key or there is a previous setting of ‘#1’}{%
      Valid keys are ‘shorthands’, ‘config’, ‘strings’, ‘main’,\MessageBreak
      ‘headfoot’, ‘safe’}
  \fi}
```

Now the option list is processed, taking into account only `<key>=<value>` options. `shorthand=off` is set separately. Unrecognized options are saved, because they are language options.

```
\DeclareOption{shorthands=off}{\bbl@a shorthands=\bbl@a}
\DeclareOption*{%
  \@expandtwoargs\in@{\string=}{\CurrentOption}%
  \ifin@
    \expandafter\bbl@a\CurrentOption\bbl@a
  \else
    \edef\bbl@language@opts{%
      \ifx\bbl@language@opts\@undefined\@empty\else\bbl@language@opts,\fi
      \CurrentOption}%
  \fi}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
\DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
\DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
```

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

```
\ProcessOptions*
```

11.2 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. In this mode, some macros are removed and one is added (`\babelshorthand`).

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

```
%    \begin{macrocode}
%      A bit of optimization. Some code makes sense only with
%      |shorthands=...|.
%    We make sure all chars are ‘other’, with the help of an auxiliary
%    macro.
%    \begin{macrocode}
\def\bbl@sh@string#1{%
  \ifx#1\@empty\else
    \string#1%
    \expandafter\bbl@sh@string
  \fi}
\ifx\bbl@opt@shorthands\@nnil
  \def\bbl@ifshorthand#1#2#3{#3}%
\else
```

We make sure all chars are ‘other’, with the help of an auxiliary macro.

```
  \def\bbl@sh@string#1{%
    \ifx#1\@empty\else
      \string#1%
      \expandafter\bbl@sh@string
    \fi}
  \edef\bbl@opt@shorthands{%
    \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

The following macros tests if a shortand is one of the allowed ones.

```
  \edef\bbl@ifshorthand#1{%
    \noexpand\expandafter
    \noexpand\bbl@ifsh@i
    \noexpand\string
    #1\bbl@opt@shorthands
    \noexpand\@empty\noexpand\@secondoftwo}
  \def\bbl@aux@ifsh#1\@secondoftwo{\@firstoftwo}
  \def\bbl@ifsh@shi#1#2{%
    \ifx#1#2%
      \expandafter\bbl@aux@ifsh
    \else
      \ifx#2\@empty
        \bbl@afterelse\expandafter\@gobble
      \else
        \bbl@afterfi\expandafter\bbl@ifsh@i
      \fi
    \fi
    #1}
```

The following is ignored with `shorthands=off`, since it is intended to take some aditional actions for certain chars. !!!! 2012/07/04 Code for bbl@languages, to be moved.

```
  \ifx\bbl@opt@shorthands\@empty
    \def\bbl@ifshorthand#1#2#3{#3}%
  \else
    \bbl@ifshorthand{’}%
```

```
      {\PassOptionsToPackage{activeacute}{babel}}{}
    \bbl@ifshorthand{`}%
      {\PassOptionsToPackage{activegrave}{babel}}{}
    % \bbl@ifshorthand{\string:}{}%
    %   {\g@addto@macro\bbl@ignorepackages{,hhline,}}
  \fi
\fi
%   \end{macrocode}
%   !!!! Added 2012/07/30 an experimental code (which misuses
%   \cs{@resetactivechars}) related to babel/3796. With
%   |headfoot=lang| we can set the language used in heads/foots.
%   For example, in babel/3796 just adds |headfoot=english|.
%   \begin{macrocode}
\ifx\bbl@opt@headfoot\@nnil\else
  \g@addto@macro\@resetactivechars{%
    \set@typeset@protect
    \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
    \let\protect\noexpand}
\fi
%
\ifx\bbl@opt@safe\@nnil
  \def\bbl@opt@safe{BR}%
\fi
%
\ifx\bbl@languages\@undefined\else
  \def\bbl@tempa#1/0/#2\@nnil{#1}%
  \edef\bbl@nulllanguage{\expandafter\bbl@tempa\bbl@languages\@nnil}
  \def\@nopatterns#1{%
    \PackageWarningNoLine{babel}%
      {No hyphenation patterns were loaded for\MessageBreak
        the language `#1'\MessageBreak
        I will use the patterns loaded for \bbl@nulllanguage\space
        instead}}
\fi
```

11.3 Language options

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

```
\def\bbl@load@language#1{%
  \let\bbl@last@loaded\CurrentOption
  \@namedef{ds@\CurrentOption}{}%
  \InputIfFileExists{#1.ldf}%
    {\csname\CurrentOption.ldf-h@@k\endcsname}% !!!!! misplaced
    {\PackageError{babel}{%
       Unknow option `\CurrentOption'. Either you misspelled it\MessageBreak
       or the language definition file \CurrentOption.ldf was not found}{%
```

```
      Valid options are: shorthands=..., KeepShorthandsActive,\MessageBreak
      activeacute, activegrave, noconfig, safe=., main=,\MessageBreak
      headfoot=, strings=, config=, or a valid language name.}}}
```

Now, we set language options, but first make sure `\LdfInit` is defined.

```
\ifx\LdfInit\@undefined\input babel.def\relax\fi
\DeclareOption{acadian}{\bbl@load@language{frenchb}}
\DeclareOption{afrikaans}{\bbl@load@language{dutch}}
\DeclareOption{american}{\bbl@load@language{english}}
\DeclareOption{australian}{\bbl@load@language{english}}
\DeclareOption{austrian}{\bbl@load@language{germanb}}
\DeclareOption{bahasa}{\bbl@load@language{bahasai}}
\DeclareOption{bahasai}{\bbl@load@language{bahasai}}
\DeclareOption{bahasam}{\bbl@load@language{bahasam}}
\DeclareOption{brazil}{\bbl@load@language{portuges}}
\DeclareOption{brazilian}{\bbl@load@language{portuges}}
\DeclareOption{british}{\bbl@load@language{english}}
\DeclareOption{canadian}{\bbl@load@language{english}}
\DeclareOption{canadien}{\bbl@load@language{frenchb}}
\DeclareOption{francais}{\bbl@load@language{frenchb}}
\DeclareOption{french}{\bbl@load@language{frenchb}}%
\DeclareOption{german}{\bbl@load@language{germanb}}
\DeclareOption{hebrew}{%
  \input{rlbabel.def}%
  \bbl@load@language{hebrew}}
\DeclareOption{hungarian}{\bbl@load@language{magyar}}
\DeclareOption{indon}{\bbl@load@language{bahasai}}
\DeclareOption{indonesian}{\bbl@load@language{bahasai}}
\DeclareOption{lowersorbian}{\bbl@load@language{lsorbian}}
\DeclareOption{malay}{\bbl@load@language{bahasam}}
\DeclareOption{meyalu}{\bbl@load@language{bahasam}}
\DeclareOption{naustrian}{\bbl@load@language{ngermanb}}
\DeclareOption{newzealand}{\bbl@load@language{english}}
\DeclareOption{ngerman}{\bbl@load@language{ngermanb}}
\DeclareOption{nynorsk}{\bbl@load@language{norsk}}
\DeclareOption{polutonikogreek}{%
  \bbl@load@language{greek}%
  \languageattribute{greek}{polutoniko}}
\DeclareOption{portuguese}{\bbl@load@language{portuges}}
\DeclareOption{russian}{\bbl@load@language{russianb}}
\DeclareOption{UKenglish}{\bbl@load@language{english}}
\DeclareOption{ukrainian}{\bbl@load@language{ukraineb}}
\DeclareOption{uppersorbian}{\bbl@load@language{usorbian}}
\DeclareOption{USenglish}{\bbl@load@language{english}}
```

Now, options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages. If not declared, the name of the option and the file are the same. The last one is saved to check if it is the last loaded (see below).

```
\@for\bbl@a:=\bbl@language@opts\do{%
  \ifx\bbl@a\@empty\else
```

```
    \@ifundefined{ds@\bbl@a}%
      {\edef\bbl@b{\noexpand\DeclareOption{\bbl@a}%
         {\noexpand\bbl@load@language{\bbl@a}}}%
       \bbl@b}%
       \@empty
    \edef\bbl@last@declared{\bbl@a}%
  \fi}
```

Now, we make sure an option is explicitly declared for any language set as global option.

```
\@for\bbl@a:=\@classoptionslist\do{%
  \ifx\bbl@a\@empty\else
    \@ifundefined{ds@\bbl@a}%
      {\IfFileExists{\bbl@a.ldf}%
        {\edef\bbl@b{\noexpand\DeclareOption{\bbl@a}%
           {\noexpand\bbl@load@language{\bbl@a}}}%
         \bbl@b}%
        \@empty}%
      \@empty
  \fi}
```

For all those languages for which the option name is the same as the name of the language specific file we specify a default option, which tries to load the file specified. If this doesn't succeed an error is signalled.

```
\DeclareOption*{}%
```

Another way to extend the list of 'known' options for babel is 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.

```
\def\AtEndOfLanguage#1{%
  \@ifundefined{#1.ldf-h@@k}%
    {\expandafter\let\csname#1.ldf-h@@k\endcsname\@empty}%
    {}%
    \expandafter\g@addto@macro\csname#1.ldf-h@@k\endcsname}
\ifx\bbl@opt@config\@nnil
  \@ifpackagewith{babel}{noconfig}{}%
    {\InputIfFileExists{bblopts.cfg}%
      {\typeout{*************************************^^J%
               * Local config file bblopts.cfg used^^J%
               *}}%
      {}}%
\else
  \InputIfFileExists{\bbl@opt@config.cfg}%
    {\typeout{*************************************^^J%
             * Local config file \bbl@opt@config.cfg used^^J%
             *}}%
    {\PackageError{babel}{%
       Local config file `\bbl@opt@config.cfg' not found}{%
```

```
      Perhaps you misspelled it.}}%
\fi
\ifx\bbl@opt@main\@nnil\else
  \@ifundefined{ds@\bbl@opt@main}%
    {\PackageError{babel}{%
      Unknown language `\bbl@opt@main' in key `main'}{!!!!!}}%
    {\expandafter\let\expandafter\bbl@loadmain
       \csname ds@\bbl@opt@main\endcsname
     \DeclareOption{\bbl@opt@main}{}}
\fi
```

The options have to be processed in the order in which the user specified them:

```
\ProcessOptions*
```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning [?? error] 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. !!!! Not yet finished.

```
\ifx\bbl@loadmain\@undefined
  \ifx\bbl@last@declared\bbl@last@loaded\else
    \PackageWarning{babel}{%
      Last declared language option is `\bbl@last@declared',\MessageBreak
      but the last processed one was `\bbl@last@loaded'.\MessageBreak
      The main language cannot be set as both a global\MessageBreak
      and a package option. Use `main=\bbl@last@declared' as\MessageBreak
      option. Reported}%
  \fi
\else
  \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
  \DeclareOption*{}
  \ProcessOptions*
\fi
```

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.

```
\ifx\bbl@main@language\@undefined
  \PackageError{babel}{%
    You haven't specified a language option}{%
    You need to specify a language, either as a global
    option\MessageBreak
    or as an optional argument to the \string\usepackage\space
    command; \MessageBreak
    You shouldn't try to proceed from here, type x to quit.}
\fi
```

`\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.

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

```
\lowercase{\immediate\openout15=#1#2.fd\relax}%
\immediate\write15{%
  \string\ProvidesFile{#1#2.fd}%
  [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
   \space generated font description file]^^J
  \string\DeclareFontFamily{#1}{#2}{}^^J
  \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
  \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
  \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
  \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
  \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
  \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
  \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
  \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
  }%
\closeout15
}
```

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

```
\@onlypreamble\substitutefontfamily
```

⟨/package⟩

12 The Kernel of Babel

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. 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. The file `babel.def` contains some TEX code that can be read in at run time. When `babel.def` is loaded it checks if `hyphen.cfg` is in the format; if not the file `switch.def` is loaded.

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.

When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format. In that case the file `plain.def` is needed.

```
⟨*kernel | core⟩
\ifx\AtBeginDocument\@undefined
```

But we need to use the second part of `plain.def` (when we load it from `switch.def`) which we can do by defining `\adddialect`.

```
⟨kernel&!patterns⟩   \def\adddialect{}
  \input plain.def\relax
\fi
⟨/kernel | core⟩
```

Check the presence of the command `\iflanguage`, if it is undefined read the file `switch.def`.

```
⟨*core⟩
\input switch.def\relax
⟨/core⟩
```

12.1 Encoding issues (part 1)

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.

```
⟨*core⟩
\AtEndOfPackage{\edef\latinencoding{\cf@encoding}}
```

But this might be overruled with a later loading of the package `fontenc`. Therefor 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.

```
\AtBeginDocument{%
  \gdef\latinencoding{OT1}%
  \ifx\cf@encoding\bbl@t@one
    \xdef\latinencoding{\bbl@t@one}%
  \else
    \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
  \fi
  }
```

`\latintext` Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding.

```
\DeclareRobustCommand{\latintext}{%
  \fontencoding{\latinencoding}\selectfont
  \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.

