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

This manual describes babel, a package that makes use of the capabilities of \TeX{} version 3 and, to some extent, xetex and luatex, to provide an environment in which documents can be typeset in a language other than US English, or in more than one language or script. However, no attempt has been done to take full advantage of the features provided by the latter, which would require a completely new core (as for example polyglossia or as part of \LaTeX{}3).
## Contents

### I User guide

1 The user interface
   1.1 Selecting languages .............................. 5
   1.2 More on selection ............................. 7
   1.3 Getting the current language name ............. 8
   1.4 Selecting scripts ............................. 8
   1.5 Shorthands ..................................... 9
   1.6 Package options ............................... 12
   1.7 The base option ............................... 14
   1.8 Hooks ........................................ 14
   1.9 Hyphenation tools ............................ 16
   1.10 Language attributes ........................... 17
   1.11 Languages supported by babel .................. 17
   1.12 Tips, workarounds, know issues and notes .... 19
   1.13 Future work ................................. 20

2 Preloading languages with language.dat .......................... 21

3 The interface between the core of babel and the language definition files
   3.1 Basic macros .................................. 23
   3.2 Skeleton ...................................... 24
   3.3 Support for active characters .................. 25
   3.4 Support for saving macro definitions .......... 26
   3.5 Support for extending macros .................. 26
   3.6 Macros common to a number of languages ...... 26
   3.7 Encoding-dependent strings ..................... 27

4 Compatibility and changes .................................. 30
   4.1 Compatibility with german.sty .................. 30
   4.2 Compatibility with ngerman.sty ................. 31
   4.3 Compatibility with the french package ........ 31
   4.4 Changes in babel version 3.9 .................. 31
   4.5 Changes in babel version 3.7 .................. 31
   4.6 Changes in babel version 3.6 .................. 32
   4.7 Changes in babel version 3.5 .................. 33

### II The code

5 Identification and loading of required files ................. 34
   5.1 Multiple languages ........................... 35

6 The Package File (\LaTeX) ................................ 36
   6.1 base ......................................... 37
   6.2 key=value options and other general option ... 37
   6.3 Conditional loading of shorthands ............... 38
   6.4 Language options .............................. 40
7 The kernel of Babel (common) 43
  7.1 Tools .............................................................. 43
  7.2 Hooks ............................................................ 46
  7.3 Setting up language files ........................................ 48
  7.4 Shorthands ........................................................ 49
  7.5 Language attributes ............................................. 59
  7.6 Support for saving macro definitions ............................... 62
  7.7 Short tags ........................................................ 63
  7.8 Hyphens .......................................................... 63
  7.9 Multiencoding strings ............................................ 65
  7.10 Macros common to a number of languages .......................... 71
  7.11 Making glyphs available ........................................ 71
    7.11.1 Quotation marks ............................................ 71
    7.11.2 Letters .................................................... 72
    7.11.3 Shorthands for quotation marks ............................. 73
    7.11.4 Umlauts and tremas ........................................ 75
8 The kernel of Babel (only \LaTeX) 76
  8.1 The redefinition of the style commands ............................ 76
  8.2 Cross referencing macros ......................................... 76
  8.3 Marks ............................................................. 80
  8.4 Preventing clashes with other packages ............................ 81
    8.4.1 ifthen ..................................................... 81
    8.4.2 varioref ................................................... 82
    8.4.3 hhline ....................................................... 83
    8.4.4 hyperref .................................................... 83
    8.4.5 fancyhdr .................................................... 83
  8.5 Encoding issues ................................................ 84
  8.6 Local Language Configuration ................................... 86
9 Internationalizing \LaTeX\ 2.09 86
10 Multiple languages 91
  10.1 Selecting the language ......................................... 92
  10.2 Errors ........................................................ 99
11 Loading hyphenation patterns ....................................... 100
12 The ‘nil’ language .................................................. 105
13 Support for Plain \TeX\ 106
  13.1 Not renaming hyphen.tex ...................................... 106
  13.2 Emulating some \LaTeX\ features ................................ 107
  13.3 General tools ................................................ 107
  13.4 Encoding related macros ..................................... 111
  13.5 Babel options ................................................ 114
14 Tentative font handling ............................................. 115
15 Hooks for Xe\TeX\ and Lua\TeX\ 115
  15.1 Xe\TeX\ ........................................................ 115
  15.2 Lua\TeX\ ....................................................... 117
16 Conclusion ......................................................... 119
17 Acknowledgements

3
Part I

User guide

1 The user interface

The basic 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 most cases, a single language is required, and then all you need in LATEX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument.

In multilingual documents, just use several option. So, 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.

You can also set the main language explicitly:

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

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

New 3.9c The basic behaviour of some languages can be modified when loading babel by means of modifiers. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accept them). An example is (spaces are not significant and they can be added or removed):\footnote{No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.}

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

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers. However, modifiers is a more general mechanism.
Because of the way babel has evolved, “language” can refer to (1) a set of hyphenation patterns as preloaded into the format, (2) a package option, (3) an \ldef file, and (4) a name used in the document to select a language or dialect. So, a package option refers to a language in a generic way – sometimes it is the actual language name used to select it, sometimes it is a file name loading a language with a different name, sometimes it is a file name loading several languages. Please, read the documentation for specific languages for further info.

Loading directly sty files in \LaTeX (ie, \usepackage{⟨language⟩}) is deprecated and you will get the error:\footnote{In former versions the error read “You have used an old interface to call babel”, not very helpful.}

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

Another typical error when using babel is the following:\footnote{In former versions the error read “You haven’t loaded the language LANG yet”.}

! Package babel Error: Unknown language ‘LANG’. Either you have misspelled its name, it has not been installed, or you requested it in a previous run. Fix its name, install it or just rerun the file, respectively

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

In Plain, load languages styles with \input and then use \begindocument (the latter is defined by babel):

\input estonian.sty
\begindocument

Note not all languages provide a sty file and some of them are not compatible with Plain.\footnote{Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues will be fixed soon.}

## 1.1 Selecting languages

This section describes the commands to be used in the document to switch the language in multilingual document.

The main language is selected automatically when the document environment begins. In the preamble it has not been selected, except hyphenation patterns and the name assigned to \languagename (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the following commands.

\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. For “historical reasons”, a macro name is converted to a
language name without the leading \\; in other words, the two following
declarations are equivalent:

\selectlanguage{german}
\selectlanguage{(german)}

Using a macro instead of a "real" name is deprecated.
If used inside braces there might be some non-local changes, as this would be
roughly equivalent to:

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

If you want a change which is really local, you must enclose this code with an
additional grouping level.
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.
Actually, there might be some non-local changes, as this environment is roughly
equivalent to:

\begin{group}
\selectlanguage{<inner-language>}
...
\end{group}
\selectlanguage{<outer-language>}

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

\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 and a warning is
shown).

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

Same as \foreignlanguage but as environment. Spaces after the environment are
not ignored.
This environment (or in some cases otherlanguage) may be required for
intermixing left-to-right typesetting with right-to-left typesetting in engines not
supporting a change in the writing direction inside a line.

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

The environment hyphenrules can be used to select only the hyphenation rules to
be used (it can be used as command, too). This can for instance be used to select
‘nohyphenation’, provided that in language.dat the ‘language’ nohyphenation is defined by loading zerohyph.tex. It deactivates language shorthands, too (but not user shorthands).
Except for these simple uses, hyphenrules is discouraged and otherlanguage* (the starred version) is preferred, as the former does not take into account possible changes in encodings or characters like, say, ’ done by some languages (eg, italian, frenchb, ukraineb). To set hyphenation exceptions, use \babelhyphenation (see below).

1.2 More on selection

\babeltags \{(tag1) = \langle language1\rangle, (tag2) = \langle language2\rangle, \ldots \} \new 3.9i
In multilingual documents with many language switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.
It defines \text{\langle tag1\rangle}{\langle text\rangle} to be \foreignlanguage{\langle language1\rangle}{\langle text\rangle}, and \begin{otherlanguage*}{\langle language1\rangle} to be \begin{otherlanguage*}{\langle language1\rangle}, and so on.
Note \langle tag1\rangle is also allowed, but remember set it locally inside a group. So, with \babeltags{de = german}
yo can write
text \textde{German text} text
and
text
\begin{de}
German text
\end{de}
text

\babelensure \{include=\langle commands\rangle, exclude=\langle commands\rangle, fontenc=\langle encoding\rangle\}{\langle language\rangle} \new 3.9i
Except in a few languages, like Russian, captions and dates are just strings, and do not switch the language. That means you should set it explicitly if you want to use them, or hyphenation (and in some cases the text itself) will be wrong. For example:
\foreignlanguage{russian}{text \foreignlanguage{polish}{\seename} text}
Of course, \TeX{} can do it for you. To avoid switching the language all the while, \babelensure redefines the captions for a given language to wrap them with a selector. By default only the basic captions and \today{} are redefined, but you can add further macros with the key include in the optional argument (without commas). Macros not to be modified are listed in exclude. You can also enforce a font encoding with fontenc.\footnote{With it encoded string may not work as expected.}
A couple of examples:

\babelensure[include=\Today]{spanish}
\babelensure[fontenc=T5]{vietnamese}
They are activated when the language is selected (at the afterextras event), and it makes some assumptions which could not be fulfilled in some languages. Note also you should include only macros defined by the language, not global macros (eg, \TeX of \dag).

### 1.3 Getting the current language name

\texttt{\textbackslash languagename}  

The control sequence \texttt{\textbackslash 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).

\texttt{\textbackslash iflanguage}  

\{(\texttt{language})\}\{(\texttt{true})\}\{(\texttt{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 \texttt{\textbackslash 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. The advice about \texttt{\textbackslash languagename} also applies here – use iflang instead of \texttt{\textbackslash iflanguage} if possible.

### 1.4 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either \texttt{\textbackslash fontencoding} (low level) or a language name (high level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.\(^6\)

Some languages sharing the same script define macros to switch it (eg, \texttt{\textbackslash textcyrillic}), but be aware they may also set the language to a certain default. Even the babel core defined \texttt{\textbackslash textlatin}, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main latin encoding was LY1), and therefore it has been deprecated.\(^7\)

\texttt{\textbackslash ensureascii}  

\{(\texttt{text})\}  

\textbf{New 3.9i} This macro makes sure (\texttt{text}) is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine \TeX and \LaTeX so that they are correctly typeset even with LGR or X2 (the complete list is stored in \texttt{\Babel\NonASCII}, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph. If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also \TeX and \LaTeX are not redefined); otherwise, \texttt{\textbackslash ensureascii} switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1, LGR, then it is set to LY1, but if you load LY1, T2A it is set to T2A. The symbol encodings T51, T3, and T53 are not taken into account, since they are not used for “ordinary” text.

\(^6\)The so-called Unicode fonts does not improve the situation either. So, a font suited for Vietnamese is not necessarily suited for, say, romanization of Indic languages, and the fact it contains glyphs for Modern Greek does not mean it includes them for Classic Greek. As to directionality, it poses special challenges because it also affects individual characters and layout elements.

\(^7\)But still defined for backwards compatibility.
The foregoing rules (which are applied “at begin document”) cover most of cases. Note no assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

1.5 Shorthands

A shorthand is a sequence of one or two characters that expands to arbitrary \TeX
code.

Shorthands can be used for different kinds of things, as for example: (1) in some
languages shorthands such as “a are defined to be able to hyphenate the word if
the encoding is OT1; (2) in some languages shorthands such as ! are used to insert
the right amount of white space; (3) several kinds of discretionaries and breaks
can be inserted easily with ". , ", =, etc.

The package inputenc as well as xe\TeX\ e\TeX\ have alleviated entering
non-ASCII characters, but minority languages and some kinds of text can still
require characters not directly available in the keyboards (and sometimes not even
as separated or precomposed Unicode characters). As to the point 2, now pdf\TeX
provides \texttt{\knbcode}. Tools of point 3 can be still very useful in general.

There are three levels of shorthands: user, language, and system (by order of
precedence). Version 3.9 introduces the language user level on top of the user
level, as described below. In most cases, you will use only shorthands provided by
languages.

Please, note the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing
brace \} and the spaces following are gobbled. With one-char shorthands
(eg, :) they are preserved.

2. If on a certain level (system, language, user) there is a one-char shorthand,
two-char ones starting with that char and on the same level are ignored.

A typical error when using shorthands is the following:

\begin{verbatim}
! Argument of \language@active@arg has an extra }.
\end{verbatim}

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

\begin{verbatim}
\shorthandon{⟨shorthands-list⟩}
\shorthandoff{⟨shorthands-list⟩}
\end{verbatim}

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 \texttt{\catcode} for each of the characters in its
argument to other (12); the command \shorthandon sets the \texttt{\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 3.9a 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

\begin{verbatim}
\shorthandoff*{~^}
\end{verbatim}
~ 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.

\useshortcuts *\{char\}

The command \useshortcuts initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.

New 3.9a However, user shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version \useshortcuts*\{char\} is provided, which makes sure shorthands are always activated. Currently, if the package option shorthands is used, you must include any character to be activated with \useshortcuts. This restriction will be lifted in a future release.

\defineshorthand \{\langle language\rangle, \langle language\rangle,...\}\{\langle shorthand\}\{\langle code\}\}

The command \defineshorthand takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to.

New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add \languageshorthand\{\langle lang\}\} to the corresponding \extras\{\langle 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 a unified set of shorthand for discretionaries (languages do not define shorthands consistently, and "-, \-, "= have different meanings). You could start with, say:

\useshortcuts*" \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 then set:

\defineshorthand*\{polish, *portuguese\}"\{\babelhyphen\{repeat\}\}

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

\aliashorthand \{\langle original\}\}{\langle alias\}\}

The command \aliashorthand 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, \aliasshorthands 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 \{\textit{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).\(^8\) Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

\addto\extrasenglish{\languageshorthands{ngerman}}

(You may also need to activate them with, for example, \useshorthands.) Very often, this is a more convenient way to deactivate shorthands than \shorthandoff, as for example if you want to define a macro to easy typing phonetic characters with tipa:

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

\babelshorthand \{\textit{shorthand}\}

With this command you can use a shorthand even if (1) not activated in shorthands (in this case only shorthands for the current language are taken into account, ie, not user shorthands), (2) turned off with \shorthandoff or (3) deactivated with the internal \bbl@deactivate; for example, \babelshorthand{"u} or \babelshorthand{:}. (You can conveniently define your own macros, or even you own user shorthands provided they do not overlap.) For your records, here is a list of shorthands, but you must check them, as they may change:\(^9\)

Languages with no shorthands Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh

Languages with only " as defined shorthand character Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

Basque "’ ~
Breton : ; ? !
Catalan "’ ,

\(^8\)Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.