```
\ifx\@undefined\DeclareTextFontCommand
  \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
\else
  \DeclareTextFontCommand{\textlatin}{\latintext}
\fi
⟨/core⟩
```

We also need to redefine a number of commands to ensure that the right font encoding is used, but this can't be done before `babel.def` is loaded.

12.2 Multiple languages

With TEX version 3.0 it has become possible to load hyphenation patterns for more than one language. This means that some extra administration has to be taken care of. The user has to know for which languages patterns have been loaded, and what values of `\language` have been used.

Some discussion has been going on in the TEX world about how to use `\language`. Some have suggested to set a fixed standard, i. e., patterns for each language should *always* be loaded in the same location. It has also been suggested to use the ISO list for this purpose. Others have pointed out that the ISO list contains more than 256 languages, which have *not* been numbered consecutively.

I think the best way to use `\language`, is to use it dynamically. This code implements an algorithm to do so. It uses an external file in which the person who maintains a TEX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[4]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

This "configuration 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
```

As the file `switch.def` needs to be read only once, we check whether it was read before. If it was, the command `\iflanguage` is already defined, so we can stop processing. 2012/08/14 Commented out

```
⟨*kernel⟩
⟨*!patterns⟩
% \expandafter\ifx\csname iflanguage\endcsname\relax \else
% \expandafter\endinput
% \fi
⟨/!patterns⟩
```

`\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.

```
\ifx\language\@undefined
  \csname newcount\endcsname\language
```

[4]This is because different operating systems sometimes use *very* different file-naming conventions.

```
\fi
```

\last@language

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

```
\ifx\newlanguage\@undefined
  \csname newcount\endcsname\last@language
```

plain TEX version 3.0 uses `\count 19` for this purpose.

```
\else
  \countdef\last@language=19
\fi
```

\addlanguage

To add languages to TEX's memory plain TEX version 3.0 supplies `\newlanguage`, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`.

For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefor `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain TEX version 3.0.

```
\ifx\newlanguage\@undefined
  \def\addlanguage#1{%
    \global\advance\last@language \@ne
    \ifnum\last@language<\@cclvi
    \else
        \errmessage{No room for a new \string\language!}%
    \fi
    \global\chardef#1\last@language
    \wlog{\string#1 = \string\language\the\last@language}}
```

For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`.

```
\else
  \def\addlanguage{\alloc@9\language\chardef\@cclvi}
\fi
```

\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.

```
\def\adddialect#1#2{%
    \global\chardef#1#2\relax
    \wlog{\string#1 = a dialect from \string\language#2}}
```

```
\def\bbl@iflanguage#1{%
  \expandafter\ifx\csname l@#1\endcsname\@undefined
    \@nolanerr{#1}%
    \expandafter\@gobble
  \else
    \expandafter\@firstofone
  \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.

```
\def\iflanguage#1{%
  \bbl@iflanguage{#1}{%
    \ifnum\csname l@#1\endcsname=\language
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}}
```

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

To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character.

To convert a control sequence to a string, we use the `\string` primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer `\escapechar` to a character number, we have to compare this number with the character of the string. To do this we have to use TEX's backquote notation to specify the character as a number.

If the first character of the `\string`'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or `\escapechar` is set to a value outside of the character range 0–255.

If the user gives an empty argument, we provide a default argument for `\string`. This argument should expand to nothing.

```
\let\bbl@select@type\z@
\edef\selectlanguage{%
  \noexpand\protect
  \expandafter\noexpand\csname selectlanguage \endcsname}
```

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

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

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

```
\ifx\documentclass\@undefined
```

```
  \def\xstring{\string\string\string}
\else
  \let\xstring\string
\fi
```

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

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

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

```
\def\bbl@push@language{%
  \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
  }
```

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

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

```
\def\bbl@pop@lang#1+#2-#3{%
  \def\languagename{#1}\xdef#3{#2}%
  }
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way:

```
\def\bbl@pop@language{%
  \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
```

This means that before `\bbl@pop@lang` is executed TEX first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something

has been pushed on the stack) followed by the '-'-sign and finally the reference to the stack.

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

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.

```
\expandafter\def\csname selectlanguage \endcsname#1{%
  \bbl@push@language
  \aftergroup\bbl@pop@language
  \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 historial reasons, language names can be either `language` of `\language`. To catch either form a trick is used, but unfortunately it has the side effect the catcode of letters in `\languagename` is not well-defined.

```
\def\bbl@set@language#1{%
  \edef\languagename{%
    \ifnum\escapechar=\expandafter`\string#1\@empty
    \else \string#1\@empty\fi}%
  \select@language{\languagename}%
```

We also write a command to change the current language in the auxiliary files.

```
  \if@filesw
    \protected@write\@auxout{}{\string\select@language{\languagename}}%
    \addtocontents{toc}{\xstring\select@language{\languagename}}%
    \addtocontents{lof}{\xstring\select@language{\languagename}}%
    \addtocontents{lot}{\xstring\select@language{\languagename}}%
    \csname bbl@hook@write\endcsname
  \fi}
```

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 `\languagename`.

Then we have to *re*define `\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.

```
\def\bbl@switch#1{%
  \originalTeX
  \expandafter\def\expandafter\originalTeX\expandafter{%
    \csname noextras#1\endcsname
    \let\originalTeX\@empty
    \babel@beginsave}%
  \languageshorthands{none}%
```

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. !!!! What if `\hyphenation` was used in `extras` ????

```
  \ifcase\bbl@select@type
    \csname captions#1\endcsname
    \csname date#1\endcsname
  \fi
  \csname bbl@hook@beforeextras\endcsname
  \csname extras#1\endcsname\relax
  \csname bbl@hook@afterextras\endcsname
  \bbl@patterns{\languagename}%
```

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.

```
  \babel@savevariable\lefthyphenmin
  \babel@savevariable\righthyphenmin
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
    \set@hyphenmins\tw@\thr@@\relax
  \else
    \expandafter\expandafter\expandafter\set@hyphenmins
      \csname #1hyphenmins\endcsname\relax
  \fi}
\def\select@language#1{%
  \bbl@iflanguage{#1}{%
    \expandafter\ifx\csname date#1\endcsname\relax
      \@noopterr{#1}%
    \else
      \let\bbl@select@type\z@
      \bbl@switch{#1}%
    \fi
  }}
\def\bbl@iflanguagename#1{%  !!!! or with meaning ????
  \edef\bbl@tempa{\expandafter\bbl@sh@string\languagename\@empty}%
  \edef\bbl@tempb{\expandafter\bbl@sh@string#1\@empty}%
  \ifx\bbl@tempa\bbl@tempb
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
% A bit of optmization:
\def\select@language@x#1{%
  \ifcase\bbl@select@type
    \bbl@iflanguagename{#1}{}{\select@language{#1}}%
  \else
    \select@language{#1}%
  \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 first thing this environment does is store the name of the language in `\languagename`; it then calls `\selectlanguage␣` to switch on everything that is needed for this language The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```
\long\def\otherlanguage#1{%
  \csname selectlanguage \endcsname{#1}%
  \ignorespaces
  }
```

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

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

```
\expandafter\def\csname otherlanguage*\endcsname#1{%
  \foreign@language{#1}%
  }
```

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

```
\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`.

```
\def\foreignlanguage{\protect\csname foreignlanguage \endcsname}
\expandafter\def\csname foreignlanguage \endcsname#1#2{%
  \begingroup
    \foreign@language{#1}%
    #2%
  \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`.

```
\def\foreign@language#1{%
  \def\languagename{#1}%
  \bbl@iflanguage{#1}{%
    \let\bbl@select@type\@ne
    \bbl@switch{#1}}}
```

\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).

```
\def\bbl@patterns#1{%
  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
    \csname l@#1\endcsname
  \else
    \csname l@#1:\f@encoding\endcsname
  \fi\relax
  \@ifundefined{bbl@hyphenation@#1}%
    {\hyphenation{\bbl@hyphenation@}}%
    {\expandafter\ifx\csname bbl@hyphenation@#1\endcsname\@empty\else
       \hyphenation{\bbl@hyphenation@}%
       \hyphenation{\csname bbl@hyphenation@#1\endcsname}%
     \fi}%
  \global\expandafter\let\csname bbl@hyphenation@#1\endcsname\@empty}
```

hyphenrules The environment hyphenrules can be used to select *just* the hyphenation rules. This environment does *not* change `\languagename` 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 `otherlanguage*`.

```
\def\hyphenrules#1{%
  \bbl@iflanguage{#1}{%
    \bbl@patterns{#1}%
    \languageshorthands{none}%
    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
      \set@hyphenmins\tw@\thr@@\relax
    \else
      \expandafter\expandafter\expandafter\set@hyphenmins
      \csname #1hyphenmins\endcsname\relax
    \fi
  }}
\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.

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

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

```
\def\set@hyphenmins#1#2{\lefthyphenmin#1\righthyphenmin#2}
```

`\babelhyphenation` This macros saves hyphenation exceptions. Two macros are used to store them: `\bbl@hyphenation@` for the global ones (preset to empty), and `\bbl@hyphenation<lang>` for language ones (not preset, because there is no way to know which languages are in use; it is set to empty when is has been used in `\bbl@patterns`). We make sure there is a space between words when multiple commands are used.

```
\@onlypreamble\babelhyphenation
\let\bbl@hyphenation@\@empty
\newcommand\babelhyphenation[2][\@empty]{%
  \ifx\@empty#1%
    \ifx\bbl@hyphenation@\@empty\let\bbl@hyphenation@\@gobble\fi
    \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
  \else
    \@for\bbl@tempa:=#1\do{%
     %%% !!!! todo: check language, zapspaces
      \@ifundefined{bbl@hyphenation@\bbl@tempa}%
        {\@namedef{bbl@hyphenation@\bbl@tempa}{\@gobble}}%
        \@empty
      \expandafter\protected@edef\csname bbl@hyphenation@\bbl@tempa\endcsname{%
        \csname bbl@hyphenation@\bbl@tempa\endcsname\space#2}}%
  \fi}
```

`\LdfInit` This macro is defined in two versions. The first version is to be part of the 'kernel' of babel, ie. the part that is loaded in the format; the second version is defined in `babel.def`. The version in the format just checks the category code of the ampersand and then loads `babel.def`.

```
\def\LdfInit{%
  \chardef\atcatcode=\catcode`\@
  \catcode`\@=11\relax
  \input babel.def\relax
```

The category code of the ampersand is restored and the macro calls itself again with the new definition from `babel.def`

```
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \LdfInit}
⟨/kernel⟩
```

The second version of this 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 ampersand. 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.

```
⟨*core⟩
```

```
\def\LdfInit#1#2{%
  \chardef\atcatcode=\catcode`\@
  \catcode`\@=11\relax
```

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. Therefor we store its current catcode and restore it later on.

```
  \chardef\eqcatcode=\catcode`\=
  \catcode`\==12\relax
```

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`.

```
  \expandafter\if\expandafter\@backslashchar
                  \expandafter\@car\string#2\@nil
    \ifx#2\@undefined
    \else
```

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

```
      \ldf@quit{#1}%
    \fi
  \else
```

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

```
    \expandafter\ifx\csname#2\endcsname\relax
    \else
      \ldf@quit{#1}%
    \fi
  \fi
```

Finally we check `\originalTeX`.

```
  \ifx\originalTeX\@undefined
    \let\originalTeX\@empty
  \else
    \originalTeX
  \fi}
```

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

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

```
\def\ldf@finish#1{%
  \loadlocalcfg{#1}%
  \expandafter\main@language\expandafter{#1}%
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \catcode`\==\eqcatcode \let\eqcatcode\relax
  }
```

After the preamble of the document the commands `\LdfInit`, `\ldf@quit` and `\ldf@finish` are no longer needed. Therefor they are turned into warning messages in LaTeX.

```
\@onlypreamble\LdfInit
\@onlypreamble\ldf@quit
\@onlypreamble\ldf@finish
```

`\main@language`
`\bbl@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.

```
\def\main@language#1{%
  \def\bbl@main@language{#1}%
  \let\languagename\bbl@main@language
  \bbl@patterns{\languagename}%
  }
```

The default is to use English as the main language.

```
\ifx\l@english\@undefined
  \chardef\l@english\z@
\fi
\main@language{english}
```

We also have to make sure that some code gets executed at the beginning of the document.

```
\AtBeginDocument{%
  \expandafter\selectlanguage\expandafter{\bbl@main@language}}
⟨/core⟩
```

`\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`.

```
⟨*kernel⟩
\ifx\originalTeX\@undefined\let\originalTeX\@empty\fi
```

Because this part of the code can be included in a format, we make sure that the macro which initialises the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```
\ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi
```

\@nolanerr
\@nopatterns

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

\@noopterr

When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about `\PackageError` it must be LaTeX 2_ε, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

```
\ifx\PackageError\@undefined
  \def\@nolanerr#1{%
    \errhelp{Your command will be ignored, type <return> to proceed}%
    \errmessage{You haven't defined the language #1\space yet}}
  \def\@nopatterns#1{%
    \message{No hyphenation patterns were loaded for}%
    \message{the language `#1'}%
    \message{I will use the patterns loaded for \bbl@nulllanguage\space
          instead}}
  \def\@noopterr#1{%
    \errmessage{The option #1 was not specified in \string\usepackage}
    \errhelp{You may continue, but expect unexpected results}}
  \def\@activated#1{%
    \wlog{Package babel Info: Making #1 an active character}}
\else
  \def\@nolanerr#1{%
    \PackageError{babel}%
                  {You haven't defined the language #1\space yet}%
        {Your command will be ignored, type <return> to proceed}}
  \def\@nopatterns#1{%
    \PackageWarningNoLine{babel}%
        {No hyphenation patterns were loaded for\MessageBreak
          the language `#1'\MessageBreak
          I will use the patterns loaded for \bbl@nulllanguage\space
          instead}}
  \def\@noopterr#1{%
    \PackageError{babel}%
                  {You haven't loaded the option #1\space yet}%
             {You may proceed, but expect unexpected results}}
  \def\@activated#1{%
    \PackageInfo{babel}{%
      Making #1 an active character}}
\fi
```

The following code is meant to be read by iniTeX 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`.

```
⟨*patterns⟩
```

`\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.

```
\def\process@line#1#2 #3/{%
  \ifx=#1
    \process@synonym#2 /
  \else
    \process@language#1#2 #3/%
  \fi
  }
```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with.

```
\toks@{}
\def\process@synonym#1 /{%
  \ifnum\last@language=\m@ne
```

When no languages have been loaded yet, the name following the `=` will be a synonym for hyphenation register 0.

```
    \expandafter\chardef\csname l@#1\endcsname0\relax
    \wlog{\string\l@#1=\string\language0}
```

As no hyphenation patterns are read in yet, we can not yet set the hyphenmin parameters. Therefor a command to do so is stored in a token register and executed when the first pattern file has been processed.

```
    \toks@\expandafter{\the\toks@
      \expandafter\let\csname #1hyphenmins\expandafter\endcsname
      \csname\languagename hyphenmins\endcsname}%
  \else
```

Otherwise the name will be a synonym for the language loaded last.

```
    \expandafter\chardef\csname l@#1\endcsname\last@language
    \wlog{\string\l@#1=\string\language\the\last@language}
```

We also need to copy the hyphenmin parameters for the synonym.

```
    \expandafter\let\csname #1hyphenmins\expandafter\endcsname
    \csname\languagename hyphenmins\endcsname
  \fi
  \xdef\bbl@languages{%
    \ifx\bbl@languages\@undefined\@empty\else\bbl@languages,\fi
    #1/\the\last@language//}%
  }
```

`\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 third argument is optional, so a `/` character is expected to delimit the last argument. 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'.

```
\def\process@language#1 #2 #3/{%
  \expandafter\addlanguage\csname l@#1\endcsname
  \expandafter\language\csname l@#1\endcsname
  \def\languagename{#1}%
```

Then the 'name' of the language that will be loaded now is added to the token register `\toks8`. and finally the pattern file is read.

```
  \global\toks8\expandafter{\the\toks8#1, }%
```

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`.

```
  \begingroup
    \bbl@get@enc#1:\@@@
    \ifx\bbl@hyph@enc\@empty
    \else
      \fontencoding{\bbl@hyph@enc}\selectfont
    \fi
```

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. TEX does not keep track of these assignments. Therefor we try to detect such assignments and store them in the \⟨*lang*⟩`hyphenmins` macro. When no assignments were made we provide a default setting.

```
    \lefthyphenmin\m@ne
```

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

```
    \input #2\relax
```

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

```
    \ifnum\lefthyphenmin=\m@ne
    \else
      \expandafter\xdef\csname #1hyphenmins\endcsname{%
        \the\lefthyphenmin\the\righthyphenmin}%
    \fi
  \endgroup
```

If the counter `\language` is still equal to zero we set the hyphenmin parameters to the values for the language loaded on pattern register 0.

```
  \ifnum\the\language=\z@
    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
      \set@hyphenmins\tw@\thr@@\relax
    \else
      \expandafter\expandafter\expandafter\set@hyphenmins
        \csname #1hyphenmins\endcsname
    \fi
```

Now execute the contents of token register zero as it may contain commands which set the hyphenmin parameters for synonyms that were defined before the first pattern file is read in.

```
    \the\toks@
  \fi
```

Empty the token register after use.

```
  \toks@{}%
```

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.

```
  \def\bbl@tempa{#3}%
  \let\bbl@tempb\@empty
  \ifx\bbl@tempa\@empty
  \else
    \ifx\bbl@tempa\space
    \else
      \input #3\relax
      \def\bbl@tempb{#3}%
    \fi
  \fi
```

`\bbl@languages` saves a snapshot of the loaded languagues in the form ⟨*language*⟩/⟨*number*⟩/⟨*patterns-fi*

```
  \xdef\bbl@languages{%
    \ifx\bbl@languages\@undefined\@empty\else\bbl@languages,\fi
    #1/\the\language/#2/\bbl@tempb}%
  }
```

`\bbl@get@enc` `\bbl@hyph@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.

```
\def\bbl@get@enc#1:#2\@@@{%
```

First store both arguments in temporary macros,

```
  \def\bbl@tempa{#1}%
  \def\bbl@tempb{#2}%
```

then, if the second argument was empty, no font encoding was specified and we're done.

```
  \ifx\bbl@tempb\@empty
    \global\let\bbl@hyph@enc\@empty
  \else
```

But if the second argument was *not* empty it will now have a superfluous colon attached to it which we need to remove. This done by feeding it to `\bbl@get@enc`. The string that we are after will then be in the first argument and be stored in `\bbl@tempa`.

```
    \bbl@get@enc#2\@@@
    \xdef\bbl@hyph@enc{\bbl@tempa}%
  \fi}
```

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

```
\openin1 = language.dat
```

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

```
\ifeof1
  \message{I couldn't find the file language.dat,\space
           I will try the file hyphen.tex}
  \input hyphen.tex\relax
  \def\l@english{0}%
  \def\languagename{english}%
\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.

```
  \last@language\m@ne
```

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

```
  \loop
```

While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
    \endlinechar\m@ne
    \read1 to \bbl@line
    \endlinechar`\^^M
```

Empty lines are skipped.

```
    \ifx\bbl@line\@empty
    \else
```

Now we add a space and a `/` character to the end of `\bbl@line`. This is needed to be able to recognize the third, optional, argument of `\process@language` later on.

```
      \edef\bbl@line{\bbl@line\space/}%
      \expandafter\process@line\bbl@line
      \ifx\bbl@defaultlanguage\@undefined
        \let\bbl@defaultlanguage\languagename
      \fi
    \fi
```

Check for the end of the file. To avoid a new `if` control sequence we create the necessary `\iftrue` or `\iffalse` with the help of `\csname`. But there is one complication with this approach: when skipping the `loop...repeat` TEX has to read `\if`/`\fi` pairs. So we have to insert a 'dummy' `\iftrue`.

```
    \iftrue \csname fi\endcsname
    \csname if\ifeof1 false\else true\fi\endcsname
  \repeat
```

Reactivate the default patterns,

```
  \language=0
  \let\languagename\bbl@defaultlanguage
  \let\bbl@defaultlanguage\@undefined
\fi
```

and close the configuration file.

```
\closein1
```

Also remove some macros from memory

```
\let\process@language\@undefined
\let\process@synonym\@undefined
\let\process@line\@undefined
\let\bbl@tempa\@undefined
\let\bbl@tempb\@undefined
\let\bbl@eq@\@undefined
\let\bbl@line\@undefined
\let\bbl@get@enc\@undefined
```

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

```
\ifx\addto@hook\@undefined
\else
  \expandafter\addto@hook\expandafter\everyjob\expandafter{%
    \expandafter\typeout\expandafter{\the\toks8 loaded.}}
\fi
```

Here the code for iniTEX ends.

```
⟨/patterns⟩
⟨/kernel⟩
```

12.3 Support for active characters

`\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).

To keep all changes local, we begin a new group. Then we redefine the macros `\do` and `\@makeother` to add themselves and the given character without expansion.

```
⟨*core | shorthands⟩
\def\bbl@add@special#1{\begingroup
    \def\do{\noexpand\do\noexpand}%
    \def\@makeother{\noexpand\@makeother\noexpand}%
```

To add the character to the macros, we expand the original macros with the additional character inside the redefinition of the macros. Because `\@sanitize` can be undefined, we put the definition inside a conditional.

```
    \edef\x{\endgroup
      \def\noexpand\dospecials{\dospecials\do#1}%
      \expandafter\ifx\csname @sanitize\endcsname\relax \else
        \def\noexpand\@sanitize{\@sanitize\@makeother#1}%
      \fi}%
```

The macro \x contains at this moment the following:

\endgroup\def\dospecials{*old contents* \do⟨*char*⟩}.

If \@sanitize is defined, it contains an additional definition of this macro. The last thing we have to do, is the expansion of \x. Then \endgroup is executed, which restores the old meaning of \x, \do and \@makeother. After the group is closed, the new definition of \dospecials (and \@sanitize) is assigned.

```
753   \x}
```

\bbl@remove@special

The companion of the former macro is \bbl@remove@special. It is used to remove a character from the set macros \dospecials and \@sanitize.

To keep all changes local, we begin a new group. Then we define a help macro \x, which expands to empty if the characters match, otherwise it expands to its nonexpandable input. Because TEX inserts a \relax, if the corresponding \else or \fi is scanned before the comparison is evaluated, we provide a 'stop sign' which should expand to nothing.

```
754 \def\bbl@remove@special#1{\begingroup
755     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
756                  \else\noexpand##1\noexpand##2\fi}%
```

With the help of this macro we define \do and \make@other.

```
757     \def\do{\x\do}%
758     \def\@makeother{\x\@makeother}%
```

The rest of the work is similar to \bbl@add@special.

```
759     \edef\x{\endgroup
760       \def\noexpand\dospecials{\dospecials}%
761       \expandafter\ifx\csname @sanitize\endcsname\relax \else
762         \def\noexpand\@sanitize{\@sanitize}%
763       \fi}%
764   \x}
```

12.4 Shorthands

\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 the following line in a language definition file:

```
\initiate@active@char{"}
```

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".

\bbl@afterelse
\bbl@afterfi

Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the \else and \fi parts of an \if-statement[5]. These macros will break if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

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

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

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

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

```
\def\bbl@active@def#1#2#3#4{%
  \@namedef{#3#1}{%
    \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
      \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
    \else
      \bbl@afterfi\csname#2@sh@#1@\endcsname
    \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.

```
  \long\@namedef{#3@arg#1}##1{%
    \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
      \bbl@afterelse\csname#4#1\endcsname##1%
    \else
      \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
    \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.

```
\def\initiate@active@char#1{%
  \expandafter\ifx\csname active@char\string#1\endcsname\relax
```

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

```
    \bbl@withactive
      {\expandafter\@initiate@active@char\expandafter}#1\string#1#1%
  \fi}
```

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 treatement to avoid making them `\relax`).

```
\def\@initiate@active@char#1#2#3{%
  \expandafter\edef\csname bbl@oricat@#2\endcsname{%
    \catcode`#2=\the\catcode`#2\relax}%
  \ifx#1\@undefined
    \expandafter\edef\csname bbl@oridef@#2\endcsname{%
      \let\noexpand#1\noexpand\@undefined}%
  \else
    \expandafter\let\csname bbl@oridef@@#2\endcsname#1%
    \expandafter\edef\csname bbl@oridef@#2\endcsname{%
      \let\noexpand#1%
      \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
  \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.

```
  \ifcat\noexpand#3\noexpand#1\relax   % !!!! or just \ifx#1#3 ???
    \expandafter\let\csname normal@char#2\endcsname#3%
  \else
    \@activated{#2}%
    \@namedef{normal@char#2}{#3}%
```

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).

```
    \@ifpackagewith{babel}{KeepShorthandsActive}{}{%
      \edef\bbl@tempa{\catcode`#2\the\catcode`#2\relax}%
      \expandafter\AtEndOfLanguage\expandafter\CurrentOption
        \expandafter{\bbl@tempa}%
      \expandafter\AtEndOfPackage\expandafter{\bbl@tempa}}%
    \AtBeginDocument{%
      \catcode`#2\active
      \if@filesw
        \immediate\write\@mainaux{\string\catcode`#2\string\active}%
      \fi}%
    \expandafter\bbl@add@special\csname#2\endcsname
    \catcode`#2\active
  \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*⟩).