\(^9\)Thanks to Enrico Gregorio
Czech " -
Esperanto ^
Estonian " ~
French (all varieties) : ; ? !
Galician " . ' ~ < >
Greek ~
Hungarian ' 
Kurmanji ^
Latin " ^ =
Slovak " ^ ' -
Spanish " . < > '
Turkish : ! =

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.\footnote{This declaration serves to nothing, but it is preserved for backward compatibility.}

1.6 Package options

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

\textbf{KeepShorthandsActive} Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.

\textbf{activeacute} For some languages babel supports this options to set ' as a shorthand in case it is not done by default.

\textbf{activegrave} Same for ‘.

\textbf{shorthands=} \texttt{(char)(char)... | off}
The only language shorthands activated are those given, like, eg:

\begin{verbatim}
\usepackage[esperanto,frenchb,shorthands=;!?]{babel}
\end{verbatim}

If ' is included, activeacute is set; if ‘ is included, activegrave is set. Active characters (like ~) should be preceded by \texttt{\string} (otherwise they will be expanded by \LaTeX before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma).

With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \texttt{\babelshorthand} is defined, which allows using them; see above.

\textbf{safe=} none | ref | bib

Some \LaTeX macros are redefined so that using shorthands is safe. With safe=bib only \texttt{\nocite}, \texttt{\bibcite} and \texttt{\bibitem} are redefined. With safe=ref only \texttt{\newlabel}, \texttt{\ref} and \texttt{\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.
math= active | normal
Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like $\{a\}'$ (a closing brace after a shorthand) are not a source of trouble any more.

config= (file)
Load (file).cfg instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).

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

headfoot= (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.

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

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

nocase New 3.9l Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.

silent New 3.9l No warnings and no infos are written to the log file.\footnote{You can use alternatively the package silence.}

strings= generic | unicode | encoded | ⟨label⟩ | ⟨font encoding⟩
Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional \TeX, LICR and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUppercase and the like.

hyphenmap= off | main | select | other | other*
\footnote{Turned off in plain.}
New 3.9g Sets the behaviour of case mapping for hyphenation, provided the language defines it. It can take the following values:

off deactivates this feature and no case mapping is applied;
first sets it at the first switching commands in the current or parent scope
(typically, when the aux file is first read and at \begin{document}, but also
the first \selectlanguage in the preamble), and it's the default if a single
language option has been stated;\footnote{Duplicated options count as several ones.}
select sets it only at \selectlanguage;
other also sets it at otherlanguage;
other* also sets it at otherlanguage* as well as in heads and foots (if the option
headfoot is used) and in auxiliary files (ie, at \select@language), and it's the
default if several language options have been stated. The option first can
be regarded as an optimized version of other* for monolingual documents.\footnote{Providing foreign is pointless, because the case mapping applied is that at the end of paragraph, but
if either xetex or luatex change this behaviour it might be added. On the other hand, other is provided
even if I [JBL] think it isn't really useful, but who knows.}

1.7 The base option

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

\AfterBabelLanguage{(option-name)}{(code)}
This command is currently the only provided by base. Executes \code when the
file loaded by the corresponding package option is finished (at \ldf@finish). The
setting is global. So

\AfterBabelLanguage{frenchb}{...}

does ... at the end of frenchb.ldf. It can be used in ldf files, too, but in such a
case the code is executed only if \texttt{(option-name)} is the same as \CurrentOption
(which could not be the same as the option name as set in \usepackage!).
For example, consider two languages foo and bar defining the same \macro with
\newcommand. An error is raised if you attempt to load both. Here is a way to
overcome this problem:

\usepackage[base]{babel}
\AfterBabelLanguage{foo}{
  \let\macroFoo\macro
  \let\macro\relax
}\usepackage[foo,bar]{babel}

1.8 Hooks

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

\AddBabelHook{(name)}{(event)}{(code)}
The same name can be applied to several events. Hooks may be enabled and
disabled for all defined events with \EnableBabelHook{(name)},
\DisableBabelHook{\langle \text{name} \rangle}. Names containing the string babel are reserved (they are used, for example, by \useshortands* to add a hook for the event \text{afterextras}).

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

**adddialect** (language name, dialect name) Used by \text{luababel.def} to load the patterns if not preloaded.

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

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

**defaultcommands** Used (locally) in \text{\StartBabelCommands}.

**encodedcommands** (input, font encodings) Used (locally) in \text{\StartBabelCommands}.

  Both \text{xetex} and \text{luatex} make sure the encoded text is read correctly.

**stopcommands** Used to reset the the above, if necessary.

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

**beforeextras** Just before executing \text{\extras\langle \text{language} \rangle}. This event and the next one should not contain language-dependent code (for that, add it to \text{\extras\langle \text{language} \rangle}).

**afterextras** Just after executing \text{\extras\langle \text{language} \rangle}. For example, the following deactivates shorthands in all languages:

\texttt{\AddBabelHook{noshort}{\text{afterextras}}{\backslash \languageshorthands{none}}}

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

\begin{verbatim}
\AddBabelHook{myhook}{\text{stringprocess}}{
\protect\@edef\BabelString{\BabelString}}
\end{verbatim}

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

**afterreset** \textbf{New \text{3.9i}} Executed when selecting a language just after \text{\originalTeX} is run and reset to its base value, before executing \text{\captions\langle \text{language} \rangle} and \text{\date\langle \text{language} \rangle}.

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

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

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

**loadpatterns** (patterns file) Loads the patterns file. Used by \text{luababel.def}.

**loadexceptions** (exceptions file) Loads the exceptions file. Used by \text{luababel.def}.
New 3.9a This macro contains a list of “toc” types which require a command to
switch the language. Its default value is toc, lof, lot, but you may redefine it with
\renewcommand (it’s up to you to make sure no toc type is duplicated).

1.9 Hyphenation tools

\babelhyphen \{\{type\}\}\{\{text\}\}

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

In TeX, - and \\- forbid further breaking opportunities in the word. This is the
desired behaviour very often, but not always, and therefore many languages
provide shorthands for these cases. Unfortunately, this has not been done
d consistently: for example, in Dutch, Portuguese, 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 opportunities in the rest of the word.

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

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

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

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

There are also some differences with \LaTeX: (1) the character used is that set for
the current font, while in \LaTeX it is hardwired to - (a typical value); (2) the hyphen
to be used in fonts with a negative \hyphenchar is -, like in \LaTeX, but it can be
changed to another value by redefining \texttt{\newhyphenchar}; (3) a break after the
hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided
it is not immediately preceded by, say, a parenthesis).
\begin{itemize}
  \item \texttt{\babelhyphenation [\textit{language},\textit{language},...]{\textit{exceptions}}} \hfill \textbf{New 3.9a}
    Sets hyphenation exceptions for the languages given or, without the optional argument, for all languages (e.g., 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 \texttt{\lccodes}'s done in \texttt{\extras/lang} as well as the language specific encoding (not set in the preamble by default). Multiple \texttt{\babelhyphenation}'s are allowed. For example:

    \begin{verbatim}
    \babelhyphenation{Wal-hal-la Dar-bhan-ga}
    \end{verbatim}

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

  \item \texttt{\babelpatterns [\textit{language},\textit{language},...]{\textit{patterns}}} \hfill \textbf{New 3.9m}
    In \texttt{luatex} only,\textsuperscript{15} adds or replaces patterns for the languages given or, without the optional argument, for all languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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

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

\end{itemize}

\section*{1.10 Language attributes}
\texttt{\languageattribute} \hfill This is a user-level command, to be used in the preamble of a document (after \texttt{\usepackage[...]{babel}}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

Very often, using a \texttt{modifier} in a package option is better.

Several language definition files use their own methods to set options. For example, \texttt{frenchb} uses \texttt{\frenchbsetup}, \texttt{magyar} \texttt{(1.5)} uses \texttt{\magyarOptions}; \texttt{modifiers provided by \texttt{spanish} have no attribute counterparts. Macros setting options are also used (e.g., \texttt{\ProsodicMarksOn} in \texttt{latin})}.

\section*{1.11 Languages supported by babel}
In the following table most of the languages supported by babel are listed, together with the names of the options which you can load babel with for each language. Note this list is open and the current options may be different.

\begin{tabular}{ll}
\textbf{Afrikaans} & \texttt{afrikaans} \\
\end{tabular}

\textsuperscript{15}With \texttt{luatex} exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and \texttt{babel} only provides the most basic tools.
There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygrek, bgreek, serbiana, frenchle, ethiop and friulan.

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

\documentclass{article}
\usepackage[hindi]{babel}
\begin{document}
Then you preprocess it with devnag \langle file\rangle, which creates \langle file\rangle . tex; you can then typeset the latter with \LaTeX.

1.12 Tips, workarounds, know issues and notes

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

- Both ltxdoc and babel use \AtBeginDocument to change some catcodes, and babel reloads 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

\begin{verbatim}
\AtBeginDocument{\DeleteShortVerb{|}}
\end{verbatim}

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:

\begin{verbatim}
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrarussian{\inputencoding{koi8-r}}
\end{verbatim}

(A recent version of inputenc is required.)

- For the hyphenation to work correctly, lccodes cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.\footnote{This explains why \LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, \savinghyphcodes is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.} So, if you write a chunk of French text with \foreignlanguage, the apostrophes might not be taken into account. This is a limitation of \TeX, not of babel. Alternatively, you may use \useshorthands to activate ’ and \defineshorthand, or redefine \textquoteright (the latter is called by the non-ASCII right quote).

- \bibitem is out of sync with \selectlanguage in the .aux file. The reason is \bibitem uses \immediate (and others, in fact), while \selectlanguage doesn’t. There is no known workaround.

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

• Plain luatex does not load patterns on the fly. Since this format is not based on babel but on etex.src further investigation is required. This is another task in the ‘to do’ list.

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

- **csquotes** Logical markup for quotes.
- **iflang** Tests correctly the current language.
- **hyphsubst** Selects a different set of patterns for a language.
- **translator** An open platform for packages that need to be localized.
- **siunitx** Typesetting of numbers and physical quantities.
- **biblatex** Programmable bibliographies and citations.
- **bicaption** Bilingual captions.
- **babelbib** Multilingual bibliographies.
- **microtype** Adjusts the typesetting according to some languages (kerning and spacing). Ligatures can be disabled.
- **substitutefont** Combines fonts in several encodings.
- **mkpattern** Generates hyphenation patterns.

### 1.13 Future work

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

More interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ből”, in Spanish an item labelled “3.o” may be referred to as either “item 3.o” or “3.er item”, and so on. Even more interesting is right-to-left, vertical and bidi typesetting. Babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better). Handling of “Unicode” fonts is also problematic. There is fontspec, but special macros are required (not only the NFSS ones) and it doesn’t provide “orthogonal axis” for features, including those related to the language (mainly language and script). A couple of tentative macros, which solve the two main cases, are provided by babel (≥3.9g) with a partial solution (only xetex and luatex, for obvious reasons), but use them at your own risk, as they might be removed in the future.

For this very reason, they are described here:

- \texttt{\textbackslash babelFSstore{\langle babel-language \rangle}} sets the current three basic families (rm, sf, tt) as the default for the language given. In most cases, this macro will be enough.

- \texttt{\textbackslash babelFSdefault{\langle babel-language \rangle}{\langle fontspec-features \rangle}} patches \fontspec so that the given features are always passed as the optional argument or added to it (not an ideal solution). Use it only if you select some fonts in the document with \fontspec.

---

17 See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR).
So, for example:

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

Note you can set any feature required for the language – not only Language, but also Script or a local .fea. This makes those macros a bit more verbose, but also more powerful.

2 Preloading languages with language.dat

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

In that file the person who maintains a \TeX{} environment has to record for which languages he has hyphenation patterns and in which files these are stored\(^{18}\). When hyphenation exceptions are stored in a separate file this can be indicated by naming that file after the file with the hyphenation patterns.

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

\begin{verbatim}
% 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
\end{verbatim}

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.\(^{19}\) For example:

\begin{verbatim}
german:T1 hyphenT1.ger
german hyphen.ger
\end{verbatim}

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

A typical error when using babel is the following:

No hyphenation patterns were preloaded for the language ‘<lang>’ into the format.
Please, configure your \TeX{} system to add them and rebuild the format. Now I will use the patterns

\(^{18}\)This is because different operating systems sometimes use very different file-naming conventions.
\(^{19}\)This in not a new feature, but in former versions it didn’t work correctly.
It simply means you must reconfigure `language.dat`, either by hand or with the tools provided by your distribution.

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

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

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

- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\langle lang\rangle_{hyphenmins}`, `\langle lang\rangle_{captions}`, `\langle lang\rangle_{date}`, `\langle lang\rangle_{extras}` and `\langle lang\rangle_{noextras}` (the last two may be left empty); where `\langle lang\rangle` is either the name of the language definition file or the name of the \LaTeX{} option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\langle lang\rangle_{date}` but not `\langle lang\rangle_{captions}` does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\langle lang\rangle_{\@}` to be a dialect of `\language0` when `\langle lang\rangle_{\@}` is undefined.

- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.

- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (e.g., `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

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

• Avoid adding things to \noextras＜lang＞ except for umlauthigh and friends, \bbl@deactivate, \bbl@(non)frenchspacing, and language specific macros. Use always, if possible, \bbl@save and \bbl@savevariable (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in ＜extras＜lang＞＞.

• Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low level) or the language (high level, which in turn may switch the font encoding). Usage of things like \latintext is deprecated.\(^{20}\)

There are no special requirements for documenting your language files. Now they are not included in the base babel manual, so provide a standalone document suited for your needs (and the corresponding PDF, if you like), as well as other files you think can be useful (eg, samples, readme).

### 3.1 Basic macros

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

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

\adddialect The macro \adddialect can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default 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.

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

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

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

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

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

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

\extras＜lang＞ The macro \extras＜lang＞ contains all the extra definitions needed for a specific

\(^{20}\)But not removed, for backward compatibility.
language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.

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

\bbl@declare@tribute 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 \caption{} 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 (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct \LaTeX{} to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.2 Skeleton

Here is the basic structure of an ldf file, with a language, a dialect and an attribute. Strings are best defined using the method explained in in sec. 3.7 (babel 3.9 and later).

\ProvidesLanguage{<language>}
[0000/00/00 v0.0 <Language> support from the babel system]
\LdfInit{<language>}{captions<language>}
\ifx\undefined\l@<language>
\@nopatterns{<Language>}
\adddialect\l@<language>0
\fi
\adddialect\l@<dialect>\l@<language>
\bbl@declare@tribute{<language>}{<attrib>}{
\expandafter\addto\expandafter\extras<language>
3.3 Support for active characters

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

\texttt{\textbackslash initiate@active@char} The internal macro \texttt{\textbackslash 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.

\texttt{\textbackslash bbl@activate} The command \texttt{\textbackslash bbl@activate} is used to change the way an active character expands. \texttt{\textbackslash bbl@activate} ‘switches on’ the active behaviour of the character.

\texttt{\textbackslash bbl@deactivate} \texttt{\textbackslash bbl@deactivate} lets the active character expand to its former (mostly) non-active self.

\texttt{\textbackslash declare@shorthand} The macro \texttt{\textbackslash 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. \texttt{-} or \texttt{“a”}; and the code to be executed when the shorthand is encountered. (It does \textit{not} raise an error if the shorthand character has not been “initiated”.)

\texttt{\textbackslash bbl@add@special} The \TeX{}book states: “Plain \TeX{} includes a macro called \texttt{\textbackslash 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 \texttt{\textbackslash dospecials}. \LaTeX{} adds another macro called \texttt{\textbackslash sanitize} representing the same character set, but without the curly braces. The macros
\bbl@add@special\langle char\rangle and \bbl@remove@special\langle char\rangle add and remove the character \langle char\rangle to these two sets.

### 3.4 Support for saving macro definitions

Language definition files may want to redefine macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this:\footnote{This mechanism was introduced by Bernd Raichle.}

\texttt{\bbl@save} To save the current meaning of any control sequence, the macro \texttt{\bbl@save} is provided. It takes one argument, \langle csname \rangle, the control sequence for which the meaning has to be saved.

\texttt{\bbl@savevariable} A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \texttt{\the} primitive is considered to be a variable. The macro takes one argument, the \langle variable \rangle.

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

### 3.5 Support for extending macros

\texttt{\addto} The macro \texttt{\addto\{\langle control sequence\}\{\langle TeX code\}\}} can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or \texttt{\relax}). This macro can, for instance, be used in adding instructions to a macro like \texttt{\extrasenglish}.

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

### 3.6 Macros common to a number of languages

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

\texttt{\allowhyphens} Same as \texttt{\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 \texttt{\accent} in OT1.

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

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

\texttt{\save@sf@q} Sometimes it is necessary to preserve the \texttt{\spacefactor}. For this purpose the macro \texttt{\save@sf@q} is available. It takes one argument, saves the current spacefactor, executes the argument, and restores the spacefactor.
The commands \bbl@frenchspacing and \bbl@nonfrenchspacing can be used to properly switch French spacing on and off.

### 3.7 Encoding-dependent strings

**New 3.9a** Babel 3.9 provides a way of defining strings in several encodings, intended mainly for luatex and xetex. This is the only new feature requiring changes in language files if you want to make use of it. Furthermore, it must be activated explicitly, with the package option strings. If there is no strings, these blocks are ignored, except \SetCases (and except if forced as described below). In other words, the old way of defining/switching strings still works and it’s used by default.

It consist is a series of blocks started with \StartBabelCommands. The last block is closed with \EndBabelCommands. Each block is a single group (ie, local declarations apply until the next \StartBabelCommands or \EndBabelCommands).

An ldf may contain several series of this kind.

Thanks to this new feature, string values and string language switching are not mixed any more. No need of \addto. If the language is french, just redefine \frenchchaptername. {\langle language-list\rangle}{\langle category\rangle}{\langle selector\rangle}

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

A “selector” is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name unicode must be used for xetex and luatex (the key strings has also other two special values: generic and encoded).

If a string is set several times (because several blocks are read), the first one take precedence (ie, it works much like \providecommand).

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

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

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with strings=generic (no block is taken into account except those). With strings=encoded, strings in those blocks are set as default (internally, \?). With strings=encoded strings are protected, but they are correctly expanded in \MakeUppercase and the like. If there is no key strings, string definitions are ignored, but \SetCases are still honoured (in a encoded way).
The ⟨category⟩ is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.\footnote{In future releases further categories may be added.} It may be empty, too, but in such a case using \SetString is an error (but not \SetCase).

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

A real example is:

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

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

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

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

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

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

When used in ldf files, previous values of \⟨category⟩⟨language⟩ are overriden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \date⟨language⟩ exists).
The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It’s up to the maintainers of the current languages to decide if using it is appropriate.\footnote{This replaces in 3.9g a short-lived \UseStrings which has been removed because it did not work.}

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands \langle code \rangle
The code is delayed and executed at the global scope just after \EndBabelCommands.

\SetString \langle macro-name \rangle \langle string \rangle
Adds \langle macro-name \rangle to the current category, and defines globally \langle lang-macro-name \rangle to \langle code \rangle (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop \langle macro-name \rangle \langle string-list \rangle
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with \abday):
\SetStringLoop{abmon#1name}{en,fb,ar,my,ja,gi,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
#1 is replaced by the roman numeral.

\SetCase \langle map-list \rangle \langle toupper-code \rangle \langle tolower-code \rangle
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would be typically things like \let\bb\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A \langle map-list \rangle is a series of macros using the internal format of \@ucclist (eg, \bb\bb\cc\cc). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only.
For example, as T1 is the default case mapping in \LaTeX, we could set for Turkish:
\SetCase
{\uccode"10='I\relax
{\lccode'='1\relax
\SetCase
{\uccode'i='˙I\relax
\uccode'ı='I\relax
{\lccode'='i\relax
\lccode'I='ı\relax

\StartBabelCommands {turkish}\{ot1enc, fontenc=OT1\}
\SetCase
{\uccode"10='I\relax
{\lccode'='1\relax
\StartBabelCommands {turkish}\{unicode, fontenc=EU1 EU2, charset=utf8\}
\SetCase
{\uccode'i='I\relax
\uccode'ı='I\relax
{\lccode'I='ı\relax
\lccode'I='ı\relax

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

\SetString \langle macro-name \rangle \langle string \rangle
Adds \langle macro-name \rangle to the current category, and defines globally \langle lang-macro-name \rangle to \langle code \rangle (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop \langle macro-name \rangle \langle string-list \rangle
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with \abday):
\SetStringLoop{abmon#1name}{en,fb,ar,my,ja,gi,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
#1 is replaced by the roman numeral.

\SetCase \langle map-list \rangle \langle toupper-code \rangle \langle tolower-code \rangle
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would be typically things like \let\bb\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A \langle map-list \rangle is a series of macros using the internal format of \@ucclist (eg, \bb\bb\cc\cc). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only.
For example, as T1 is the default case mapping in \LaTeX, we could set for Turkish:
\SetCase
{\uccode"10='I\relax
{\lccode'='1\relax
\SetCase
{\uccode'i='˙I\relax
\uccode'ı='I\relax
{\lccode'='i\relax
\lccode'I='ı\relax

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

\SetString \langle macro-name \rangle \langle string \rangle
Adds \langle macro-name \rangle to the current category, and defines globally \langle lang-macro-name \rangle to \langle code \rangle (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop \langle macro-name \rangle \langle string-list \rangle
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with \abday):
\SetStringLoop{abmon#1name}{en,fb,ar,my,ja,gi,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
#1 is replaced by the roman numeral.

\SetCase \langle map-list \rangle \langle toupper-code \rangle \langle tolower-code \rangle
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would be typically things like \let\bb\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A \langle map-list \rangle is a series of macros using the internal format of \@ucclist (eg, \bb\bb\cc\cc). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only.
For example, as T1 is the default case mapping in \LaTeX, we could set for Turkish:
\SetCase
{\uccode"10='I\relax
{\lccode'='1\relax
\SetCase
{\uccode'i='˙I\relax
\uccode'ı='I\relax
{\lccode'='i\relax
\lccode'I='ı\relax

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

\SetString \langle macro-name \rangle \langle string \rangle
Adds \langle macro-name \rangle to the current category, and defines globally \langle lang-macro-name \rangle to \langle code \rangle (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop \langle macro-name \rangle \langle string-list \rangle
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with \abday):
\SetStringLoop{abmon#1name}{en,fb,ar,my,ja,gi,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
#1 is replaced by the roman numeral.

\SetCase \langle map-list \rangle \langle toupper-code \rangle \langle tolower-code \rangle
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would be typically things like \let\bb\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A \langle map-list \rangle is a series of macros using the internal format of \@ucclist (eg, \bb\bb\cc\cc). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only.
For example, as T1 is the default case mapping in \LaTeX, we could set for Turkish:
\SetCase
{\uccode"10='I\relax
{\lccode'='1\relax
\SetCase
{\uccode'i='˙I\relax
\uccode'ı='I\relax
{\lccode'='i\relax
\lccode'I='ı\relax

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

\SetString \langle macro-name \rangle \langle string \rangle
Adds \langle macro-name \rangle to the current category, and defines globally \langle lang-macro-name \rangle to \langle code \rangle (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop \langle macro-name \rangle \langle string-list \rangle
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with \abday):
\SetStringLoop{abmon#1name}{en,fb,ar,my,ja,gi,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
#1 is replaced by the roman numeral.

\SetCase \langle map-list \rangle \langle toupper-code \rangle \langle tolower-code \rangle
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would be typically things like \let\bb\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A \langle map-list \rangle is a series of macros using the internal format of \@ucclist (eg, \bb\bb\cc\cc). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only.
For example, as T1 is the default case mapping in \LaTeX, we could set for Turkish:
(Note the mapping for OT1 is not complete.)

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

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

- \BabelLower{\uccode}{\lccode} is similar to \lccode but it’s ignored if the char has been set and saves the original lccode to restore it when switching the language (except with hyphenmap=first).

- \BabelLowerMM{\uccode-from}{\uccode-to}{\step}{\lccode-from} loops though the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for many-to-many).

- \BabelLowerMO{\uccode-from}{\uccode-to}{\step}{\lccode} loops though the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for many-to-one).

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

\SetHyphenMap{\BabelLowerMM{“100}{“11F}{2}{“101}}

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

4 Compatibility and changes

4.1 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 \langle number \rangle 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\textsuperscript{24}. 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.

### 4.2 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.

### 4.3 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.

### 4.4 Changes in babel version 3.9

Most of changes in version 3.9 are related to bugs, either to fix them (there were lots), or to provide some alternatives. Even new features like \texttt{\textbackslash babel\textbackslash hyphen} are intended to solve a certain problem (in this case, the lacking of a uniform syntax and behaviour for shorthands across languages). These changes are described in this manual in the corresponding place.

### 4.5 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:

- Shorthands are expandable again. The disadvantage is that one has to type '{\textbackslash 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, \texttt{\textbackslash shorthandon} and \texttt{\textbackslash 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 \texttt{\textbackslash greeknumeral} has been added.

- Support for typesetting Basque is now available thanks to Juan Aguirregabiria.

\textsuperscript{24}The ‘b’ is added to the name to distinguish the file from Partls’ file.
• 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.

4.6 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 ‘{‘} works as expected for languages that have the ‘ active.

• Support for typesetting French texts is much enhanced; the file français.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\textsuperscript{2e}.

4.7 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.

Part II
The code

5 Identification and loading of required files

*Code documentation is still under revision.*

The babel package after unpacking it consists of the following files:

- `switch.def` defines macros to set and switch languages.
- `babel.def` defines the rest of macros. It has tow parts: a generic one and a second one only for \LaTeX{}.
- `babel.sty` is the \LaTeX{} package, which set options and load language styles.
- `plain.def` defines some \LaTeX{} macros required by `babel.def` and provides a few tools for Plain.
- `hyphen.cfg` is the file to be used when generating the formats to load hyphenation patterns. By default it also loads `switch.def`.

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

```
1 ⟨⟨version=3.9m⟩⟩
2 ⟨⟨date=2015/08/03⟩⟩
```

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

```
3 ⟨⟨*Basic macros⟩⟩ ≡
4 \def\bbl@add#1#2{%
5 \@ifundefined{\expandafter\@gobble\string#1}{
6 \def#1{#2}}{
7 \expandafter\def\expandafter#1\expandafter{#1#2}}%
8 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
9 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
```

34
Some files identify themselves with a \LaTeX macro. The following code is placed before them to define (and then undefine) if not in \LaTeX.  

\input switch.1df is used, if different from that in the format.  

The following code is used in babel.sty and babel.def, and makes sure the current version of switch.1df is used, if different from that in the format.  

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

The following code is used in switch.def.  

5.1 \textbf{Multiple languages}

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

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

To add languages to \TeX’s memory plain \TeX version 3.0 supplies \texttt{\addlanguage}, 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 \texttt{\newlanguage} was defined to be \texttt{\outer}.
For a format based on plain version 2.x, the definition of \newlanguage cannot be copied because \count 19 is used for other purposes in these formats. Therefore \addlanguage is defined using a definition based on the macros used to define \newlanguage in plain \TeX version 3.0. For formats based on plain version 3.0 the definition of \newlanguage can be simply copied, removing \outer. Plain \TeX version 3.0 uses \count 19 for this purpose.

42 ⟨⟨∗ Define core switching macros⟩⟩ ≡
43 \ifx\newlanguage\@undefined
44 \csname newcount\endcsname\last@language
45 \def\addlanguage#1{%
46 \global\advance\last@language\@ne
47 \ifnum\last@language<\@cclvi
48 \else
49 \errmessage{No room for a new \string\language!}%
50 \fi
51 \global\chardef\last@language\@cclvi
52 \wlog{\string#1 = \string\language\the\last@language}}
53 \else
54 \countdef\last@language=19
55 \def\addlanguage{\alloc@9\language\chardef\@cclvi}
56 \fi
57 ⟨⟨/Define core switching macros⟩⟩

Identify each file that is produced from this source file.
58 ⟨∗ driver & user⟩
59 \ProvidesFile{babel.drv}[(⟨⟨date⟩⟩ ⟨⟨version⟩⟩)]
60 ⟨/driver & user⟩
61 ⟨∗ driver & user⟩
62 \ProvidesFile{user.drv}[(⟨⟨date⟩⟩ ⟨⟨version⟩⟩)]
63 ⟨/driver & user⟩

Now we make sure all required files are loaded. When the command \AtBeginDocument doesn't exist we assume that we are dealing with a plain-based format or \LaTeX2ε. In that case the file plain.def is needed (which also defines \AtBeginDocument, and therefore it is not loaded twice). We need the first part when the format is created, and \orig@dump is used as a flag. Otherwise, we need to use the second part, so \orig@dump is not defined (plain.def undefines it). Check if the current version of switch.def has been previously loaded (mainly, hyphen.cfg). If not, load it now. We cannot load babel.def here because we first need to declare and process the package options.