```
\@namedef{active@char#2}{%
  \if@safe@actives
    \bbl@afterelse\csname normal@char#2\endcsname
  \else
    \bbl@afterfi\csname user@active#2\endcsname
  \fi}%
```

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!).

```
\bbl@withactive\xdef#1{%
  \noexpand\active@prefix\noexpand#1%
  \expandafter\noexpand\csname normal@char#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.

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

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

```
\@namedef{\user@group @sh@#2@@}%
  {\csname normal@char#2\endcsname}%
\@namedef{\user@group @sh@#2@\string\protect@}%
  {\csname user@active#2\endcsname}%
```

Finally, a couple of special cases are taken care of. 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'.

```
\ifx'#3%   !!!!! Ensure catcode to other ????
  \let\prim@s\bbl@prim@s
  \@namedef{normal@char#2}{\textormath{#3}{\sp\bgroup\prim@s}}%
  \let\active@math@prime'%
\fi
```

If we are using the caret as a shorthand character special care should be taken to make sure math still works. Therefor an extra level of expansion is introduced with a check for math mode on the upper level – we first expand to `\bbl@act@caret` in order to be able to handle math mode correctly.

```
\ifx#3^%
  \gdef\bbl@act@caret{%
    \textormath
      {\if@safe@actives
         \bbl@afterelse\csname normal@char#2\endcsname
       \else
         \bbl@afterfi\csname user@active#2\endcsname
       \fi}
      {\csname normal@char#2\endcsname}}%
  \@namedef{active@char#2}{\bbl@act@caret}% !!!! Or \let ????
\fi}
```

`\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.

```
\def\bbl@sh@select#1#2{%
  \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
    \bbl@afterelse\bbl@scndcs
  \else
    \bbl@afterfi\csname#1@sh@#2@sel\endcsname
  \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 `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`.

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

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

```
    \ifx\protect\@unexpandable@protect
      \noexpand#1%
    \else
      \protect#1%
    \fi
    \expandafter\@gobble
  \fi}
```

`\if@safe@actives` In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char`⟨*char*⟩.

```
\newif\if@safe@actives
\@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.

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

`\bbl@activate` `\bbl@deactivate` oth macros take one argument, like `\initiate@active@char`. The macro is used to change the definition of an active character to expand to `\active@char`⟨*char*⟩ in the case of `\bbl@activate`, or `\normal@char`⟨*char*⟩ in the case of `\bbl@deactivate`. First, an auxiliary macro is defined with shared code, which also makes sure the catcode is set to active (parameters 1 and 2 are the same here, but different when called from `\aliasshorthand`).

```
\def\bbl@set@activate#1#2#3{%
  \edef#1{%
    \noexpand\active@prefix\noexpand#1%
    \expandafter\noexpand\csname#3@char\string#2\endcsname}}
\def\bbl@activate#1{\bbl@withactive\bbl@set@activate#1#1{active}}
\def\bbl@deactivate#1{\bbl@withactive\bbl@set@activate#1#1{normal}}
```

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

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

```
\def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
\def\@decl@short#1#2#3\@nil#4{%
  \def\bbl@tempa{#3}%
  \ifx\bbl@tempa\@empty
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
    \@ifundefined{#1@sh@\string#2@}{}%
      {\def\bbl@tempa{#4}%
       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
       \else
         \PackageWarning{Babel}%
```

```
          {Redefining #1 shorthand \string#2\MessageBreak
           in language \CurrentOption}%
      \fi}%
    \@namedef{#1@sh@\string#2@}{#4}%
  \else
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
    \@ifundefined{#1@sh@\string#2@\string#3@}{}%
      {\def\bbl@tempa{#4}%
       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
       \else
         \PackageWarning{Babel}%
           {Redefining #1 shorthand \string#2\string#3\MessageBreak
            in language \CurrentOption}%
       \fi}%
    \@namedef{#1@sh@\string#2@\string#3@}{#4}%
  \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.

```
\def\textormath#1#2{%
  \ifmmode
    \bbl@afterelse#2%
  \else
    \bbl@afterfi#1%
  \fi}
```

`\user@group` `\language@group` `\system@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'.

```
\def\user@group{user}
\def\language@group{english}
\def\system@group{system}
```

`\useshorthands` This is the user level command to tell LaTeX that user level shorthands will be used in the document. It takes one argument, the character that starts a shorthand.

```
\def\useshorthands#1{%
```

First note that this is user level.

```
  \def\user@group{user}%
```

Then initialize the character for use as a shorthand character.

```
  \bbl@s@initiate@active@char{#1}%
```

!!!!! Is this the right place to activate it??? I don't think so, but changing that could be bk-inc, so perhaps just document it. Or a starred version useshorthands*

```
  \bbl@s@activate{#1}}%
```

\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.

```
\def\user@language@group{user@\language@group}
\def\bbl@set@user@generic#1#2{%
  \@ifundefined{user@generic@active#1}%
    {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
     \@namedef{#2@sh@#1@@}{\csname normal@char#1\endcsname}%
     \@namedef{#2@sh@#1@\string\protect@}{\csname user@active#1\endcsname}}%
  \@empty}
\newcommand\defineshorthand[3][\@empty]{%
  \ifx\@empty#1% !!!!!!!!!!! simplify ????
    \bbl@s@declare@shorthand{user}{#2}{#3}%
  \else
    \edef\bbl@tempa{\zap@space#1 \@empty}%
    \@for\bbl@tempb:=\bbl@tempa\do{%
      \if*\expandafter\@car\bbl@tempb\@nil
        \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
        \@expandtwoargs
          \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
      \fi
      \declare@shorthand{\bbl@tempb}{#2}{#3}}%
  \fi}
```

\languageshorthands A user level command to change the language from which shorthands are used.

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

\aliasshorthand First the new shorthand needs to be initialized,

```
\def\aliasshorthand#1#2{%
  \expandafter\ifx\csname active@char\string#2\endcsname\relax
     \ifx\document\@notprerr
       \@notshorthand{#2}
     \else
       \initiate@active@char{#2}%
```

Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands{"}{/} is \active@prefix /\active@char".

```
       \bbl@withactive\bbl@set@activate#2#1{active}%
     \fi
   \fi}
```

\@notshorthand

```
\def\@notshorthand#1{%
       \PackageError{babel}{%
         The character ‘\string #1’ should be made
```

```
        a shorthand character;\MessageBreak
        add the command \string\useshorthands\string{#1\string} to
        the preamble.\MessageBreak
        I will ignore your instruction}{}%
  }
```

`\shorthandon` `\shorthandoff` 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.

```
\newcommand*\shorthandon[1]{\bbl@switch@sh{on}#1\@nil}
\DeclareRobustCommand*\shorthandoff{%
  \@ifstar{\bbl@shorthandoff{ori}}{\bbl@shorthandoff{off}}}
\def\bbl@shorthandoff#1#2{\bbl@switch@sh{#1}#2\@nil}
```

`\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.

```
\def\bbl@switch@sh#1#2#3\@nil{%
  \@ifundefined{active@char\string#2}{%
    \PackageError{babel}{%
      The character '\string #2' is not a shorthand character
      in \languagename}{%
      Maybe you made a typing mistake?\MessageBreak
      I will ignore your instruction}}{%
    \csname bbl@switch@sh@#1\endcsname#2}%
```

Now that, as the first character in the list has been taken care of, we pass the rest of the list back to `\bbl@switch@sh`.

```
  \ifx#3\@empty\else
    \bbl@afterfi\bbl@switch@sh{#1}#3\@nil
  \fi}
```

`\bbl@switch@sh@off` All that is left to do is define the actual switching macros. 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.

```
\def\bbl@switch@sh@off#1{\catcode`#112\relax}
\def\bbl@switch@sh@on#1{\catcode`#1\active}
\def\bbl@switch@sh@ori#1{%
  \csname bbl@oricat@\string#1\endcsname
  \csname bbl@oridef@\string#1\endcsname}
```

12.5 Conditional loading of shorthands

!!! To be documented

```
\let\bbl@s@initiate@active@char\initiate@active@char
\let\bbl@s@declare@shorthand\declare@shorthand
\let\bbl@s@switch@sh@on\bbl@switch@sh@on
\let\bbl@s@switch@sh@off\bbl@switch@sh@off
\let\bbl@s@activate\bbl@activate
\let\bbl@s@deactivate\bbl@deactivate
```

!!!!!TO DO: package options are expanded by LaTeX, and raises an error, but not ~. Is there a way to fix it?

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

```
\ifx\bbl@opt@shorthands\@nnil\else
  \def\babelshorthand#1{%
    \@ifundefined{bbl@@\languagename @@\bbl@sh@string#1\@empty}%
      {#1}%
      {\@nameuse{bbl@@\languagename @@\bbl@sh@string#1\@empty}}}
  \def\initiate@active@char#1{%
    \bbl@ifshorthand{#1}%
      {\bbl@s@initiate@active@char{#1}}%
      {\@namedef{active@char\string#1}{}}}%
  \def\declare@shorthand#1#2{%
    \expandafter\bbl@ifshorthand\expandafter{\@car#2\@nil}%
      {\bbl@s@declare@shorthand{#1}{#2}}%
       {\def\bbl@tempa{#2}%
        \@namedef{bbl@@#1@@\bbl@sh@string#2\@empty}}}%
  \def\bbl@switch@sh@on#1{%
    \bbl@ifshorthand{#1}{\bbl@s@switch@sh@on{#1}}\@empty}
  \def\bbl@switch@sh@off#1{%
    \bbl@ifshorthand{#1}{\bbl@s@switch@sh@off{#1}}\@empty}
  \def\bbl@activate#1{%
    \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}\@empty}
  \def\bbl@deactivate#1{%
    \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}\@empty}
\fi
%   \end{macrocode}
%
%    \subsection{System values for some characters}
%
%    To prevent problems with constructs such as |\char"01A| when the
%    double quote is made active, we define a shorthand on system
%    level. This declaration (as well as those based on using
%    |\normal@char|) is in fact redundant, because the latter command
%    will be execucuted eventually if there is no system shorthand.
% \changes{babel~3.5a}{1995/03/10}{Replaced 16 system shorthands to
%    deal with hex numbers by one}
%    \begin{macrocode}
\declare@shorthand{system}{"}{\csname normal@char\string"\endcsname}
```

When the right quote is made active we need to take care of handling it correctly in mathmode. Therefore we define a shorthand at system level to make it

expand to a non-active right quote in textmode, but expand to its original definition in mathmode. (Note that the right quote is ‘active’ in mathmode because of its mathcode.)

```
\declare@shorthand{system}{’}{%
  \textormath{\csname normal@char\string’\endcsname}%
             {\sp\bgroup\prim@s}}
```

When the left quote is made active we need to take care of handling it correctly when it is followed by for instance an open brace token. Therefore we define a shorthand at system level to make it expand to a non-active left quote.

```
\declare@shorthand{system}{‘}{\csname normal@char\string‘\endcsname}
```

`\bbl@prim@s` `\bbl@pr@m@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.

```
\def\bbl@prim@s{%
  \prime\futurelet\@let@token\bbl@pr@m@s}
\def\bbl@if@primes#1#2{%
  \ifx#1\@let@token
    \expandafter\@firstoftwo
  \else\ifx#2\@let@token
    \bbl@afterelse\expandafter\@firstoftwo
  \else
    \bbl@afterfi\expandafter\@secondoftwo
  \fi\fi}
\begingroup
  \catcode‘\^=7  \catcode‘\*=\active  \lccode‘\*=‘\^
  \catcode‘\’=12 \catcode‘\"=\active  \lccode‘\"=‘\’
  \lowercase{%
    \gdef\bbl@pr@m@s{%
      \bbl@if@primes"’%
        \pr@@@s
        {\bbl@if@primes*^\pr@@@t\egroup}}}
\endgroup
```

⟨/core | shorthands⟩

Normally 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.

```
⟨*core⟩
\initiate@active@char{~}
\declare@shorthand{system}{~}{\leavevmode\nobreak\ }
\bbl@activate{~}
```

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

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

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

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

12.6 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.

```
\newcommand\languageattribute[2]{%
```

First check whether the language is known.

```
  \bbl@iflanguage{#1}{%
```

Than process each attribute in the list.