6 The Package File (\LaTeX)

In order to make use of the features of \LaTeX2ε, 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 additional package options. Apart from all the language options below we also have a few options that influence the behaviour of language definition files. Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user.
6.1 base

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

```
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{babel}[^date] [version] The Babel package
\@ifpackagewith{babel}{debug}{\input switch.def\relax}
\@ifpackagewith{babel}{base}{}
\DeclareOption*{\bbl@patterns{\CurrentOption}}\DeclareOption{base}{}
\ProcessOptions
\global\expandafter\let\csname opt@babel.sty\endcsname\relax
\global\expandafter\let\csname ver@babel.sty\endcsname\relax
\global\let\@ifl@ter@@\@ifl@ter
\def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}
\endinput{}
```

6.2 key=value options and other general option

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

```
\bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
\def\bbl@tempb#1.#2{#1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}
\def\bbl@tempd#1.#2\@nnil{\ifx\@empty#2%
  \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1\ifin@
    \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}
  \else
    \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}
    \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}
  \fi}
\else
  \\in@{}\#1\ifin@
  \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1\if\#1.\fi1}\%}
\else
  \\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\fi2\@nnil}
\fi}
\let\bbl@tempc\@empty
\bbl@for\bbl@tempa\bbl@tempa{%
  \edef{\bbl@tempc}{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\fi2\@nnil}
  \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\fi2\@nnil}
  \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\fi2\@nnil}
  \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}\%
  \\fi}
\let\bbl@tempc\@empty
```

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}{}
```
Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax `<key>=<value>`, the second one loads the requested languages, except the main one if set with the key `main`, and the third one loads the latter. First, we “flag” valid keys with a nil value.

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

Now the option list is processed, taking into account only currently declared options (including those declared with a `=`), and `<key>=<value>` options (the former take precedence). Unrecognized options are saved in `\bbl@language@opts`, because they are language options.

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

### 6.3 Conditional loading of shorthands

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

\def\bbl@ifshorthand#1{\@expandtwoargs\in@{\string#1}{\bbl@opt@shorthands}\ifin@\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}

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

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

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

\edef\bbl@opt@shorthands{"\@expandtwoargs\in@{\string#1}{\bbl@opt@shorthands}\@empty}%
\bbl@ifshorthand{"}{\PassOptionsToPackage{activeacute}{babel}}{}
\bbl@ifshorthand{"}{\PassOptionsToPackage{activegrave}{babel}}{}
\fi

With headfoot=lang we can set the language used in heads/foots. For example, in babel/3796 just adds headfoot=english. It misuses \@resetactivechars but seems to work.

\ifx\bbl@opt@headfoot\@nnil\else
\g@addto@macro\@resetactivechars{%
\set@typeset@protect\expandafter\select@language@x\expandafter{\bbl@opt@headfoot}\let\protect\noexpand\expandafter\@secondoftwo\fi

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

\ifundefined{bbl@opt@safe}\{\def\bbl@opt@safe{BR}\}\fi
\ifx\bbl@opt@main\@nnil\else
\edef\bbl@language@opts{%
\ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
\bbl@opt@main}%
\fi

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

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

\let\bbl@afterlang\relax
\let\BabelModifiers\relax
\let\bbl@loaded\@empty
\def\bbl@load@language#1{\InputIfFileExists{#1.ldf}{\edef\bbl@loaded{\CurrentOption\ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}\expandafter\let\expandafter\bbl@afterlang\csname\CurrentOption.ldf-h@@k\endcsname\expandafter\let\expandafter\BabelModifiers\csname bbl@mod@\CurrentOption\endcsname}\bbl@error{\unknown option '{\CurrentOption}'. Either you misspelled it or the language definition file {\CurrentOption.ldf} was not found}{Valid options are: shorthands=, KeepShorthandsActive, activeacute, activegrave, noconfigs, safe=, main=, math=\headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

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

\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{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}}
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.

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in bbl@language@opts are assumed to be languages (note this list also contains the language given with main). If not declared above, the name of the option and the file are the same.
Now, we make sure an option is explicitly declared for any language set as global option, by checking if an ldf exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.

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

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (except, of course, global options, which \LaTeX{} processes before):

This finished the second pass. Now the third one begins, which loads the main language set with the key \texttt{main}. A warning is raised if the main language is not the same as the last named one, or if the value of the key \texttt{main} is not a language. Then execute directly the option (because it could be used only in \texttt{main}). After loading all languages, we deactivate \texttt{\AfterBabelLanguage}.

Last declared language option is \texttt{\bbl@tempc,\ldots} but the last processed one was \texttt{\bbl@tempb,\ldots} The main language cannot be set as both a global\ldots and a package option. Use \texttt{main=\bbl@tempc} as\ldots option. Reported\ldots
In order to catch the case where the user forgot to specify a language we check whether \texttt{\bbl@main@language}, has become defined. If not, no language has been loaded and an error message is displayed.

\input{package}

\section{The kernel of Babel (common)}

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

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

\subsection{Tools}

\texttt{\bbl@engine} takes the following values: 0 is pdf\TeX, 1 is luatex, and 2 is xetex. You may use it in your language style if necessary.

\input{core}
\ifeof1
\closein1
\message{I couldn’t find the file language.def}
\else
\closein1
\begingroup
\def\addlanguage#1#2#3#4#5{%
  \expandafter\ifx\csname lang@#1\endcsname\relax\else
    \global\expandafter\let\csname l@#1\expandafter\endcsname
      \csname lang@#1\endcsname
  \fi%
  \def\uselanguage#1{}%
\input language.def
\endgroup
\fi
\chardef\l@english\z@
\fi
⟨⟨
Basic macros⟩⟩
\def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
\chardef\bbl@engine=%
\ifx\directlua\@undefined
\ifx\XeTeXinputencoding\@undefined
\z@
\else
\tw@
\fi
\else
\@ne
\fi
\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\textsuperscript{25}. 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}
\addto
For each language four control sequences have to be defined that control the
language-specific definitions. To be able to add something to these macro once
they have been defined the macro \addto is introduced. It takes two arguments, a
〈control sequence〉 and \TeX-code to be added to the 〈control sequence〉. If the 〈control sequence〉
has not been defined before it is defined now. The control
sequence could also expand to \relax, in which case a circular definition results.
The net result is a stack overflow. Otherwise the replacement text for the 〈control sequence〉
is expanded and stored in a token register; together with the \TeX-code
to be added. Finally the 〈control sequence〉 is redefined, using the contents of the
token register.
\def\addto#1#2{%
  \ifx#1\@undefined
    \def#1{#2}%
  \else
    \def#1(#2)%
  \else
    \ifx#1\relax
\textsuperscript{25}This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power
Lemma” by Sonja Maus.
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~} \}

\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@...
\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\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. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define \foo.
\def\bbl@redefinerobust#1{% 
\edef\bbl@tempa{\expandafter\@gobble\string#1}% 
\expandafter\ifx\csname\bbl@tempa\endcsname\space\endcsname\relax 
\expandafter\def\csname\bbl@tempa\endcsname\bbl@tempa\endcsname} \}
\@onlypreamble\bbl@redefinerobust

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

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

```latex
\def\AddBabelHook#1#2{% 
  \@ifundefined{bbl@hk@#1}{\EnableBabelHook{#1}}{% 
    \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}% 
    \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty 
    \@ifundefined{bbl@ev@#1@#2}{\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}% 
      \bbl@csarg\newcommand}{\bbl@csarg\let{ev@#1@#2}\relax 
      \bbl@csarg\newcommand}{ev@#1@#2}\[\bbl@tempb]}
\def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone} 
\def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble} 
\def\bbl@usehooks#1#2{% 
  \def\bbl@elt##1{\@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}% 
  \@nameuse{bbl@ev@#1}}
```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```latex
\def\bbl@evargs{,% don’t delete the comma 
  everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,% 
  adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,% 
  beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,% 
  hyphenation=2,initiateactive=3,afterreset=0}
```

\texttt{\bbl@ensure} The user command just parses the optional argument and creates a new macro named \bbl@ens@⟨language⟩. We register a hook at the afterextras event which just executes this macro in a “complete” selection (which, if undefined, is \relax and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro \bbl@ens@⟨language⟩ contains \bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}, which in turn loops over the macros names in \bbl@ensured, excluding (with the help of \in@) those in the exclude list. If the fontenc is given (and not \relax), the \fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

\bbl@ensured is the list of macros supposed to be “ensured”.

```latex
\newcommand\bbl@ensure[2][]{% 
  \AddBabelHook{babel-ensure}{afterextras}{% 
    \ifcase\bbl@select@type% 
      \@nameuse{bbl@e@\languagename}% 
    \fi}% 
  \begingroup% 
    \bbl@ens@include\@empty% 
  \endgroup
  \let\bbl@ens@include\@empty
```

\texttt{\bbl@ensure}
\let\bbl@ens@exclude\empty
\def\bbl@ens@fontenc{\relax}
\def\bbl@tempb##1{%  
  \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi%
  
  \edef\bbl@tempa{\bbl@tempb#1\@empty}%
  
  \def\bbl@tempb##1=##2\@@{
    \@namedef{bbl@ens@##1}{##2}}%

  \bbl@for\bbl@tempa\bbl@tempa{%    
    \expandafter\bbl@add\expandafter\bbl@tempc
      \expandafter{\bbl@tempa}\expandafter{}
  }

  \def\bbl@tempc{\bbl@ensure}

  \expandafter\bbl@add\expandafter\bbl@tempc
    \expandafter{\bbl@tempc}\expandafter{}

  \edef\x{%
    \endgroup
    \noexpand\@namedef{bbl@e@#2}{\the	oks@{\bbl@ens@fontenc}}%  }

\def\bbl@ensured#1#2#3{%
  \def\bbl@tempb##1{% elt for \bbl@ensured list
    \ifx##1\@empty\else
      \in@{##1}{#2}\%
    \fi
  }

  \edef\bbl@tempa{% elt for include list
    \ifx##1\@empty\else
      \bbl@csarg\in@{ensure@\languagename\expandafter}{##1}\%
    \fi
  }

  \edef\x{%
    \endgroup
    \noexpand\@namedef{bbl@e@#2}{\the	oks@{\bbl@ens@fontenc}}%  }

\def\bbl@ensured{\bbl@ensured@empty}

\def\bbl@tempb##1{% elt for \bbl@ensured@empty
  \ifx##1\@empty\else
    \in@{##1}{\bbl@ens@exclude}\%
  \fi
}

\edef\x{%
  \endgroup
  \noexpand\@namedef{bbl@e@#2}{\the	oks@{\bbl@ens@fontenc}}%  }

\def\bbl@ensured@empty{%
  \bbl@tempb##1=##2\@@{
    \@namedef{bbl@ens@##1}{##2}}%

  \bbl@for\bbl@tempa\bbl@tempa{%    
    \expandafter\bbl@add\expandafter\bbl@tempc
      \expandafter{\bbl@tempa}\expandafter{}
  }

  \def\bbl@tempc{\bbl@ensure}

  \expandafter\bbl@add\expandafter\bbl@tempc
    \expandafter{\bbl@tempc}\expandafter{}

  \edef\x{%
    \endgroup
    \noexpand\@namedef{bbl@e@#2}{\the	oks@{\bbl@ens@fontenc}}%  }

\def\bbl@ensured@empty%
7.3 Setting up language files

\LdfInit The second version of \LdfInit macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before. At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save its name as the last called option, even if not loaded. Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions with the \let primitive. Therefore we store its current catcode and restore it later on.

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to \LdfInit is a control sequence. We do that by looking at the first token after passing \#2 through string. When it is equal to \@backslashchar we are dealing with a control sequence which we can compare with \@undefined. If so, we call \ldf@quit to set the main language, restore the category code of the @-sign and call \endinput. When \#2 was not a control sequence we construct one and compare it with \relax. Finally we check \originalTeX. 

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

\ldf@quit#1%  
\chardef\atcatcode=\catcode'@  
\catcode'@=11\relax  
\chardef\eqcatcode=\catcode'==  
\catcode'==12\relax  
\expandafter\if\expandafter\@backslashchar\string#2\@nil  
\expandafter\if\csname#2\endcsname\relax\else  
\ldf@quit{#1}%  
\fi  
\else  
\expandafter\if\expandafter\csname#2\endcsname\relax\else  
\ldf@quit{#1}%  
\fi  
\fi  
\let\bbl@screset\@empty  
\let\BabelStrings\bbl@opt@strings  
\let\BabelOptions\@empty  
\let\BabelLanguages\relax  
\ifx\originalTeX\@undefined  
\let\originalTeX\@empty  
\else  
\originalTeX  
\fi

\ldf@quit#1%  
\expandafter\main@language\expandafter{#1}%  
\catcode'@=\atcatcode \let\atcatcode\relax  
\catcode'==\eqcatcode \let\eqcatcode\relax  
\endinput}
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}%
  \bbl@afterlang
  \let\bbl@afterlang\relax
  \let\BabelModifiers\relax
  \let\bbl@screset\relax
  \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. Therefore they are turned into warning
messages in \LaTeX.
\onlypreamble\LdfInit
\onlypreamble\ldf@quit
\onlypreamble\ldf@finish

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}}

We also have to make sure that some code gets executed at the beginning of the
document.
\AtBeginDocument{%
  \expandafter\selectlanguage\expandafter{\bbl@main@language}}

### 7.4 Shorthands

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.
To add the character to the macros, we expand the original macros with the
additional character inside the redefine of the macros. Because \@sanitize
can be undefined, we put the definition inside a conditional.
\def\bbl@add@special#1{%
  \begingroup
  \def\do\noexpand\do\noexpand{\noexpand\do\noexpand}
  \def\@makeother\noexpand\@makeother\noexpand{\noexpand}\@makeother\noexpand
  \edef\x{\endgroup
    \def\noexpand\dospecials{\dospecials\do{#1}%
    \expandafter\ifx\csname @sanitize\endcsname\relax \else
    \def\noexpand\@sanitize{\@sanitize\@makeother#1}%
    \fi%
    \x}
The macro \x contains at this moment the following:
\endgroup\def\dospecials{\o{\d 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.