```
    \@for\bbl@attr:=#2\do{%
```

We want to make sure that each attribute is selected only once; therefor 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.

```
      \ifx\bbl@known@attribs\@undefined
        \in@false
      \else
```

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

```
        \@expandtwoargs\in@{,#1-\bbl@attr,}{,\bbl@known@attribs,}%
      \fi
```

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

```
      \ifin@
        \PackageWarning{Babel}{%
          You have more than once selected the attribute
          '\bbl@attr'\MessageBreak for language #1}%
      \else
```

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

```
        \edef\bbl@tempa{%
          \noexpand\bbl@add@list\noexpand\bbl@known@attribs{#1-\bbl@attr}}%
        \bbl@tempa
        \edef\bbl@tempa{#1-\bbl@attr}%
        \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
        {\csname#1@attr@\bbl@attr\endcsname}%
        {\@attrerr{#1}{\bbl@attr}}%
```

```
    \fi
     }%
}}
```

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

```
\@onlypreamble\languageattribute
```

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

```
\newcommand*{\@attrerr}[2]{%
  \PackageError{babel}%
                {The attribute #2 is unknown for language #1.}%
      {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.

```
\def\bbl@declare@ttribute#1#2#3{%
  \bbl@add@list\bbl@attributes{#1-#2}%
```

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}`.

```
  \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.

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

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

```
  \ifx\bbl@known@attribs\@undefined
    \in@false
  \else
```

The we need to check the list of known attributes.

```
    \@expandtwoargs\in@{,#1-#2,}{,\bbl@known@attribs,}%
  \fi
```

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

```
  \ifin@
    \bbl@afterelse#3%
  \else
    \bbl@afterfi#4%
  \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

```
\def\bbl@add@list#1#2{%
  \ifx#1\@undefined
    \def#1{#2}%
  \else
    \ifx#1\@empty
      \def#1{#2}%
    \else
      \edef#1{#1,#2}%
    \fi
  \fi
  }
```

`\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.

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

We first assume the attribute is unknown.

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

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

```
  \@for\bbl@tempb:=#2\do{%
    \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
    \ifin@
```

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

```
      \let\bbl@tempa\@firstoftwo
    \else
    \fi}%
```

Finally we execute `\bbl@tempa`.

```
  \bbl@tempa
}
```

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

```
\def\bbl@clear@ttribs{%
  \ifx\bbl@attributes\@undefined\else
    \@for\bbl@tempa:=\bbl@attributes\do{%
      \expandafter\bbl@clear@ttrib\bbl@tempa.
      }%
    \let\bbl@attributes\@undefined
  \fi
  }
\def\bbl@clear@ttrib#1-#2.{%
  \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
\AtBeginDocument{\bbl@clear@ttribs}
```

12.7 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`).

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

`\babel@beginsave`

```
\def\babel@beginsave{\babel@savecnt\z@}
```

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

```
\newcount\babel@savecnt
\babel@beginsave
```

`\babel@save` The macro `\babel@save`⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to `\originalTeX`[6]. 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.

```
\def\babel@save#1{%
  \expandafter\let\csname babel@\number\babel@savecnt\endcsname #1\relax
  \begingroup
    \toks@\expandafter{\originalTeX \let#1=}%
    \edef\x{\endgroup
      \def\noexpand\originalTeX{\the\toks@ \expandafter\noexpand
         \csname babel@\number\babel@savecnt\endcsname\relax}}%
  \x
  \advance\babel@savecnt\@ne}
```

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

```
\def\babel@savevariable#1{\begingroup
    \toks@\expandafter{\originalTeX #1=}%
    \edef\x{\endgroup
      \def\noexpand\originalTeX{\the\toks@ \the#1\relax}}%
  \x}
```

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

```
\def\bbl@frenchspacing{%
  \ifnum\the\sfcode`\.=\@m
    \let\bbl@nonfrenchspacing\relax
  \else
    \frenchspacing
```

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

```
    \let\bbl@nonfrenchspacing\nonfrenchspacing
  \fi}
\let\bbl@nonfrenchspacing\nonfrenchspacing
```

12.8 Support for extending macros

`\addto` For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro `\addto` is introduced. It takes two arguments, a ⟨*control sequence*⟩ and TEX-code to be added to the ⟨*control sequence*⟩.

If the ⟨*control sequence*⟩ has not been defined before it is defined now.

```
\def\addto#1#2{%
  \ifx#1\@undefined
    \def#1{#2}%
  \else
```

The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow.

```
    \ifx#1\relax
      \def#1{#2}%
    \else
```

Otherwise the replacement text for the ⟨*control sequence*⟩ is expanded and stored in a token register, together with the TEX-code to be added. Finally the ⟨*control sequence*⟩ is *re*defined, using the contents of the token register.

```
      {\toks@\expandafter{#1#2}%
        \xdef#1{\the\toks@}}%
    \fi
  \fi
}
```

12.9 Hyphens

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`[7].

```
\def\bbl@allowhyphens{\nobreak\hskip\z@skip}
\def\bbl@t@one{T1}
\def\allowhyphens{%
  \ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
```

`\babelhyphen` Macros to insert common hyphens.

```
\newcommand\babelnullhyphen{\char\hyphenchar\font}
\DeclareRobustCommand\babelhyphen{%
  \@ifstar{\bbl@hyphen @}{\bbl@hyphen\@empty}}
\def\bbl@hyphen#1#2{%
  \@ifundefined{bbl@hy@#1#2\@empty}%
```

[7] TEX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```
    {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
    {\csname bbl@hy@#1#2\@empty\endcsname}}
```

The following two commands are used to wrap the "hyphen" and set the behaviour of the rest of the word – the version with a single `@` is used when further hyphenation is allowed, while that with `@@` if no more hyphen are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

There should not be a discretionaty after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like "(-suffix)". `\nobreak` is always preceded by `\leavevmode`, in case the shorthand starts a paragraph.

```
\def\bbl@usehyphen#1{%
  \leavevmode
  \ifdim\lastskip>\z@\hbox{#1}\nobreak\else\nobreak#1\fi
  \hskip\z@skip}
\def\bbl@@usehyphen#1{%
  \leavevmode
  \ifdim\lastskip>\z@\hbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
\def\bbl@hyphenchar{%
  \ifnum\hyphenchar\font=\m@ne
    \babelnullhyphen
  \else
    \char\hyphenchar\font
  \fi}
```

Finally, we define the hyphen "types". Their names won't change, so you may use them in `ldf`'s.

```
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
\def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
\def\bbl@hy@nobreak{\bbl@usehyphen{\hbox{\bbl@hyphenchar}\nobreak}}
\def\bbl@hy@@nobreak{\hbox{\bbl@hyphenchar}}
\def\bbl@hy@double{%
  \bbl@usehyphen{%
    \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}%
    \nobreak}}
\def\bbl@hy@@double{%
  \bbl@@usehyphen{%
    \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@empty{\hskip\z@skip}
\def\bbl@hy@@empty{\discretionary{}{}{}}%
```

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

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

12.10 Macros common to a number of languages

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

```
\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
    \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.

```
\def\save@sf@q#1{\leavevmode
  \begingroup
    \edef\@SF{\spacefactor \the\spacefactor}#1\@SF
  \endgroup
}
```

12.11 Making glyphs available

The file babel.dtx[8] 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.

12.12 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, therefor we make it available by lowering the normal open quote character to the baseline.

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

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

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

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

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

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

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

[8] The file described in this section has version number v3.9a-alpha-5, and was last revised on 2012/09/11.

\guillemotleft
\guillemotright

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

```
\ProvideTextCommand{\guillemotleft}{OT1}{%
  \ifmmode
    \ll
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guillemotright}{OT1}{%
  \ifmmode
    \gg
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
  \fi}
```

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

```
\ProvideTextCommandDefault{\guillemotleft}{%
  \UseTextSymbol{OT1}{\guillemotleft}}
\ProvideTextCommandDefault{\guillemotright}{%
  \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft
\guilsinglright

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

```
\ProvideTextCommand{\guilsinglleft}{OT1}{%
  \ifmmode
    <%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guilsinglright}{OT1}{%
  \ifmmode
    >%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
  \fi}
```

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

```
\ProvideTextCommandDefault{\guilsinglleft}{%
  \UseTextSymbol{OT1}{\guilsinglleft}}
\ProvideTextCommandDefault{\guilsinglright}{%
  \UseTextSymbol{OT1}{\guilsinglright}}
```

12.13 Letters

\ij
\IJ

The dutch language uses the letter 'ij'. It is available in T1 encoded fonts, but not in the OT1 encoded fonts. Therefor we fake it for the OT1 encoding.

```
\DeclareTextCommand{\ij}{OT1}{%
  i\kern-0.02em\bbl@allowhyphens j}
\DeclareTextCommand{\IJ}{OT1}{%
  I\kern-0.02em\bbl@allowhyphens J}
\DeclareTextCommand{\ij}{T1}{\char188}
\DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
\ProvideTextCommandDefault{\ij}{%
  \UseTextSymbol{OT1}{\ij}}
\ProvideTextCommandDefault{\IJ}{%
  \UseTextSymbol{OT1}{\IJ}}
```

\dj
\DJ

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

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

```
\def\crrtic@{\hrule height0.1ex width0.3em}
\def\crttic@{\hrule height0.1ex width0.33em}
%
\def\ddj@{%
  \setbox0\hbox{d}\dimen@=\ht0
  \advance\dimen@1ex
  \dimen@.45\dimen@
  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
  \advance\dimen@ii.5ex
  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
\def\DDJ@{%
  \setbox0\hbox{D}\dimen@=.55\ht0
  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
  \advance\dimen@ii.15ex %            correction for the dash position
  \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
  \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
%
\DeclareTextCommand{\dj}{OT1}{\ddj@ d}
\DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
\ProvideTextCommandDefault{\dj}{%
  \UseTextSymbol{OT1}{\dj}}
\ProvideTextCommandDefault{\DJ}{%
  \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. Therefor we make it available here.

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

12.14 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode.

\glq The 'german' single quotes.

\grq

```
\ProvideTextCommand{\glq}{OT1}{%
  \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
\ProvideTextCommand{\glq}{T1}{%
  \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
\ProvideTextCommandDefault{\glq}{\UseTextSymbol{OT1}\glq}
```

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

```
\ProvideTextCommand{\grq}{T1}{%
  \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
\ProvideTextCommand{\grq}{OT1}{%
  \save@sf@q{\kern-.0125em%
  \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
  \kern.07em\relax}}
\ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The 'german' double quotes.

\grqq

```
\ProvideTextCommand{\glqq}{OT1}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\ProvideTextCommand{\glqq}{T1}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\ProvideTextCommandDefault{\glqq}{\UseTextSymbol{OT1}\glqq}
```

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

```
\ProvideTextCommand{\grqq}{T1}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\ProvideTextCommand{\grqq}{OT1}{%
  \save@sf@q{\kern-.07em%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
  \kern.07em\relax}}
\ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq The 'french' single guillemets.

\frq

```
\ProvideTextCommand{\flq}{OT1}{%
  \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
\ProvideTextCommand{\flq}{T1}{%
  \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
\ProvideTextCommandDefault{\flq}{\UseTextSymbol{OT1}\flq}
```

```
\ProvideTextCommand{\frq}{OT1}{%
  \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
\ProvideTextCommand{\frq}{T1}{%
  \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
\ProvideTextCommandDefault{\frq}{\UseTextSymbol{OT1}\frq}
```

\flqq The 'french' double guillemets.

\frqq

```
\ProvideTextCommand{\flqq}{OT1}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\ProvideTextCommand{\flqq}{T1}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\ProvideTextCommandDefault{\flqq}{\UseTextSymbol{OT1}\flqq}
```

```
\ProvideTextCommand{\frqq}{OT1}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
\ProvideTextCommand{\frqq}{T1}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
\ProvideTextCommandDefault{\frqq}{\UseTextSymbol{OT1}\frqq}
```

12.15 Umlauts and trema's

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
\umlautlow

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

```
\def\umlauthigh{%
  \def\bbl@umlauta##1{\leavevmode\bgroup%
      \expandafter\accent\csname\f@encoding dqpos\endcsname
      ##1\bbl@allowhyphens\egroup}%
  \let\bbl@umlaute\bbl@umlauta}
\def\umlautlow{%
  \def\bbl@umlauta{\protect\lower@umlaut}}
\def\umlautelow{%
  \def\bbl@umlaute{\protect\lower@umlaut}}
\umlauthigh
```

\lower@umlaut

The command `\lower@umlaut` is used to position the `\"` closer the the letter.

We want the umlaut character lowered, nearer to the letter. To do this we need an extra ⟨*dimen*⟩ register.