\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.

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

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

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

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, \langle level\rangle@group, \langle level\rangle@active and \langle next-level\rangle@active (except in system).
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\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 treatment 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
oexpand#1\noexpand\@undefined}\else\expandafter\let\csname bbl@oridef@@#2\endcsname#1\expandafter\edef\csname bbl@oridef@#2\endcsname{\let
oexpand\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. If the character is mathematically active when babel is loaded (for example ’) the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to "8000 a posteriori).

\ifx#1#3\relax\expandafter\let\csname normal@char#2\endcsname#3\else\bbl@info{Making #2 an active character}\ifnum\mathcode'#2="8000\@namedef{normal@char#2}{\textormath{#3}{\csname bbl@oridef@@#2\endcsname}}\else\@namedef{normal@char#2}{#3}\fi\fi

To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each
language file (except with KeepShorthandsActive). It is re-activate again at \begin{document}. We also need to make sure that the shorthands are active during the processing of the .aux file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of \cite{example}. Then we make it active (not strictly necessary, but done for backward compatibility).

\begin{verbatim}
\bbl@restoreactive{#2}\
\AtBeginDocument{%\
\catcode'#2\active\
\if@filesw\
\immediate\write\@mainaux{\catcode'\string#2\active}%\
\fi}\
\expandafter\bbl@add@special\csname#2\endcsname\
\catcode'#2\active\
\fi
\end{verbatim}

Now we have set \normal@char, we must define \active@char, to be executed when the character is activated. We define the first level expansion of \active@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 to start the search of a definition in the user, language and system levels (or eventually normal@char).

\begin{verbatim}
\let\bbl@tempa@firstoftwo\if\string^#2%\def\bbl@tempa{\noexpand\textormath}%\else\ifx\bbl@mathnormal@undefined\else\let\bbl@tempa\bbl@mathnormal\fi\fi\expandafter\edef\csname active@char#2\endcsname{\bbl@tempa{\noexpand\if@safe@actives\noexpand\expandafter\csname normal@char#2\endcsname\noexpand\else\noexpand\expandafter\csname bbl@doactive#2\endcsname\noexpand\fi}}%\bbl@csarg\edef{doactive#2}{\expandafter\noexpand\csname user@active#2\endcsname}%
\end{verbatim}

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

\begin{verbatim}
\active@prefix \normal@char
\end{verbatim}

(\verb+\active@char+ is one control sequence!).

\begin{verbatim}
\bbl@csarg\edef{active@#2}{%\noexpand\active@prefix\noexpand1%\expandafter\bbl@csarg\edef{doactive#2}{%\noexpand\active@prefix\noexpand1%\expandafter\bbl@csarg\edef{normal@#2}{%\noexpand\active@prefix\noexpand1%\expandafter\bbl@csarg\edef{active@#2}{%\noexpand\active@prefix\noexpand1%\expandafter\bbl@csarg\edef{doactive#2}{%\noexpand\active@prefix\noexpand1%
\end{verbatim}

52
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.

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.

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (‘’) active we need to change \pr@m@s as well. Also, make sure that a single ‘ in math mode ‘does the right thing’. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

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

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package and the end of the \ldf.

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.

```latex
\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}
```

The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it protects the active character whenever `\protect` is not `\@typeset@protect`.

```latex
\def\active@prefix#1{\ifx\protect\@typeset@protect\else\noexpand#1\fi\expandafter\@gobble\fi}
```

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⟩`.

```latex
\newif\if@safe@actives\@safe@activesfalse
```

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.

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

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

```latex
\def\bbl@activate#1{\bbl@withactive\expandafter\let\expandafter}#1\csname bbl@active@\string#1\endcsname}
\def\bbl@deactivate#1{\bbl@withactive\expandafter\let\expandafter}#1\csname bbl@normal@\string#1\endcsname}
```

These macros have two arguments. They use one of their arguments to build a control sequence from.

```latex
\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.

\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.

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

\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.
First note that this is user level, and then initialize and activate the character for
use as a shorthand character (i.e., it's active in the preamble). Languages can
deactivate shorthands, so a starred version is also provided which activates them
always after the language has been switched.

\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.

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

\aliasshorthand
First the new shorthand needs to be initialized,
Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands{"}{/} is \active@prefix /\active@char/, so we still need to let the lattest to \active@char".

\expandafter\let\csname active@char\string#2\endcsname
\expandafter\let\csname normal@char\string#2\endcsname
\bbl@activate{#2}

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

\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@ne#1@nnil}
\DeclareRobustCommand\shorthandoff{% \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}
\def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2@nnil}

\bbl@switch@sh The macro \bbl@switch@sh takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of \bbl@switch@sh. But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as \active@char" should exist. Switching off and on is easy – we just set the category code to ‘other’ (12) and \active. With the starred version, the original catcode and the original definition, saved in @initiate@active@char, are restored.
\def\bbl@switch@sh@#1#2{%\ifx#2@nnil\else
  \@ifundefined{bbl@active@\string#2}{\bbl@error{I cannot switch \string#2 on or off--not a shorthand}%,\%
  This character is not a shorthand. Maybe you made\%
  a typing mistake? I will ignore your instruction}}%
Note the value is that at the expansion time, eg, in the preamble shorthands are usually deactivated.

```latex
\def\babelshorthand{(active@prefix\babelshorthand\bbl@putsh}
\def\bbl@putsh#1{\@ifundefined{bbl@active@\string#1}{\bbl@putsh@i#1\@empty\@nnil}{\csname bbl@active@\string#1\endcsname}}
\def\bbl@putsh@i#1#2\@nnil{\csname\languagename @sh@\string#1@%\ifx\@empty#2\else\string#2@\fi\endcsname}
\ifx\bbl@opt@shorthands\@nnil\else
\let\bbl@s@initiate@active@char\initiate@active@char
\let\bbl@s@switch@sh\bbl@switch@sh
\let\bbl@s@activate\bbl@activate
\let\bbl@s@deactivate\bbl@deactivate
\fi
```

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.

```latex
\def\bbl@prim@s\bbl@pr@ms\bbl@prim@s\bbl@pr@ms
\def\prime{\futurelet\@let@token\bbl@pr@ms}
\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{...

```
usually the ~ is active and expands to \penalty\@M\. When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

\initiateactivechar{~}
\declareshorthand{system}{~}{\leavevmode
nobreak} \bblactivate{~}
\OT1dqpos The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.
\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

7.5 Language attributes

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

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.
\newcommand\languageattribute[2]{% 
def\bbl@tempc[#1]{% \bbl@fixname\bbl@tempc \bbl@iflanguage\bbl@tempc{\bbl@loopx\bbl@attr(#2){% 

We want to make sure that each attribute is selected only once; therefore we store the already selected attributes in \bbl@known@attrs. When that control sequence is not yet defined this attribute is certainly not selected before.
\ifx\bbl@known@attrs\@undefined
\in@false \else
\expandafter\inargsin@{,\bbl@tempc-\bbl@attr},{,\bbl@known@attrs,} \fi

Now we need to see if the attribute occurs in the list of already selected attributes.
When the attribute was in the list we issue a warning; this might not be the users intention.

```latex
\ifin@
    \bbl@warning{%
        You have more than once selected the attribute `\bbl@attr' 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.

```latex
\edef\bbl@tempa{\noexpand\bbl@add@list \noexpand\bbl@known@attrs{\bbl@tempc-\bbl@attr}}\bbl@tempa\edef\bbl@tempa{\bbl@tempc-\bbl@attr}\expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes{\csname\bbl@tempc @attr@\bbl@attr\endcsname}{\@attrerr{\bbl@tempc}{\bbl@attr}}%}
```

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

```latex\@onlypreamble\languageattribute
The error text to be issued when an unknown attribute is selected.
```
The we need to check the list of known attributes.

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

When we're this far \fin@ 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’.

\fin@
\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\fi

\bbl@ifknown@atrib
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@atrib#1#2{\let\bbl@tempa@\@secondoftwoThen we loop over the list of known attributes, trying to find a match.
\bbl@loopx\bbl@tempb{#2}{\expandafter\in@\expandafter{\expandafter,,\bbl@tempb,}{,#1,}%;\ifin@\let\bbl@tempa@\@firstoftwo\else\fi};\expandafter\bbl@clear@trib\bbl@tempa@}%

Finally we execute \bbl@tempa@.

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

\def\bbl@clear@trib{%\ifx\bbl@attributes\@undefined\else\bbl@loopx\bbl@tempa{\bbl@attributes}{\expandafter\bbl@clear@trib\bbl@tempa,}\%\expandafter\bbl@clear@trib\bbl@tempa,\}%
7.6 Support for saving macro definitions

To save the meaning of control sequences using \texttt{\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 \texttt{\selectlanguage} and \texttt{\originalTeX}). Note undefined macros are not undefined any more when saved – they are \texttt{\relax}’ed.

\begin{verbatim}
\def\babel@beginsave{\advance\babel@savecnt\@ne}
\end{verbatim}

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

\begin{verbatim}
\def\bbl@frenchspacing{\let\bbl@nonfrenchspacing\relax}
\end{verbatim}
7.7 Short tags

\babeltags \text{This macro is straightforward. After zapping spaces, we loop over the list and define the macros \text{⟨tag⟩} and ⟨tag⟩. Definitions are first expanded so that they don’t contain \texttt{csname} but the actual macro.}

\def\babeltags#1{% 
  \edef\bbl@tempa{\zap@space#1 \@empty} 
  \def\bbl@tempb##1=##2\@@{% 
    \edef\bbl@tempc{\noexpand\newcommand\expandafter\noexpand\csname ##1\endcsname{\noexpand\protect\expandafter\noexpand\csname otherlanguage\endcsname{##2}}} 
    \noexpand\newcommand\expandafter\noexpand\csname\text##1\endcsname{\noexpand\foreignlanguage{##2}}} 
  \bbl@tempc} 
  \bbl@for\bbl@tempa\bbl@tempa{% 
    \expandafter\bbl@tempb\bbl@tempa\@@}}

7.8 Hyphens

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

\@onlypreamble\babelhyphenation 
\AtEndOfPackage{% 
  \newcommand\babelhyphenation[2][\@empty]{% 
    \ifx\bbl@hyphenation@\relax 
      \let\bbl@hyphenation@\@empty 
    \fi 
    \ifx\@empty#1% 
      \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}% 
    \else 
      \edef\bbl@tempb{\zap@space#1 \@empty}% 
      \bbl@for\bbl@tempa\bbl@tempb{% 
        \bbl@fixname\bbl@tempa 
        \bbl@iflanguage\bbl@tempa{\bbl@csargs\protected@edef{hyphenation@\bbl@tempa}{% 
          \ifundefined{\bbl@hyphenation@\bbl@tempa}\@empty% 
          \@empty} 
    \fi 
    \ifx\@empty#1% 
      \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space\bbl@hyphenation@\space#2}% 
    \else 
      \edef\bbl@tempb{\zap@space#1 \@empty}% 
      \bbl@for\bbl@tempa\bbl@tempb{% 
        \bbl@fixname\bbl@tempa 
        \bbl@iflanguage\bbl@tempa{\bbl@csargs\protected@edef{hyphenation@\bbl@tempa}{% 
          \ifundefined{\bbl@hyphenation@\bbl@tempa}\@empty% 
          \@empty}}} 

63
This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hspace{0pt plus 0pt}`\textsuperscript{27}.

\def\bbl@allowhyphens{\ifvmode\else\nobreak\hspace{0pt}\fi}
\def\bbl@t@one{T1}
\def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

Macros to insert common hyphens. Note the space before `\@` in `\babelhyphen`.

Instead of protecting it with `\DeclareRobustCommand`, which could insert a `\relax`, we use the same procedure as shorthands.

\newcommand\babelnullhyphen{\char\hyphenchar\font}
\def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
\def\bbl@hyphen{\@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphen@i#1#2{\if@undefined{bbl@hy@#1#2\@empty}{\csname bbl@usehyphen\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 “(\text{-suffix})”. `\nobreak` is always preceded by `\leavevmode`, in case the shorthand starts a paragraph.

\def\bbl@usehyphen#1{\leavevmode\ifdim\lastskip>\z@\mbox{#1}\nobreak\else\nobreak#1\fi}
\def\bbl@@usehyphen#1{\leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

Finally, we define the hyphen “types”. Their names will not change, so you may use them in \textit{ldf}'s.

\def\bbl@usehyphen{i}{\nobreak}\else
\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@usehyphen{i}{\nobreak}\else
\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@nullhyphen{\leavevmode\ifnum\hyphenchar\font=\m@ne\bblnullhyphen\else
\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@hyphenchar{\ifnum\hyphenchar\font=\m@ne\bblnullhyphen
\else\nobreak\mbox{#1}\elset\nobreak#1\fi}

\def\bbl@usehyphen{i}{\nobreak}\else
\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@nullhyphen{\leavevmode\ifnum\hyphenchar\font=\m@ne\bblnullhyphen
\else\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@hyphenchar{\ifnum\hyphenchar\font=\m@ne\bblnullhyphen
\else\nobreak\mbox{#1}\elset\nobreak#1\fi}

\def\bbl@usehyphen{i}{\nobreak}\else
\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@nullhyphen{\leavevmode\ifnum\hyphenchar\font=\m@ne\bblnullhyphen
\else\nobreak\mbox{#1}\elset\nobreak#1\fi}
\def\bbl@hyphenchar{\ifnum\hyphenchar\font=\m@ne\bblnullhyphen
\else\nobreak\mbox{#1}\elset\nobreak#1\fi}

\begin{itemize}
\item `\nobreak` is always preceded by `\leavevmode`, in case the shorthand starts a paragraph.
\end{itemize}

\footnote{\LaTeX{} begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.}
For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.

7.9 Multiencoding strings

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

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

The second one. We need to patch \@uclclist, but it is done once and only if \SetCase is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact \@uclclist is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually \reserved@a), we pass it as argument to \bbl@uclc. The parser is restarted inside \lang@bbl@uclc because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

```latex
% \let\bbl@tolower\@empty\bbl@toupper\@empty
%
```

and starts over (and similarly when lowercasing).
The following package options control the behaviour of \SetString.

\let\bbl@opt@strings@nnil % accept strings=value
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\def\BabelStringsDefault{generic}

Main command This is the main command. With the first use it is redefined to
omit the basic setup in subsequent blocks. We make sure strings contain actual
letters in the range 128-255, not active characters.
\@onlypreamble\StartBabelCommands
\def\StartBabelCommands{%
  \begingroup
  \bbl@recatcode{11}%
  ⟨⟨Macros local to BabelCommands⟩⟩
  \def\bbl@provstring##1##2{%
    \providecommand##1{##2}%
    \bbl@toglobal##1}
  \global\let\bbl@scafter@empty
  \let\StartBabelCommands\bbl@startcmds
  \ifx\BabelLanguages\relax
    \let\BabelLanguages\CurrentOption
  \fi
  \begingroup
  \let\bbl@screset@nnil % local flag - disable 1st stopcommands
  \StartBabelCommands{}
  \if\bbl@screset\@nil
    \bbl@usehooks{stopcommands}{}%
  \fi
  \endgroup
  \begingroup
  \@ifstar
    \if\bbl@opt@strings\@nil
      \let\bbl@opt@strings\BabelStringsDefault
    \fi
    \bbl@startcmds@i}\
  \bbl@startcmds@i
  \def\bbl@startcmds@i#1#2{%
    \edef\bbl@L{\zap@space#1 \@empty}%
    \edef\bbl@G{\zap@space#2 \@empty}%
    \bbl@startcmds@ii}

Parse the encoding info to get the label, input, and font parts.
Select the behaviour of \SetString. There are two main cases, depending on if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

\newcommand\bbl@startcmds@ii[1][\empty]{{}
\let\SetString@gobbletwo
\let\bbl@stringdef@gobbletwo
\let\AfterBabelCommands@gobble
\if\empty#1%
  \def\bbl@sc@label{generic}%
  \def\bbl@encstring##1##2{%  
    \ProvideTextCommandDefault##1{##2}%  
    \bbl@toglobal##1%
    \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%  
  \let\bbl@sctest\in@true
\else
  \let\bbl@sc@charset\space % <- zapped below
  \let\bbl@sc@fontenc\space % <- " "
  \def\bbl@tempa##1=##2@nil{%  
    \bbl@csarg\edef{sc@\zap@space##1 \empty}{##2 }%  
  }\bbl@for\bbl@tempb{label=#1}{\expandafter\bbl@tempa\bbl@tempb@nil}%  
  \def\bbl@tempa##1 ##2{% space -> comma
    ##1%  
    \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%  
  \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%  
  \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%  
  \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%  
  \def\bbl@encstring##1##2{  
    \bbl@for\bbl@tempc\bbl@sc@fontenc{  
      \@ifundefined{T@\bbl@tempc}%  
        {}%  
        {\ProvideTextCommand##1\bbl@tempc{##2}%  
          \bbl@toglobal##1%  
          \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}}}%  
  \def\bbl@sctest{%  
    \@expandtwoargs
    \in@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%  
  }\fi
\if\empty#2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa#2@fi)%
  \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
  \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
  \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
  \def\bbl@encstring##1##2{%  
    \bbl@for\bbl@tempc\bbl@sc@fontenc{  
      \@ifundefined{T@\bbl@tempc}%  
        {}%  
        {\ProvideTextCommand##1\bbl@tempc{##2}%  
          \bbl@toglobal##1%  
          \expandafter\bbl@afterfi\bbl@encstring##1##2%}
    }%}
  \def\bbl@sctest{%  
    \@expandtwoargs
    \in@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%  
  }\fi
\if\empty\bbl@opt@strings\@nil % ie, no strings key -> defaults  
\else\if\empty\bbl@opt@strings\relax % ie, strings=encoded  
  \let\AfterBabelCommands\bbl@aftercmds
  \let\SetString\bbl@setstring
  \let\bbl@stringdef\bbl@encstring
  \else % ie, strings=value  
    \let\bbl@sctest
  \fi
  \let\AfterBabelCommands\bbl@aftercmds
  \let\SetString\bbl@setstring
There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \langle group\rangle \langle language\rangle is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing. The macro \bbl@forforlang loops \bbl@L but its body is executed only if the value is in \BabelLanguages (inside babel) or \date\langle language\rangle is defined (after babel has been loaded). There are also two version of \bbl@forforlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded).

Now we define commands to be used inside \StartBabelCommands.

**Strings** The following macro is the actual definition of \SetString when it is "active"
First save the "switcher". Create it if undefined. Strings are defined only if undefined (i.e., like \providescommand). With the event \stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

\def\bbl@setstring#1#2{% 
\bbl@forlang\bbl@tempa{% 
\edef\bbl@LC{\bbl@tempa\expandafter\@gobble\string#1}% 
@ifundefined{\bbl@LC}{% eg, \germanchaptername 
 \global\expandafter \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter}\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}{% 
\expandafter\bbl@stringdef \csname\bbl@LC\endcsname{\BabelString}}% 
}{}
\def\BabelString{#2}\
\bbl@usehooks{stringprocess}\{}% 
\expandafter\bbl@stringdef 
\csname\bbl@LC\endcsname{\BabelString}}%

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

\ifx\bbl@opt@strings\relax
\def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
\bbl@patchuclc
\let\bbl@encoded\relax
\def\bbl@encoded@uclc#1{% 
@@inmathwarn#1%
\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
\expandafter\ifx\csname ?\string#1\endcsname\TextSymbolUnavailable\relax
\else
\csname ?\string#1\endcsname
\fi
\else
\csname \string#1\endcsname
\fi
\else
\def\bbl@scset#1#2{\def#1{#2}}
\fi

Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current definition is somewhat complicated because we need a count, but \count@ is not under our control (remember \SetString may call hooks).

\def\SetStringLoop##1##2{% 
\def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
\count@\z@
\bbl@loop\bbl@tempa{#2}{% 
\advance\count@@\@ne
\toks@\bbl@tempa{% 
\edef\bbl@tempb{% 
\bbl@templ{\romannumeral\count@}{\the\toks@}{\the\count@}{\relax}% 
\expandafter\SetString\bbl@tempb}}% 
\expandafter\noexpand\csname\bbl@LC\endcsname{\expandafter\bbl@tempb}}}%
Delaying code  Now the definition of \AfterBabelCommands when it is activated.

\begin{verbatim}
def\bbl@aftercmds#1{\toks@\expandafter{\bbl@scafter#1}\xdef\bbl@scafter{\the\toks@}}
\end{verbatim}

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

\begin{verbatim}
\newcommand\SetCase[3][]{\bbl@patchuclc\bbl@forlang\bbl@tempa{\expandafter\bbl@encstring\csname\bbl@tempa @bbl@uc\endcsname{##2}\expandafter\bbl@encstring\csname\bbl@tempa @bbl@uc\endcsname{##3}}}\newcommand\SetHyphenMap[1]{\bbl@forlang\bbl@tempa{\expandafter\bbl@stringdef\csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}
\end{verbatim}

Macros to deal with case mapping for hyphenation. To decide if the document is monolingual or multilingual, we make a rough guess – just see if there is a comma in the languages list, built in the first pass of the package options.

\begin{verbatim}
(#{ Macros local to BabelCommands}) ≡ \newcommand\BabelLower[2][]{\ifnum\lccode#1=#2\else\babel@savevariable{\lccode#1}\lccode#1=#2\relax\fi}
\newcommand\BabelLowerMM[4][]{\@tempcnta=#1\relax\@tempcntb=#4\relax\def\bbl@tempa{\ifnum\@tempcnta>#2\else\@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}\advance\@tempcnta#3\relax\expandafter\bbl@tempa\fi}}\newcommand\BabelLowerMO[4][]{\@tempcnta=#1\relax\def\bbl@tempa{\ifnum\@tempcnta>#2\else\@expandtwoargs\BabelLower{\the\@tempcnta}{#4}\advance\@tempcnta\relax\expandafter\bbl@tempa\fi}}
\end{verbatim}

There are 3 helper macros which do most of the work for you.
The following package options control the behaviour of hyphenation mapping.

\DeclareOption{hyphenmap=off}{\chardef\bbl@hymapopt\z@}
\DeclareOption{hyphenmap=first}{\chardef\bbl@hymapopt\@ne}
\DeclareOption{hyphenmap=select}{\chardef\bbl@hymapopt\tw@}
\DeclareOption{hyphenmap=other}{\chardef\bbl@hymapopt\thr@@}
\DeclareOption{hyphenmap=other*}{\chardef\bbl@hymapopt4\relax}

Initial setup to provide a default behaviour if \texttt{hyphenmap} is not set.

\AtEndOfPackage{% 
\ifx\bbl@hymapopt\@undefined 
\@expandtwoargs\in@{,}{\bbl@language@opts} 
\chardef\bbl@hymapopt\ifin@4\else\@ne\fi 
\fi}

\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 \texttt{\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}

\set@low@box
\save@sf@q

\section*{7.10 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 \texttt{\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}

\section*{7.11 Making glyphs available}
This section makes a number of glyphs available that either do not exist in the \texttt{OT1} encoding and have to be ‘faked’, or that are not accessible through \texttt{T1enc.def}.

\subsection*{7.11.1 Quotation marks}
\texttt{\quotedblbase} In the \texttt{T1} encoding the opening double quote at the baseline is available as a separate character, accessible via \texttt{\quotedblbase}. In the \texttt{OT1} encoding it is not available, therefore 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 \texttt{OT1} or \texttt{T1} is used this glyph can still be typeset.
\ProvideTextCommandDefault{\quotedblbase}{% 
UseTextSymbol{OT1}{\quotedblbase}}

\texttt{\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}}

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

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

\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}}

\section{Letters}

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

\DeclareTextCommand{\ij}{OT1}{%
Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\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).

\SS For the T1 encoding \SS is defined and selects a specific glyph from the font, but for other encodings it is not available. Therefore we make it available here.

\glq The ‘german’ single quotes.
\grq

7.11.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode.
The definition of \textquoteleft depends on the font encoding. With T1 encoding no extra kerning is needed.

\textquoteleft
The 'german' double quotes.

\textquotedblleft
The 'french' single guillemets.

\textquotedblright
The 'french' double guillemets.

\textquotesingle
The 'french' single quotes.
7.11.4 Umlauts and tremas

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

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

\umlautlow
\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 \textit{dimen} register.

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

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

For all vowels we declare \" to be a composite command which uses \bbl@umlaut or \bbl@umlaut to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package fontenc with option OT1 is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but babel sets them for all languages – you may want to redefine
\umlauta and/or \umlaute for a language in the corresponding ldf (using the babel switching mechanism, of course).

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

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

\ifx\english\undefined
\chardef\english\z@
\fi
\mainlanguage{english}

Now we load definition files for engines.

\ifcase\engine\or
\input luababel.def
\or
\input xebabel.def
\fi

8 The kernel of Babel (only \LaTeX)

8.1 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}
\if\fmtname\format
\else
\def\format{LaTeX2e}
\if\fmtname\format
\else
\aftergroup\endinput
\fi
\fi

8.2 Cross referencing macros

The \LaTeX book states:
The key argument is any sequence of letters, digits, and punctuation symbols; upper- and lowercase letters are regarded as different.

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

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

The only way to accomplish this in most cases is to use the trick described in the 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’.

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

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

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

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

\@testdef An internal \LaTeX macro used to test if the labels that have been written on the .aux file have changed. It is called by the \enddocument macro. This macro needs to be completely rewritten, using \meaning. The reason for this is that in some cases the expansion of \#1@\#2 contains the same characters as the \#3; but the character codes differ. Therefore \LaTeX keeps reporting that the labels may have changed.
Now that we made sure that `@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’.

```latex
\begin{verbatim}
\def\@testdef#1#2#3{% 
  \@safe@activetrue
  \let\expandafter\@expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
  \@safe@activesfalse

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

  \newl@bel{#3}
  \@safe@activesfalse

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

  \begin{verbatim}
  \ifx\bbl@tempa\relax
    \else
      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}\
  \fi
  \end{verbatim}
  \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}
  \ifx\bbl@tempa\bbl@tempb
    \else
      \@tempswatrue
  \fi
}
\end{verbatim}
```

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

```latex
\begin{verbatim}
\begin{verbatim}
\edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}
\ifx\bbl@tempa\bbl@tempb
  \else
    \@tempswatrue
\fi
\end{verbatim}
\end{verbatim}
```

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 \texttt{\def} here instead of \texttt{\bbl@redefine} because \texttt{\org@@citex} is
already defined and we don't want to overwrite that definition (it would result in
parameter stack overflow because of a circular definition).
(Recent versions of natbib change dynamically \texttt{\@citex}, so PR4087 doesn't seem
fixable in a simple way. Just load natbib before.)
\def\@citex[#1][#2]#3{%\@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
\org@@citex[#1][#2]{\@tempa}{}%
}}

The package cite has a definition of \texttt{\@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 \texttt{\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}
\nocite
\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 \texttt{\@safe@activestrue} is in effect. This switch
needs to be reset inside the \texttt{\hbox} which contains the citation label. In order to
determine during .aux file processing which definition of \texttt{\bibcite} is needed we
define \texttt{\bibcite} in such a way that it redefines itself with the proper definition.
\bbl@redefine\bibcite%
We call \texttt{\bbl@cite@choice} to select the proper definition for \texttt{\bibcite}. This new
definition is then activated.
\bbl@cite@choice
\bibcite
\bbl@bibcite
The macro \texttt{\bbl@bibcite} holds the definition of \texttt{\bibcite} needed when neither
natbib nor cite is loaded.
\def\bbl@bibcite#1#2{%\@safe@activestrue\org@bibcite[#1]{\@safe@activesfalse#2}}%
\bibcite
\bbl@cite@choice
The macro \texttt{\bbl@cite@choice} determines which definition of \texttt{\bibcite} is needed.
\def\bbl@cite@choice{%
First we give \texttt{\bibcite} its default definition.
\global\let\bibcite\bbl@bibcite
Then, when natbib is loaded we restore the original definition of \texttt{\bibcite}.
\@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
For cite we do the same.