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

```
\def\lower@umlaut#1{%
```

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.

```
  \leavevmode\bgroup
    \U@D 1ex%
```

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.)

```
    {\setbox\z@\hbox{%
      \expandafter\char\csname\f@encoding dqpos\endcsname}%
      \dimen@ -.45ex\advance\dimen@\ht\z@
```

If the new x-height is too low, it is not changed.

```
      \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
```

Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```
    \expandafter\accent\csname\f@encoding dqpos\endcsname
    \fontdimen5\font\U@D #1%
  \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. Therefor these declarations are postponed until the beginning of the document.

```
\AtBeginDocument{%
  \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
  \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
  \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
  \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
  \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
  \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
  \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
  \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
  \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
  \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
  \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
}
```

12.16 The redefinition of the style commands

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

```
{\def\format{lplain}
\ifx\fmtname\format
\else
```

```
  \def\format{LaTeX2e}
  \ifx\fmtname\format
  \else
    \aftergroup\endinput
  \fi
\fi}
```

Now that we're sure that the code is seen by LaTeX only, we have to find out what the main (primary) document style is because we want to redefine some macros. This is only necessary for releases of LaTeX dated before December 1991. Therefor this part of the code can optionally be included in `babel.def` by specifying the `docstrip` option `names`.

```
⟨*names⟩
```

The standard styles can be distinguished by checking whether some macros are defined. In table 1 an overview is given of the macros that can be used for this purpose.

article	:	both the `\chapter` and `\opening` macros are undefined
report and book	:	the `\chapter` macro is defined and the `\opening` is undefined
letter	:	the `\chapter` macro is undefined and the `\opening` is defined

Table 1: How to determine the main document style

The macros that have to be redefined for the `report` and `book` document styles happen to be the same, so there is no need to distinguish between those two styles.

`\doc@style` First a parameter `\doc@style` is defined to identify the current document style. This parameter might have been defined by a document style that already uses macros instead of hard-wired texts, such as `artikel1.sty` [6], so the existence of `\doc@style` is checked. If this macro is undefined, i. e., if the document style is unknown and could therefore contain hard-wired texts, `\doc@style` is defined to the default value '0'.

```
\ifx\@undefined\doc@style
  \def\doc@style{0}%
```

This parameter is defined in the following `if` construction (see table 1):

```
  \ifx\@undefined\opening
    \ifx\@undefined\chapter
      \def\doc@style{1}%
    \else
      \def\doc@style{2}%
    \fi
  \else
    \def\doc@style{3}%
```

```
  \fi%
\fi%
```

12.16.1 Redefinition of macros

Now here comes the real work: we start to redefine things and replace hard-wired texts by macros. These redefinitions should be carried out conditionally, in case it has already been done.

For the `figure` and `table` environments we have in all styles:

```
\@ifundefined{figurename}{\def\fnum@figure{\figurename{} \thefigure}}{}
\@ifundefined{tablename}{\def\fnum@table{\tablename{} \thetable}}{}
```

The rest of the macros have to be treated differently for each style. When `\doc@style` still has its default value nothing needs to be done.

```
\ifcase \doc@style\relax
\or
```

This means that `babel.def` is read after the `article` style, where no `\chapter` and `\opening` commands are defined[9].

First we have the `\tableofcontents`, `\listoffigures` and `\listoftables`:

```
\@ifundefined{contentsname}%
    {\def\tableofcontents{\section*{\contentsname\@mkboth
          {\uppercase{\contentsname}}{\uppercase{\contentsname}}}%
      \@starttoc{toc}}}{}

\@ifundefined{listfigurename}%
    {\def\listoffigures{\section*{\listfigurename\@mkboth
          {\uppercase{\listfigurename}}{\uppercase{\listfigurename}}}%
     \@starttoc{lof}}}{}

\@ifundefined{listtablename}%
    {\def\listoftables{\section*{\listtablename\@mkboth
          {\uppercase{\listtablename}}{\uppercase{\listtablename}}}%
      \@starttoc{lot}}}{}
```

Then the `\thebibliography` and `\theindex` environments.

```
\@ifundefined{refname}%
    {\def\thebibliography#1{\section*{\refname
      \@mkboth{\uppercase{\refname}}{\uppercase{\refname}}}%
      \list{[\arabic{enumi}]}{\settowidth\labelwidth{[#1]}%
        \leftmargin\labelwidth
        \advance\leftmargin\labelsep
        \usecounter{enumi}}%
        \def\newblock{\hskip.11em plus.33em minus.07em}%
        \sloppy\clubpenalty4000\widowpenalty\clubpenalty
        \sfcode`\.=1000\relax}}{}

```

[9] A fact that was pointed out to me by Nico Poppelier and was already used in Piet van Oostrum's document style option `nl`.

```
\@ifundefined{indexname}%
    {\def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
     \columnseprule \z@
     \columnsep 35pt\twocolumn[\section*{\indexname}]%
       \@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}%
       \thispagestyle{plain}%
       \parskip\z@ plus.3pt\parindent\z@\let\item\@idxitem}}{}
```

The `abstract` environment:

```
\@ifundefined{abstractname}%
    {\def\abstract{\if@twocolumn
    \section*{\abstractname}%
    \else \small
    \begin{center}%
    {\bf \abstractname\vspace{-.5em}\vspace{\z@}}%
    \end{center}%
    \quotation
    \fi}}{}
```

And last but not least, the macro `\part`:

```
\@ifundefined{partname}%
{\def\@part[#1]#2{\ifnum \c@secnumdepth >\m@ne
        \refstepcounter{part}%
        \addcontentsline{toc}{part}{\thepart
        \hspace{1em}#1}\else
      \addcontentsline{toc}{part}{#1}\fi
   {\parindent\z@ \raggedright
    \ifnum \c@secnumdepth >\m@ne
      \Large \bf \partname{} \thepart
      \par \nobreak
    \fi
    \huge \bf
    #2\markboth{}{}\par}%
    \nobreak
    \vskip 3ex\@afterheading}%
}{}
```

This is all that needs to be done for the `article` style.

```
\or
```

The next case is formed by the two styles `book` and `report`. Basically we have to do the same as for the `article` style, except now we must also change the `\chapter` command.

The tables of contents, figures and tables:

```
\@ifundefined{contentsname}%
    {\def\tableofcontents{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \fi\chapter*{\contentsname\@mkboth
          {\uppercase{\contentsname}}{\uppercase{\contentsname}}}%
      \@starttoc{toc}%
```

```
      \csname if@restonecol\endcsname\twocolumn
      \csname fi\endcsname}}{}

\@ifundefined{listfigurename}%
    {\def\listoffigures{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \fi\chapter*{\listfigurename\@mkboth
          {\uppercase{\listfigurename}}{\uppercase{\listfigurename}}}%
      \@starttoc{lof}%
      \csname if@restonecol\endcsname\twocolumn
      \csname fi\endcsname}}{}

\@ifundefined{listtablename}%
    {\def\listoftables{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \fi\chapter*{\listtablename\@mkboth
          {\uppercase{\listtablename}}{\uppercase{\listtablename}}}%
      \@starttoc{lot}%
      \csname if@restonecol\endcsname\twocolumn
      \csname fi\endcsname}}{}
```

Again, the `bibliography` and `index` environments; notice that in this case we use `\bibname` instead of `\refname` as in the definitions for the `article` style. The reason for this is that in the `article` document style the term 'References' is used in the definition of `\thebibliography`. In the `report` and `book` document styles the term 'Bibliography' is used.

```
\@ifundefined{bibname}%
    {\def\thebibliography#1{\chapter*{\bibname
     \@mkboth{\uppercase{\bibname}}{\uppercase{\bibname}}}%
     \list{[\arabic{enumi}]}{\settowidth\labelwidth{[#1]}%
     \leftmargin\labelwidth \advance\leftmargin\labelsep
     \usecounter{enumi}}%
     \def\newblock{\hskip.11em plus.33em minus.07em}%
     \sloppy\clubpenalty4000\widowpenalty\clubpenalty
     \sfcode`\.=1000\relax}}{}

\@ifundefined{indexname}%
    {\def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
    \columnseprule \z@
    \columnsep 35pt\twocolumn[\@makeschapterhead{\indexname}]%
      \@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}%
    \thispagestyle{plain}%
    \parskip\z@ plus.3pt\parindent\z@ \let\item\@idxitem}}{}
```

Here is the `abstract` environment:

```
\@ifundefined{abstractname}%
    {\def\abstract{\titlepage
    \null\vfil
    \begin{center}%
    {\bf \abstractname}%
```

```
    \end{center}}}{}
```

And last but not least the `\chapter`, `\appendix` and `\part` macros.

```
\@ifundefined{chaptername}{\def\@chapapp{\chaptername}}{}
%
\@ifundefined{appendixname}%
    {\def\appendix{\par
      \setcounter{chapter}{0}%
      \setcounter{section}{0}%
      \def\@chapapp{\appendixname}%
      \def\thechapter{\Alph{chapter}}}}{}
%
\@ifundefined{partname}%
    {\def\@part[#1]#2{\ifnum \c@secnumdepth >-2\relax
            \refstepcounter{part}%
            \addcontentsline{toc}{part}{\thepart
            \hspace{1em}#1}\else
            \addcontentsline{toc}{part}{#1}\fi
       \markboth{}{}%
       {\centering
        \ifnum \c@secnumdepth >-2\relax
          \huge\bf \partname{} \thepart
        \par
        \vskip 20pt \fi
        \Huge \bf
        #1\par}\@endpart}}{}%
```

```
\or
```

Now we address the case where `babel.def` is read after the `letter` style. The `letter` document style defines the macro `\opening` and some other macros that are specific to `letter`. This means that we have to redefine other macros, compared to the previous two cases.

First two macros for the material at the end of a letter, the `\cc` and `\encl` macros.

```
\@ifundefined{ccname}%
    {\def\cc#1{\par\noindent
     \parbox[t]{\textwidth}%
     {\@hangfrom{\rm \ccname : }\ignorespaces #1\strut}\par}}{}

\@ifundefined{enclname}%
    {\def\encl#1{\par\noindent
     \parbox[t]{\textwidth}%
     {\@hangfrom{\rm \enclname : }\ignorespaces #1\strut}\par}}{}
```

The last thing we have to do here is to redefine the `headings` pagestyle:

```
\@ifundefined{headtoname}%
    {\def\ps@headings{%
        \def\@oddhead{\sl \headtoname{} \ignorespaces\toname \hfil
                      \@date \hfil \pagename{} \thepage}%
        \def\@oddfoot{}}}{}
```

This was the last of the four standard document styles, so if `\doc@style` has another value we do nothing and just close the `if` construction.

```
\fi
```

Here ends the code that can be optionally included when a version of LaTeX is in use that is dated *before* December 1991.

```
⟨/names⟩
⟨/core⟩
```

12.17 Cross referencing macros

The LaTeX book states:

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

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

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

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

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

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

```
⟨*core | shorthands⟩
\def\bbl@redefine#1{%
  \edef\bbl@tempa{\expandafter\@gobble\string#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1
  \expandafter\def\csname\bbl@tempa\endcsname}
```

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

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

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

```
\def\bbl@redefine@long#1{%
  \edef\bbl@tempa{\expandafter\@gobble\string#1}%
```

```
  \expandafter\let\csname org@\bbl@tempa\endcsname#1
  \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
\@onlypreamble\bbl@redefine@long
```

`\bbl@redefinerobust` 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.

```
\def\bbl@redefinerobust#1{%
  \edef\bbl@tempa{\expandafter\@gobble\string#1}%
  \expandafter\ifx\csname \bbl@tempa\space\endcsname\relax
    \expandafter\let\csname org@\bbl@tempa\endcsname#1
    \expandafter\edef\csname\bbl@tempa\endcsname{\noexpand\protect
      \expandafter\noexpand\csname\bbl@tempa\space\endcsname}%
  \else
    \expandafter\let\csname org@\bbl@tempa\expandafter\endcsname
                     \csname\bbl@tempa\space\endcsname
  \fi
```

The result of the code above is that the command that is being redefined is always robust afterwards. Therefor all we need to do now is define `\foo␣`.

```
  \expandafter\def\csname\bbl@tempa\space\endcsname}
```

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

```
\@onlypreamble\bbl@redefinerobust
```

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

```
%\bbl@redefine\newlabel#1#2{%
%  \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

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

```
\ifx\bbl@opt@safe\@empty\else
  \def\@newl@bel#1#2#3{%
```

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.