Make sure this only happens once.

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

8.3 Marks

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

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

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

Next, we store the argument to \markright in the scratch token register, 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.

Then we define a temporary control sequence using \edef.
When \bbl@tempa is executed, only \languagename will be expanded, because of the way the token register was filled.

\begin{verbatim}
\noexpand\org@markright{\the\toks@}\% 
\bbl@tempa
\bbl@tempc
\end{verbatim}

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

\begin{verbatim}
\ifx\@mkboth\markboth
\def\bbl@tempc{\let\@mkboth\markboth}
\else
\def\bbl@tempc{}
\fi
\end{verbatim}

Now we can start the new definition of \markboth

\begin{verbatim}
\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\% 
\bbl@tempc
\end{verbatim}

and copy it to \@mkboth if necessary.

8.4 Preventing clashes with other packages

8.4.1 ifthen

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

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:
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.

\begin{verbatim}
\@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. When the package wasn't loaded we do nothing.
\@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}%
}%
}
}
\end{verbatim}

8.4.2 varioref

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

\begin{verbatim}
\AtBeginDocument{%
@ifpackageloaded{varioref}{%
\bbl@redefine\@@vpageref#1[#2]#3{%
The same needs to happen for \vrefpagenum.
\bbl@redefine\vrefpagenum#1#2{%
\@safe@activestrue
\org@vrefpagenum(#1){#2}%
\@safe@activesfalse}%
\end{verbatim}

82
The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref` to call `\org@ref` instead of `\ref`. The disadvantage of this solution is that whenever the definition of `\Ref` changes, this definition needs to be updated as well.

```latex
\expandafter\def\csname Ref \endcsname#1{\protect@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
```

8.4.3 hhline

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

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

```latex
\AtEndOfPackage{%
\AtBeginDocument{%
@ifpackageloaded{hhline}{%\expandafter\ifx\csname normal@char\string: endcsname\endcsname\relax{\else}{%\expandafter\ifx\csname normal@char\string: endcsname\endcsname\relax
\@currname{hhline}\input{hhline.sty}\makeatother}%}{}%}
```

8.4.4 hyperref

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

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

8.4.5 fancyhdr

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

\begin{verbatim}
\DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%  
  \lowercase{\foreignlanguage{#1}}}
\end{verbatim}

\textbf{\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.

\begin{verbatim}
\def\substitutefontfamily#1#2#3{\lowercase{\immediate\openout15=#1#2.fd\relax}%
  \immediate\write15{\string\ProvidesFile{#1#2.fd}[	he\year/\two@digits{\the\month}/\two@digits{\the\day}]
  \space generated font description file!^^J
  \string\DeclareFontFamily{#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}
\end{verbatim}

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

\textbf{8.5 Encoding issues}

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

\textbf{\ensureascii}

\begin{verbatim}
\newcommand{\BabelNonASCII}{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,}
\let\org@TeX=\TeX{}
\let\org@LaTeX=\LaTeX{}
\let\ensureascii=\@firstofone
\AtBeginDocument{\in@false
  \bbl@loopx\bbl@tempa{\BabelNonASCII}{ is there a non-ascii enc?\ifin@
    \edef{\bbl@tempb}{\langle,\bbl@tempa,enc.def,\rangle,\@filelist,}\lowercase{\expandafter{\expandafter{\in\bbl@tempb}}\ifin@\% if a non-ascii has been loaded\def{\ensureasciiset}{\langle,\fontencoding{OT1},\selectfont,\rangle}\fi
  }%\fi}
\end{verbatim}

84
Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which latin fontencoding to use.

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.

\texttt{\AtEndOfPackage{\edef\latinencoding{EU\ifcase\bbl@engine\or2\or1\fi}}}\}

But this might be overruled with a later loading of the package fontenc. Therefore we check at the execution of \begin{document} whether it was loaded with the T1 option. The normal way to do this (using @@ifpackageloaded) is disabled for this package. Now we have to revert to parsing the internal macro \@filelist which contains all the filenames loaded.

Then we can define the command \texttt{\latintext} which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

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.
8.6 Local Language Configuration

\loadlocalcfg At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension .cfg. For instance the file norsk.cfg will be loaded when the language definition file norsk.ldf is loaded. For plain-based formats we don’t want to override the definition of \loadlocalcfg from plain.def.

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⟩

9 Internationalizing \LaTeX 2.09

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. Therefore this part of the code can optionally be included in babel.def by specifying the docstrip option 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.

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.
Table 1: How to determine the main document style

<table>
<thead>
<tr>
<th>Document Type</th>
<th>\chapter and \opening Macros</th>
</tr>
</thead>
<tbody>
<tr>
<td>article</td>
<td>both the \chapter and \opening macros are undefined</td>
</tr>
<tr>
<td>report and book</td>
<td>the \chapter macro is defined and the \opening is undefined</td>
</tr>
<tr>
<td>letter</td>
<td>the \chapter macro is undefined and the \opening is defined</td>
</tr>
</tbody>
</table>

\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}%
\fi

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

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:

\ifundef\figname
\def\fnum@figure{\figname{} \thefigure}%
\fi

\ifundef\tabname
\def\fnum@table{\tabname{} \thetable}%
\fi

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\(^{28}\).

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

\ifundef\contname
{\section*{\contname@mkboth{\uppercase{\contname}}{\uppercase{\contname}}}%
\starttoc{toc}}%
\fi

\ifundef\listfigurename%
{\def\tableofcontents{%
\section*{\contname@mkboth{\uppercase{\contname}}{\uppercase{\contname}}}%
\starttoc{toc}}}%
\fi

\footnote{\(^{28}\)A fact that was pointed out to me by Nico Poppelier and was already used in Piet van Oostrum’s document style option nl.}
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}}{}
\@ifundefined{indexname}{\def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi\columnseprule\z@\columnsep35pt\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]{\ifnum\c@secnumdepth\@ne\refstepcounter{part}\addcontentsline{toc}{part}{\thepart\hspace{1em}#1}\else\addcontentsline{toc}{part}{#1}\fi{\parindent\z@\raggedright\vspace{\z@}\parindent\z@\nobreak\vspace{3ex}\@afterheading}}}}{}

This is all that needs to be done for the article style.
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
  \if\chapter*{\contentsname\@mkboth{\uppercase{\contentsname}}{\uppercase{\contentsname}}}\
  \@starttoc{toc}\
  \@ifundefined{listfigurename}\
  {\def\listoffigures{\@restonecolfalse
    \if@twocolumn\@restonecoltrue\onecolumn
    \if\chapter*{\listfigurename\@mkboth{\uppercase{\listfigurename}}{\uppercase{\listfigurename}}}\
    \@starttoc{lof}\
    \@ifundefined{listtablename}\
    {\def\listoftables{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \if\chapter*{\listtablename\@mkboth{\uppercase{\listtablename}}{\uppercase{\listtablename}}}\
      \@starttoc{lot}\
      \@ifundefined{bibname}\
      {\def\thebibliography#1{\chapter*{\bibname
        \@mkboth{\uppercase{\bibname}}{\uppercase{\bibname}}}\
        \list{[\arabic{enumi}]}{\settowidth\labelwidth{[#1]}\
        \leftmargin\labelwidth \advance\leftmargin\labelsep
        \@mkboth{\uppercase{\bibname}}{\uppercase{\bibname}}}\
        \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@ plus 3pt\parindent\@let\item@idxitem}}\
      \@ifundefined{abstractname}\
      {\def\abstract{\null\vfil

Here is the abstract environment:
And last but not least the \chapter, \appendix and \part macros.

\ifundefined{chaptername}{\def\@chapapp{\chaptername}}{}
\ifundefined{appendixname}{}
\ifundefined{partname}{}
\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}{}
\ifnum \c@secnumdepth >-2\relax
\refstepcounter{part}
\addcontentsline{toc}{part}{\thechapter{\Alph{chapter}}}{}
\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

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
\fi

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

## 10 Multiple languages

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.

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

\bbl@iflanguage executes code only of the language exists. Otherwise raises and error.

The argument of \bbl@fixname has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intended to fix a long-standing bug when \foreignlanguage and the like appear in a \MakeXXXcase. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped).

\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.
10.1 Selecting the language

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.

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

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

\def\bbl@language@stack{}

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

\bbl@push@language \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{\language\bbl@language@stack}%
}

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

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

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

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

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

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 selectlanguage \endcsname#1{%
  \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel=\tw@i
  \bbl@set@language
}

\bbl@set@language The macro \bbl@set@language takes care of switching the language environment and of writing entries on the auxiliary files. For historical reasons, language names can be either language of \language. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in \language are not well
defined. The list of auxiliary files can be extended by redefining \BabelContentsFiles, but make sure they are loaded inside a group (as aux, toc, lof, and lot do) or the last language of the document will remain active afterwards.

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

\begin{verbatim}
def\BabelContentsFiles{toc,lof,lot}
def\bbl@set@language#1{%
edef\languagename{%\ifnum\escapechar=\string#1@empty
\else\string#1@empty\fi}%\select@language\languagename%
\expandafter\ifx\csname date\languagename\endcsname\relax\else
\expandafter\if\csname date\languagename\endcsname\relax\else
\if@filesw
\protected@write\@auxout{}{\string\select@language\languagename}%
\bbl@for\bbl@tempa\BabelContentsFiles{%
\addtocontents\bbl@tempa{\xstring\select@language\languagename}}%
\bbl@usehooks{write}{}%
\fi\fi}
\def\select@language#1{%
\ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
\edef\languagename{#1}%
\bbl@fixname\languagename
\bbl@iflanguage\languagename{%
\expandafter\if\csname date\languagename\endcsname\relax
\bbl@error
{Unknown language ‘#1’. Either you have\%
misspelled its name, it has not been installed,\%
or you requested it in a previous run. Fix its name,\%
install it or just rerun the file, respectively}\%
{You may proceed, but expect unexpected results}\%
\else
\let\bbl@select@type\z@
\bbl@switch\languagename%
\expandafter\if\csname date\languagename\endcsname\relax
\bbl@error%
{Unknown language ‘#1’. Either you have\%
misspelled its name, it has not been installed,\%
or you requested it in a previous run. Fix its name,\%
install it or just rerun the file, respectively}\%
{You may proceed, but expect unexpected results}\%
\else
\bbl@select@type#1%
\ifcase\bbl@select@type%
\bbl@error
{Unknown language ‘#1’. Either you have\%
misspelled its name, it has not been installed,\%
or you requested it in a previous run. Fix its name,\%
install it or just rerun the file, respectively}\%
{You may proceed, but expect unexpected results}\%
\else
\select@language#1%
\fi%
\fi}
% A bit of optimization:
def\select@language@x#1{%
\ifcase\bbl@select@type%
\bbl@error
{Unknown language ‘#1’. Either you have\%
misspelled its name, it has not been installed,\%
or you requested it in a previous run. Fix its name,\%
install it or just rerun the file, respectively}\%
{You may proceed, but expect unexpected results}\%
\else
\select@language#1%
\fi%}
\end{verbatim}

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 redefine \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras⟨lang⟩ command at definition time by expanding the \csname primitive.