```
    {%
      \@safe@activestrue
      \@ifundefined{#1@#2}%
        \relax
        {%
          \gdef \@multiplelabels {%
            \@latex@warning@no@line{There were multiply-defined labels}}%
          \@latex@warning@no@line{Label ‘#2’ multiply defined}%
        }%
      \global\@namedef{#1@#2}{#3}%
      }%
    }
```

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

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

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

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

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

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

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

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

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

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

We do the same for \bbl@tempb.

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

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

```
  \ifx\bbl@tempa\bbl@tempb
  \else
    \@tempswatrue
  \fi}
\fi
```

\ref
\pageref

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

```
\@expandtwoargs\in@{R}\bbl@opt@safe
\ifin@
  \bbl@redefinerobust\ref#1{%
    \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
  \bbl@redefinerobust\pageref#1{%
```

```
    \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
\else
  \let\org@ref\ref
  \let\org@pageref\pageref
\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.

```
\@expandtwoargs\in@{B}\bbl@opt@safe
\ifin@
  \bbl@redefine\@citex[#1]#2{%
    \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
    \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.

```
  \AtBeginDocument{%
    \@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). !!!! 2012/08/03 But many things could happen between the value is saved and it's redefined. So, first restore and then redefine To be further investigated. !!!!Recent versions of natbib change dynamically citex, so PR4087 doesn't seem fixable in a simple way. Just load natbib before.

```
      \let\@citex\org@@citex
      \bbl@redefine\@citex[#1][#2]#3{%
        \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
        \org@@citex[#1][#2]{\@tempa}}%
    }{}}
```

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

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

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

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

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

```
\bbl@redefine\bibcite{%
```

We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
  \bbl@cite@choice
  \bibcite}
```

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

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

`\bbl@cite@choice` The macro `\bbl@cite@choice` determines which definition of `\bibcite` is needed.

```
\def\bbl@cite@choice{%
```

First we give `\bibcite` its default definition.

```
  \global\let\bibcite\bbl@bibcite
```

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

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

For `cite` we do the same.

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

Make sure this only happens once.

```
  \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.

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

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

```
\bbl@redefine\@bibitem#1{%
  \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
\else
  \let\org@nocite\nocite
  \let\org@@citex\@citex
  \let\org@bibcite\bibcite
  \let\org@@bibitem\@bibitem
\fi
```

12.18 marks

\markright Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of \markright and \markboth somewhat.

```
\bbl@redefine\markright#1{%
```

First of all we temporarily store the language switching command, using an expanded definition in order to get the current value of \languagename.

```
  \edef\bbl@tempb{\noexpand\protect
    \noexpand\foreignlanguage{\languagename}}%
```

Then, we check whether the argument is empty; if it is, we just make sure the scratch token register is empty.

```
  \def\bbl@arg{#1}%
  \ifx\bbl@arg\@empty
    \toks@{}%
  \else
```

Next, we store the argument to \markright in the scratch token register, together with the expansion of \bbl@tempb (containing the language switching command) as defined before. This way these commands will not be expanded by using \edef later on, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while \@safe@activestrue is in effect.

```
    \expandafter\toks@\expandafter{%
              \bbl@tempb{\protect\bbl@restore@actives#1}}%
  \fi
```

Then we define a temporary control sequence using \edef.

```
  \edef\bbl@tempa{%
```

When \bbl@tempa is executed, only \languagename will be expanded, because of the way the token register was filled.

```
    \noexpand\org@markright{\the\toks@}}%
  \bbl@tempa
}
```

\markboth \@mkboth 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. Therefor we need to check whether \@mkboth has already been set. If so we neeed to do that again with the new definition of \makrboth.

```
\ifx\@mkboth\markboth
  \def\bbl@tempc{\let\@mkboth\markboth}
\else
  \def\bbl@tempc{}
\fi
```

Now we can start the new definition of \markboth

```
\bbl@redefine\markboth#1#2{%
  \edef\bbl@tempb{\noexpand\protect
    \noexpand\foreignlanguage{\languagename}}%
  \def\bbl@arg{#1}%
  \ifx\bbl@arg\@empty
    \toks@{}%
  \else
   \expandafter\toks@\expandafter{%
             \bbl@tempb{\protect\bbl@restore@actives#1}}%
  \fi
  \def\bbl@arg{#2}%
  \ifx\bbl@arg\@empty
    \toks8{}%
  \else
    \expandafter\toks8\expandafter{%
             \bbl@tempb{\protect\bbl@restore@actives#2}}%
  \fi
  \edef\bbl@tempa{%
    \noexpand\org@markboth{\the\toks@}{\the\toks8}}%
  \bbl@tempa
}
```

and copy it to \@mkboth if necesary.

```
\bbl@tempc
⟨/core | shorthands⟩
```

12.19 Multiencoding strings

!!!!! Tentative. To be documented

```
⟨*core⟩
%^^A This single flag must be changed by multiple flags taking into
%^^A account the language (and the group?)
\newif\ifbbl@scdone
\bbl@scdonefalse
\def\bbl@scparse#1{%
  \ifx\@empty#1\else
    \ifx<#1\noexpand\@nil\noexpand\bbl@tempa{from}%
    \else\ifx>#1\noexpand\@nil\noexpand\bbl@tempa{to}%
    \else#1%
    \fi\fi
    \expandafter\bbl@scparse
  \fi}
\def\StartBabelCommands{%
%^^A Do only if #1 is the current option, or not?
%^^A Error if #3#1 is empty or undefined
%^^A Reset #3#1
  \begingroup
%^^A   We make sure strings contain actual letters in the range 128-255,
```

```
%^^A   not active characters
  \@tempcnta="7F
  \def\bbl@tempa{%
    \ifnum\@tempcnta>"FF\else
      \catcode\@tempcnta=11
      \advance\@tempcnta\@ne
      \expandafter\bbl@tempa
    \fi}%
  \let\StartBabelCommands\bbl@startcmds
  \StartBabelCommands}
\def\bbl@startcmds{%
  \@ifstar{\bbl@startcmds@i\@nil}{\bbl@startcmds@i}}
%
%^^A =auto con \LastDeclaredEncoding ???
%
\def\bbl@startcmds@i#1#2#3{%
  \babel@scstop
  \let\babel@scstop\relax
%^^A TODO: If there is no string=, do nothing, and perserve #3#1
%^^A if there is string=, reset #3#1
%^^A  Parse the encoding info to get the label, from (|<|) and to (|>|)
%^^A  parts. Most of the word is done by |\bbl@scparse| above.
  \let\bbl@sc@from\@empty
  \let\bbl@sc@to\@empty
  \edef\bbl@the@group{\zap@space#3 \@empty}%
  \ifx\@nil#1%
    \edef\bbl@the@lang{\zap@space#2 \@empty}%
    \def\bbl@sc@label{generic}%
  \else
    \edef\bbl@the@lang{\zap@space#1 \@empty}%
    \protected@edef\bbl@tempb{\noexpand\bbl@tempa{label}\bbl@scparse#2\@empty}%
    \def\bbl@tempa##1##2\@nil{\@namedef{bbl@sc@##1}{##2}}%
    \bbl@tempb\@nil
  \fi
  % Select the behaviour: encoded, ENC, or nothing
\tracingmacros2
  \ifx\bbl@opt@strings\relax % set by DeclOpt string=encoded
    \let\SetBabelString\bbl@setstring
    \ifx\@nil#1%
      \def\bbl@stringdef##1##2{%
         \@dec@text@cmd\gdef##1?{##2}%
         \global\let##1##1}%
    \else
      \def\bbl@stringdef##1##2{%
        \@for\bbl@tempa:=\bbl@sc@to\do{%
          \@ifundefined{T@\bbl@tempa}{}%
            {\@dec@text@cmd\gdef##1\bbl@tempa{##2}%
             \global\let##1##1}}}%
    \fi
    \babel@scstart
```

```
  \else
    \ifx\bbl@opt@strings\@nnil  % Not optimal -- too late
      \in@false
    \else\ifx\@nil#1%
      \ifbbl@scdone
        \in@false
      \else
      %  And warning if opt@strings is set -- No strings xxx for lang zzz
        \in@true
      \fi
    \else
      \@expandtwoargs\in@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@to,}%
    \fi\fi
    \ifin@
      \bbl@scdonetrue % To be replaced by flags considering language, etc.
      \let\SetBabelString\bbl@setstring
      \let\bbl@stringdef\gdef
      \babel@scstart
    \else
      \let\SetBabelString\@gobbletwo
      \babel@scskip
    \fi
  \fi}
\def\EndBabelCommands{\babel@scstop\endgroup}
%
%^^A set stringdef = \gdef or \DeclareTextCommand{com}{enc} or gobbletwo
%
\def\bbl@setstring#1#2{%
  \@for\bbl@tempc:=\bbl@the@lang\do{%
  % empties !!!
    \edef\bbl@tempa{\bbl@tempc\expandafter\@gobble\string#1}%
    \edef\bbl@tempc{\bbl@the@group\bbl@tempc}%
    \@ifundefined{\bbl@tempc}{\@namedef{\bbl@tempc}{}}{}%
    \@ifundefined{\bbl@tempa}%
      {\toks@\expandafter\expandafter\expandafter{\csname\bbl@tempc\endcsname}%
       \expandafter\xdef\csname\bbl@tempc\endcsname{%
         \the\toks@
         \def\noexpand#1{%
           \expandafter\noexpand\csname\bbl@tempa\endcsname}}}%
      {}%
    \babel@scprocess\bbl@tempb{#2}%
    \expandafter\bbl@stringdef
      \csname\bbl@tempa\expandafter\endcsname\expandafter{\bbl@tempb}}}
%
\let\babel@scstart\relax
\let\babel@scskip\relax
\let\babel@scstop\relax
\let\babel@scprocess\def
%
\ifx\XeTeXinputencoding\@undefined\else
```

```
  \def\babel@scstart{%
    \ifx\bbl@sc@from\@empty
      \XeTeXinputencoding"bytes"%
    \else
      \XeTeXinputencoding"\bbl@sc@from"%
    \fi
    \def\babel@scstop{\XeTeXinputencoding"utf8"}}%
  \def\babel@scskip{%
    \XeTeXinputencoding"bytes"%
    \def\babel@scstop{\XeTeXinputencoding"utf8"}}%
\fi
%
\ifx\directlua\@undefined\else
  \directlua{%
    Babel = {}
    function Babel.bytes(line)
      return line:gsub("(.)",
        function (chr) return unicode.utf8.char(string.byte(chr)) end)
    end
    function Babel.begin_process_input()
      if luatexbase and luatexbase.add_to_callback then
        luatexbase.add_to_callback('process_input_buffer',Babel.bytes,'Babel.bytes')
      else
        Babel.callback = callback.find('process_input_buffer')
        callback.register('process_input_buffer',Babel.bytes)
      end
    end
    function Babel.end_process_input ()
      if luatexbase and luatexbase.remove_from_callback then
        luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
      else
        callback.register('process_input_buffer',Babel.callback)
      end
    end
  }
  \def\babel@scstart{%
    \def\bbl@tempa{utf8}%
    \ifx\bbl@tempa\bbl@sc@from\else
      \directlua{Babel.begin_process_input()}%
      \def\babel@scstop{%
        \directlua{Babel.end_process_input()}}%
    \fi}
    \let\babel@scskip\babel@scstart
\fi
⟨/core⟩
```

12.20 Encoding issues (part 2)

\TeX
\LaTeX

Because documents may use font encodings other than one of the latin encodings, we make sure that the logos of TEX and LATEX always come out in the right encoding.

```
⟨*core⟩
\bbl@redefine\TeX{\textlatin{\org@TeX}}
\bbl@redefine\LaTeX{\textlatin{\org@LaTeX}}
⟨/core⟩
```

12.21 Preventing clashes with other packages

12.21.1 ifthen

\ifthenelse

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

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

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

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

```
⟨*package⟩
\@expandtwoargs\in@{R}\bbl@opt@safe
\ifin@
  \AtBeginDocument{%
    \@ifpackageloaded{ifthen}{%
```

Then we can redefine \ifthenelse:

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

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

```
        \let\bbl@tempa\pageref
        \let\pageref\org@pageref
        \let\bbl@tempb\ref
        \let\ref\org@ref
```

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

```
        \@safe@activestrue
        \org@ifthenelse{#1}{%
```

```
          \let\pageref\bbl@tempa
          \let\ref\bbl@tempb
          \@safe@activesfalse
          #2}{%
          \let\pageref\bbl@tempa
          \let\ref\bbl@tempb
          \@safe@activesfalse
          #3}%
        }%
```

When the package wasn't loaded we do nothing.

```
      }{}%
    }
```

12.21.2 varioref

\@@vpageref
\vrefpagenum
\Ref

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.

```
  \AtBeginDocument{%
    \@ifpackageloaded{varioref}{%
      \bbl@redefine\@@vpageref#1[#2]#3{%
        \@safe@activestrue
        \org@@@vpageref{#1}[#2]{#3}%
        \@safe@activesfalse}%
```

The same needs to happen for \vrefpagenum.

```
      \bbl@redefine\vrefpagenum#1#2{%
        \@safe@activestrue
        \org@vrefpagenum{#1}{#2}%
        \@safe@activesfalse}%
```

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

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

12.21.3 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. Therefor we need to *reload* the package when the ':' is an active character.

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

```
\AtBeginDocument{%
  \@ifpackageloaded{hhline}%
```

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

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

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

```
       \makeatletter
       \def\@currname{hhline}\input{hhline.sty}\makeatother
     \fi}%
    {}}
```

12.21.4 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 removed for the moment because hyperref is expecting it, .

```
\AtBeginDocument{%
  \@ifundefined{pdfstringdefDisableCommands}%
    {}%
    {\pdfstringdefDisableCommands{%
       \languageshorthands{system}}%
    }%
}
```

12.21.5 General

\FOREIGNLANGUAGE

The package fancyhdr treats the running head and fout 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.

```
\DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
  \lowercase{\foreignlanguage{#1}}}
⟨/package⟩
```

\nfss@catcodes

LaTeX's font selection scheme sometimes wants to read font definition files in the middle of processing the document. In order to guard against any characters having the wrong \catcodes it always calls \nfss@catcodes before loading a file. Unfortunately, the characters " and ' are not dealt with. Therefor we have to add them until LaTeX does that itself. !!!! Well, Latex already does that itself, but : should be added, too, and perhaps others...

```
⟨*core | shorthands⟩
\ifx\nfss@catcodes\@undefined
\else
  \addto\nfss@catcodes{%
    \@makeother\'%
    \@makeother\"%
    }
\fi

⟨/core | shorthands⟩
```

13 Local Language Configuration

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

```
⟨*core⟩
```

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

```
\ifx\loadlocalcfg\@undefined
  \@ifpackagewith{babel}{noconfig}%
    {\let\loadlocalcfg\@gobble}%
    {\def\loadlocalcfg#1{%
      \InputIfFileExists{#1.cfg}%
        {\typeout{***********************************^^J%
                        * Local config file #1.cfg used^^J%
                        *}}%
        \@empty}}
\fi
```

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

```
\ifx\@unexpandable@protect\@undefined
  \def\@unexpandable@protect{\noexpand\protect\noexpand}
  \long\def \protected@write#1#2#3{%
        \begingroup
         \let\thepage\relax
         #2%
         \let\protect\@unexpandable@protect
         \edef\reserved@a{\write#1{#3}}%
         \reserved@a
        \endgroup
        \if@nobreak\ifvmode\nobreak\fi\fi
  }
\fi
⟨/core⟩
```

14 Driver files for the documented source code

Since babel version 3.4 all source files that are part of the babel system can be typeset separately. But to typeset them all in one document, the file `babel.drv` can be used. If you only want the information on how to use the babel system and what goodies are provided by the language-specific files, you can run the file `user.drv` through LaTeX to get a user guide.

```
⟨*driver⟩
\documentclass{ltxdoc}
\usepackage{url,t1enc,supertabular}
\usepackage[icelandic,english]{babel}
\DoNotIndex{\!,\',\,,\.,\-,\:,\;,\?,\/,\^,\`,\@M}
\DoNotIndex{\@,\@ne,\@m,\@afterheading,\@date,\@endpart}
\DoNotIndex{\@hangfrom,\@idxitem,\@makeschapterhead,\@mkboth}
\DoNotIndex{\@oddfoot,\@oddhead,\@restonecolfalse,\@restonecoltrue}
\DoNotIndex{\@starttoc,\@unused}
\DoNotIndex{\accent,\active}
\DoNotIndex{\addcontentsline,\advance,\Alph,\arabic}
\DoNotIndex{\baselineskip,\begin,\begingroup,\bf,\box,\c@secnumdepth}
\DoNotIndex{\catcode,\centering,\char,\chardef,\clubpenalty}
\DoNotIndex{\columnsep,\columnseprule,\crcr,\csname}
\DoNotIndex{\day,\def,\dimen,\discretionary,\divide,\dp,\do}
\DoNotIndex{\edef,\else,\@empty,\end,\endgroup,\endcsname,\endinput}
\DoNotIndex{\errhelp,\errmessage,\expandafter,\fi,\filedate}
\DoNotIndex{\fileversion,\fmtname,\fnum@figure,\fnum@table,\fontdimen}
\DoNotIndex{\gdef,\global}
\DoNotIndex{\hbox,\hidewidth,\hfil,\hskip,\hspace,\ht,\Huge,\huge}
\DoNotIndex{\ialign,\if@twocolumn,\ifcase,\ifcat,\ifhmode,\ifmmode}
\DoNotIndex{\ifnum,\ifx,\immediate,\ignorespaces,\input,\item}
\DoNotIndex{\kern}
\DoNotIndex{\labelsep,\Large,\large,\labelwidth,\lccode,\leftmargin}
\DoNotIndex{\lineskip,\leavevmode,\let,\list,\ll,\long,\lower}
\DoNotIndex{\m@ne,\mathchar,\mathaccent,\markboth,\month,\multiply}
\DoNotIndex{\newblock,\newbox,\newcount,\newdimen,\newif,\newwrite}
\DoNotIndex{\nobreak,\noexpand,\noindent,\null,\number}
\DoNotIndex{\onecolumn,\or}
\DoNotIndex{\p@,par, \parbox,\parindent,\parskip,\penalty}
\DoNotIndex{\protect,\ps@headings}
\DoNotIndex{\quotation}
\DoNotIndex{\raggedright,\raise,\refstepcounter,\relax,\rm,\setbox}
\DoNotIndex{\section,\setcounter,\settowidth,\scriptscriptstyle}
\DoNotIndex{\sfcode,\sl,\sloppy,\small,\space,\spacefactor,\strut}
\DoNotIndex{\string}
\DoNotIndex{\textwidth,\the,\thechapter,\thefigure,\thepage,\thepart}
\DoNotIndex{\thetable,\thispagestyle,\titlepage,\tracingmacros}
\DoNotIndex{\tw@,\twocolumn,\typeout,\uppercase,\usecounter}
\DoNotIndex{\vbox,\vfil,\vskip,\vspace,\vss}
\DoNotIndex{\widowpenalty,\write,\xdef,\year,\z@,\z@skip}
```

Here \dlqq is defined so that an example of "' can be given.

```
\makeatletter
\gdef\dlqq{{\setbox\tw@=\hbox{,}\setbox\z@=\hbox{''}%
  \dimen\z@=\ht\z@ \advance\dimen\z@-\ht\tw@
  \setbox\z@=\hbox{\lower\dimen\z@\box\z@}\ht\z@=\ht\tw@
  \dp\z@=\dp\tw@ \box\z@\kern-.04em}}
```

The code lines are numbered within sections,

```
⟨*!user⟩
\@addtoreset{CodelineNo}{section}
\renewcommand\theCodelineNo{%
  \reset@font\scriptsize\thesection.\arabic{CodelineNo}}
```

which should also be visible in the index; hence this redefinition of a macro from doc.sty.

```
\renewcommand\codeline@wrindex[1]{\if@filesw
        \immediate\write\@indexfile
            {\string\indexentry{#1}%
            {\number\c@section.\number\c@CodelineNo}}\fi}
```

The glossary environment is used or the change log, but its definition needs changing for this document.

```
\renewenvironment{theglossary}{%
    \glossary@prologue%
    \GlossaryParms \let\item\@idxitem \ignorespaces}%
   {}
⟨/!user⟩
\makeatother
```

A few shorthands used in the documentation

```
\font\manual=logo10 % font used for the METAFONT logo, etc.
\newcommand*\MF{{\manual META}\-{\manual FONT}}
\newcommand*\TeXhax{\TeX hax}
\newcommand*\babel{\textsf{babel}}
\newcommand*\Babel{\textsf{Babel}}
\newcommand*\m[1]{\mbox{$\langle$\it#1\/$\rangle$}}
\newcommand*\langvar{\m{lang}}
```

Some more definitions needed in the documentation.

```
%\newcommand*\note[1]{\textbf{#1}}
\newcommand*\note[1]{}
\newcommand*\bsl{\protect\bslash}
\newcommand*\Lopt[1]{\textsf{#1}}
\newcommand*\Lenv[1]{\textsf{#1}}
\newcommand*\file[1]{\texttt{#1}}
\newcommand*\cls[1]{\texttt{#1}}
\newcommand*\pkg[1]{\texttt{#1}}
\newcommand*\langdeffile[1]{%
⟨-user⟩   \clearpage
  \DocInput{#1}}
```

When a full index should be generated uncomment the line with `\EnableCrossrefs`. Beware, processing may take some time. Use `\DisableCrossrefs` when the index is ready.

```
% \EnableCrossrefs
\DisableCrossrefs
```

Inlude the change log.

```
⟨-user⟩\RecordChanges
```

The index should use the linenumbers of the code.

```
⟨-user⟩\CodelineIndex
```

Set everything in `\MacroFont` instead of `\AltMacroFont`

```
\setcounter{StandardModuleDepth}{1}
```

For the user guide we only want the description parts of all the files.

```
⟨user⟩\OnlyDescription
```

Here starts the document

```
\begin{document}
\DocInput{babel.dtx}
```

All the language definition files.

```
⟨user⟩\clearpage
\langdeffile{esperanto.dtx}
\langdeffile{interlingua.dtx}
%
\langdeffile{dutch.dtx}
\langdeffile{english.dtx}
\langdeffile{germanb.dtx}
\langdeffile{ngermanb.dtx}
%
\langdeffile{breton.dtx}
\langdeffile{welsh.dtx}
\langdeffile{irish.dtx}
\langdeffile{scottish.dtx}
%
\langdeffile{greek.dtx}
%
\langdeffile{frenchb.dtx}
\langdeffile{italian.dtx}
\langdeffile{latin.dtx}
\langdeffile{portuges.dtx}
\langdeffile{spanish.dtx}
\langdeffile{catalan.dtx}
\langdeffile{galician.dtx}
\langdeffile{basque.dtx}
\langdeffile{romanian.dtx}
%
\langdeffile{danish.dtx}
\langdeffile{icelandic.dtx}
\langdeffile{norsk.dtx}
```

```
\langdeffile{swedish.dtx}
\langdeffile{samin.dtx}
%
\langdeffile{finnish.dtx}
\langdeffile{magyar.dtx}
\langdeffile{estonian.dtx}
%
\langdeffile{albanian.dtx}
\langdeffile{croatian.dtx}
\langdeffile{czech.dtx}
\langdeffile{polish.dtx}
\langdeffile{serbian.dtx}
\langdeffile{slovak.dtx}
\langdeffile{slovene.dtx}
\langdeffile{russianb.dtx}
\langdeffile{bulgarian.dtx}
\langdeffile{ukraineb.dtx}
%
\langdeffile{lsorbian.dtx}
\langdeffile{usorbian.dtx}
\langdeffile{turkish.dtx}
%
\langdeffile{hebrew.dtx}
\DocInput{hebinp.dtx}
\DocInput{hebrew.fdd}
\DocInput{heb209.dtx}
\langdeffile{bahasa.dtx}
\langdeffile{bahasam.dtx}
%\langdeffile{sanskrit.dtx}
%\langdeffile{kannada.dtx}
%\langdeffile{nagari.dtx}
%\langdeffile{tamil.dtx}
\clearpage
\DocInput{bbplain.dtx}
```

Finally print the index and change log (not for the user guide).

```
⟨*!user⟩
\clearpage
\def\filename{index}
\PrintIndex
\clearpage
\def\filename{changes}
\PrintChanges
⟨/!user⟩
\end{document}
⟨/driver⟩
```

15 Conclusion

A system of document options has been presented that enable the user of LaTeX to adapt the standard document classes of LaTeX to the language he or she prefers to use. These options offer the possibility of switching between languages in one document. The basic interface consists of using one option, which is the same for *all* standard document classes.

In some cases the language definition files provide macros that can be useful to plain TeX users as well as to LaTeX users. The babel system has been implemented so that it can be used by both groups of users.

16 Acknowledgements

I would like to thank all who volunteered as β-testers for their time. I would like to mention Julio Sanchez who supplied the option file for the Spanish language and Maurizio Codogno who supplied the option file for the Italian language. 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] Donald E. Knuth, *The TeXbook*, Addison-Wesley, 1986.

[2] Leslie Lamport, *LaTeX, A document preparation System*, Addison-Wesley, 1986.

[3] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst.* SDU Uitgeverij ('s-Gravenhage, 1988). A Dutch book on layout design and typography.

[4] Hubert Partl, *German TeX*, *TUGboat* 9 (1988) #1, p. 70–72.

[5] Leslie Lamport, in: TeXhax Digest, Volume 89, #13, 17 February 1989.

[6] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

[7] Joachim Schrod, *International LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.