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

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

```
\def\bbl@switch#1{%
\originalTeX
\expandafter\def\expandafter\originalTeX\expandafter{%
\csname noextras\#1\endcsname
\let\originalTeX@empty
\babel@beginsave}%
\bbl@usehooks{afterreset}{}% 
\languageshorthands{none}%
\ifcase\bbl@select@type
\csname captions\#1\endcsname
\csname date\#1\endcsname
\fi
\bbl@usehooks{beforeextras}{}%
\csname extras\#1\endcsname\relax
\bbl@usehooks{afterextras}{}%
\ifcase\bbl@hymapopt
\def\BabelLower##1##2{\lccode##1=##2\relax}%
\ifnum\bbl@hymapsel>4\else
\csname\languagename @bbl@hyphenmap\endcsname
\fi
\chardef\bbl@hymapopt\z@
\else
\ifnum\bbl@hymapsel>\bbl@hymapopt
\fi
\fi
\global\let\bbl@hymapsel\@cclv
\bbl@patterns{#1}%
\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\bbl@ifsamestring#1#2{%
\protected@edef\bbl@tempb{#1}%
\edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
\protected@edef\bbl@tempc{#2}%
\edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
\ifx\bbl@tempb\bbl@tempc
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\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{% 
  \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel=\thr@@\fi 
  \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 \foreignlanguage.

\long\def\otherlanguage+\endcsname#1{% 
  \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel=4\relax\fi 
  \foreignlanguage{#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”.

\long\def\endotherlanguage+\endcsname{
  \global\@ignoretrue \ignorespaces}

\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 \extrals command doesn't make any
\global changes. The coding is very similar to part of \selectlanguage.

\edef\foreignlanguage{% 
  \noexpand\protect \expandafter\noexpand\csname foreignlanguage \endcsname}

\expandafter\def\csname foreignlanguage \endcsname#1#2{% 
  \begingroup 
  \foreignlanguage{#1}% 
  #2% 
  \endgroup}

\foreignlanguage

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.

\edef\foreignlanguage{% 
  \edef\languagename{#1}% 
  \bbl@fixname\languagename 
  \bbl@iflanguage\languagename{ 
    \expandafter\ifx\csname date\languagename\endcsname\relax 
    \bbl@warning{You haven’t loaded the language \languagename\space yet\%%% 
      I’ll proceed, but expect unexpected results.\%%% 
      Reported}\fi} 
  \let\bbl@select@type=1% 
  \bbl@switch\bbl@fixname\languagename 
  \languagename 
  \bbl@iflanguage\languagename{ 
    \expandafter\ifx\csname date\languagename\endcsname\relax 
    \bbl@warning{You haven’t loaded the language \languagename\space yet\%%% 
      I’ll proceed, but expect unexpected results.\%%% 
      Reported}\fi} 
  \let\bbl@select@type=1% 
  \bbl@switch\expandafter{\languagename}}
This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default. It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

\let\bbl@hyphlist@empty
\let\bbl@hyphenation@relax
\let\bbl@pttnlist@empty
\let\bbl@patterns@relax
\def\bbl@patterns#1{%
\language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
\csname l@#1\endcsname
\edef\bbl@tempa{#1}%
\else
\csname l@#1:\f@encoding\endcsname
\edef\bbl@tempa{#1:\f@encoding}%
\fi\relax
@expandtwoargs\bbl@usehooks\patterns\{{#1}{\bbl@tempa}}%
@ifundefined{bbl@hyphenation@}{}{%
\begingroup
@expandtwoargs\in@{,\number\language,}{{#1}{\bbl@hyphlist}}%
@ifin@\else
@expandtwoargs\bbl@usehooks\hyphenation\{{#1}{\bbl@tempa}}%
@hyphenation{%
\bbl@hyphenation@
@ifundefined{bbl@hyphenation@#1}{}%
@empty
\{\csname bbl@hyphenation@#1\endcsname\}%
@edef\bbl@hyphlist\{{\bbl@hyphlist}\number\language,}%
@fi
@endgroup}%

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{%
\edef\languagename{#1}%
\bbl@fixname\languagename
\bbl@iflanguage\languagename{%\expandafter\patterns\expandafter\languagename\relax
\languageshorthands{none}\relax
\expandafter\ifx\csname\languagename\hyphenmins\endcsname\relax
\set@hyphenmins\tw@\thr@@\relax
\else
\expandafter\expandafter\expandafter\set@hyphenmins\csname\languagename\hyphenmins\endcsname\relax
\else
\expandafter\expandafter\expandafter\set@hyphenmins\csname\languagename\hyphenmins\endcsname\relax
\fi}
@endhyphenrules@empty
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 \langle lang\rangle hyphenmins is already defined this command has no effect.

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

\ProvidesLanguage
The identification code for each file is something that was introduced in LaTeX2ε. When the command \ProvidesFile does not exist, a dummy definition is provided temporarily. For use in the language definition file the command \ProvidesLanguage is defined by babel. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

\LdfInit
This macro is defined in two versions. The first version is to be part of the ‘kernel’ of babel, ie. the part that is loaded in the format; the second version is defined in babel.def. The version in the format just checks the category code of the ampersand and then loads babel.def. The category code of the ampersand is restored and the macro calls itself again with the new definition from babel.def

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

\begin{verbatim}
2133 \ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi
\end{verbatim}

### 10.2 Errors

\texttt{\@nolanerr} The babel package will signal an error when a document 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 \texttt{\language=0} in that case. In most formats that will be (US) English, but it might also be empty.

\texttt{\@nopatterns} 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 \texttt{\PackageError} it must be \LaTeX, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

\begin{verbatim}
2134 \edef\bbl@nulllanguage{\string\language=0}
2135 \ifx\PackageError\@undefined
2136 \def\bbl@error#1#2{\begingroup\newlinechar='\^^J
2137 \def\{"\^J(babel) \}
2138 \errhelp{#2}\errmessage{\#1}\endgroup}
2139 \def\bbl@warning#1{\begingroup\newlinechar='\^^J
2140 \def\{"\^J\}
2141 \message{\#1}\endgroup}
2142 \def\bbl@info#1{\begingroup\newlinechar='\^^J
2143 \def\{"\}\wlog{#1}\endgroup}
2144 \else
2145 \def\bbl@error#1#2{\begingroup\def\{"\MessageBreak\}
2146 \PackageError{babel}{#1}{#2}\endgroup}
2147 \def\bbl@warning#1{\begingroup\def\{"\MessageBreak\}
2148 \PackageWarning{babel}{#1}\endgroup}
2149 \def\bbl@info#1{\begingroup\def\{"\MessageBreak\}
2150 \PackageInfo{babel}{#1}\endgroup}
2151 \fi
\end{verbatim}
11 Loading hyphenation patterns

The following code is meant to be read by init\TeX{} because it should instruct \TeX{} to read hyphenation patterns. To this end the docstrip option patterns can be used to include this code in the file hyphen.cfg. Code is written with lower level macros.

toks8 stores info to be shown when the program is run.
We want to add a message to the message L\TeX{} 2.09 puts in the \everyjob register. This could be done by the following code:

\begin{verbatim}
% \let\orgeveryjob\everyjob
% \def\everyjob#1{%
% \orgeveryjob{#1}%
% \orgeveryjob{\expandafter{\the\orgeveryjob\immediate\write16{%
% hyphenation patterns for \the\loaded@patterns loaded.}}}%
% \let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}
\end{verbatim}

The code above redefines the control sequence \everyjob in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before L\TeX{} fills the register. There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with SLiT\TeX{} the above scheme won’t work. The reason is that SLiT\TeX{} overwrites the contents of the \everyjob register with its own message.

- Plain \TeX{} does not use the \everyjob register so the message would not be displayed.

To circumvent this a ‘dirty trick’ can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, \dump. Therefore the original \dump is saved in \org@dump and a new definition is supplied.

To make sure that L\TeX{} 2.09 executes the \@begindocumenthook we would want to alter \begin{document}, but as this done too often already, we add the new code at the front of \@preamblecmds. But we can only do that after it has been defined, so we add this piece of code to \dump.
This new definition starts by adding an instruction to write a message on the
terminal and in the transcript file to inform the user of the preloaded hyphenation
patterns.

Then everything is restored to the old situation and the format is dumped.

\providefile{hyphen.cfg}{\date}{\version} Babel hyphens
\xdef\bbl@format{\jobname}
\ifx\AtBeginDocument\@undefined
\def\@empty{}
\let\orig@dump\dump
\def\dump{%
\ifx\@ztryfc\@undefined
\else
\toks0=\expandafter{\@preamblecmds}%
\edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
\def\@begindocumenthook{}%
\fi
\let\dump\orig@dump\let\orig@dump\@undefined\dump}%
\fi
\Define core switching macros
\toks8{Babel «@version@>> and hyphenation patterns for }
\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 #4 {%
\ifx=#1%
\process@synonym{#2}%
\else
\process@language{#1#2}{#3}{#4}%
\fi
\ignorespaces}
\process@synonym
This macro takes care of the lines which start with an =. It needs an empty token
register to begin with. \bbl@languages is also set to empty.
\toks@{}
\def\bbl@languages{}

When no languages have been loaded yet, the name following the = will be a
synonym for hyphenation register 0. So, it is stored in a token register and
executed when the first pattern file has been processed. (The \relax just helps to
the \if below catching synonyms without a language.)
Otherwise the name will be a synonym for the language loaded last.
We also need to copy the hyphenmin parameters for the synonym.

\def\process@synonym#1{%
\ifnum\last@language=\m@ne
\toks@=\expandafter{\@preamblecmds}%
\edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks@}%
\def\@begindocumenthook{}%
\fi
\let\dump\orig@dump\let\orig@dump\@undefined\dump}
\process@synonym
The macro \process@language is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call \addlanguage to allocate a pattern register and to make that register ‘active’. Then the ‘name’ of the language that will be loaded now is added to the token register \toks8 and finally the pattern file is read. For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file language.dat by adding for instance ‘:T1’ to the name of the language. The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. The latter can be used in hyphenation files if you need to set a behaviour depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. \TeX{} does not keep track of these assignments. Therefore we try to detect such assignments and store them in the \langle lang \rangle hyphenmins macro. When no assignments were made we provide a default setting.

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

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.) \bbl@languages saves a snapshot of the loaded languages in the form \bbl@elt\langle language-name\rangle\langle number\rangle\langle patterns-file\rangle\langle exceptions-file\rangle. Note the last 2 arguments are empty in ‘dialects’ defined in language.dat with =. Note also the language name can have encoding info.

Finally, if the counter \language is equal to zero we execute the synonyms stored.
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.

\begin{verbatim}
def\bbl@get@enc#1:#2:#3\@@@{
def\bbl@hyph@enc{#2}
def\bbl@hook@everylanguage#1{}
def\bbl@hook@loadpatterns#1{
\input #1\relax}
def\bbl@hook@loadexceptions\bbl@hook@loadpatterns
\let\bbl@hook@loadkernel\bbl@hook@loadpatterns\begingroup
\def\AddBabelHook#1#2{\expandafter\ifx\csname bbl@hook@#2\endcsname\relax
\def\next{\toks1}\else\def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}\fi
\next}
\ifx\directlua\@undefined
\ifx\XeTeXinputencoding\@undefined\else
\input xebabel.def
\else
\input luababel.def
\fi\else
\input \bbl@format.cfg\relax
\closein1\fi\bbl@hook@loadkernel{switch.def}\readconfigfile
\end{verbatim}

The configuration file can now be opened for reading.

\begin{verbatim}
\openin1 = babel-\bbl@format.cfg\ifeof1
\input babel-\bbl@format.cfg\relax\fi\closein1\endgroup
\bbl@hook@loadkernel{switch.def}
\end{verbatim}

See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.
\message{I couldn’t find the file language.dat, I will try the file hyphen.tex}
\input hyphen.tex
\chardef\l@english\z@

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 $-1$.

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’

If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of \bbl@line. This is needed to be able to recognize the arguments of \process@line later on. The default language should be the very first one.

\if T\ifeof1F\fi T\relax
\ifx\bbl@line\@empty\else
\edef\bbl@line{\bbl@line\space\space\space}%
\expandafter\process@line\bbl@line\relax
\fi
\repeat

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

\begingroup
\def\bbl@elt#1#2#3#4{%
\global\language=#2\relax
\gdef\languagename{#1}%
\def\bbl@elt##1##2##3##4{}%}
\bbl@languages
\endgroup
\fi

and close the configuration file.
\closein1

We add a message about the fact that babel is loaded in the format and with which language patterns to the \everyjob register.
\if\the\toks@\else
\errhelp{language.dat loads no language, only synonyms}
\errmessage{Orphan language synonym}
\fi
\edef\bbl@tempa{%
\everyjob{%
Also remove some macros from memory and raise an error if \toks@ is not empty. 
Finally load switch.def, but the letter is not required and the line inputting it may be commented out.

\let\bbl@line\@undefined
\let\process@line\@undefined
\let\process@synonym\@undefined
\let\process@language\@undefined
\let\bbl@get@enc\@undefined
\let\bbl@hyph@enc\@undefined
\let\bbl@tempa\@undefined
\let\bbl@hook@loadkernel\@undefined
\let\bbl@hook@everylanguage\@undefined
\let\bbl@hook@loadpatterns\@undefined
\let\bbl@hook@loadexceptions\@undefined
⟨/patterns⟩

Here the code for iniTeX ends.

12 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation.
For this language currently no special definitions are needed or available.
The macro \LdfInit takes care of preventing that this file is loaded more than once, checking the category code of the @ sign, etc.

⟨∗nil⟩
\ProvidesLanguage{nil}{⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language} \LdfInit{nil}{datenil}

When this file is read as an option, i.e. by the \usepackage command, nil could be an ‘unknown’ language in which case we have to make it known.
\ifdef\@nohyphenation\@undefined
 \@nopatterns{nil}
 \adddialect\l@nil
\else
 \let\l@nil\@empty
 \let\datenil\@empty
\fi

This macro is used to store the values of the hyphenation parameters \lefthyphenmin and \righthyphenmin.
\providehyphenmins{\CurrentOption}{\m@ne\m@ne}

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

\captionnil
 \datenil \let\captionsnil\@empty
 \let\datenil\@empty
The macro \ldf@finish takes care of looking for a configuration file, setting the main language to be switched on at \begin{document} and resetting the category code of @ to its original value.

\ldf@finish{nil}
\langle/nil\rangle

13 Support for Plain TeX

13.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename hyphen.tex may only be used to designate his version of the American English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based TeX-format. When asked he responded:

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

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

The files bplain.tex and blplain.tex can be used as replacement wrappers around plain.tex and lplain.tex to achieve the desired effect, based on the babel package. If you load each of them with iniTeX, you will get a file called either bplain.fmt or blplain.fmt, which you can use as replacements for plain.fmt and lplain.fmt.

As these files are going to be read as the first thing iniTeX sees, we need to set some category codes just to be able to change the definition of \input

\langle*bplain | blplain\rangle
\catcode'\{=1 % left brace is begin-group character
\catcode'\}=2 % right brace is end-group character
\catcode'\#=6 % hash mark is macro parameter character

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

\openin 0 hyphen.cfg
\ifeof 0
\else
When hyphen.cfg could be opened we make sure that it will be read instead of the file hyphen.tex which should (according to Don Knuth’s ruling) contain the American English hyphenation patterns and nothing else.

We do this by first saving the original meaning of \input (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

\let\a\input

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

\def\input #1 {%
\let\input\a
\a hyphen.cfg
Once that’s done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```latex
\let\a\undefined
\fi
\fi
\fi
\fi
```

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

```latex
\let\a\undefined
\fi
\fi
\fi
\fi
```

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

```latex
\def\fmtname{babel-plain}
\def\fmtname{babel-lplain}
```

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

### 13.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of \LaTeX2e that are needed for `babel`.

We need to define `\loadlocalcfg` for plain users as the \LaTeX definition uses `\InputIfFileExists`. We have to execute `@endofldf` in this case.

```latex
\def\@empty{}
\def\loadlocalcfg#1{\ifvoid#1 \else \closein0 \ifnum\noexpand\input#1.cfg \relax \@endofldf \fi \fi \fi \ifieof0 \closein0 \else \closein0 \fi}
```

### 13.3 General tools

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

```latex
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\def\@nnil{\@nil}
\def\@gobbletwo#1#2{\relax}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@staror@long#1{\@ifstar{\let\l@ngrel@x\relax#1}{\let\l@ngrel@x\long#1}}
\let\l@ngrel@x\relax
```

```latex
\long\def\@firstoftwo#1#2{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\def\@nnil{\@nil}
\def\@gobbletwo#1#2{\relax}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@staror@long#1{\@ifstar{\let\l@ngrel@x\relax#1}{\let\l@ngrel@x\long#1}}
\let\l@ngrel@x\relax
```
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\let\@typeset@protect\relax
\let\protected@edef\edef
\long\def\@gobble#1{}
\edef\@backslashchar{\expandafter\@gobble\string\}\
\def\strip@prefix#1>{}
\def\g@addto@macro#1#2{{%\toks@\expandafter{#1#2}%%\xdef#1{\the\toks@}}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\@expandtwoargs#1#2#3{\edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{#1\ifx#2\@empty\else\expandafter\zap@space\fi #2}

\LaTeX{} has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.
\ifx\@preamblecmds\@undefined\def\@preamblecmds{}\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble
\def\begindocument{\@begindocumenthook\global\let\@begindocumenthook\@undefined\def\do##1{\global\let##1\@undefined}%%\@preamblecmds\global\let\do\noexpand\fi\ifx\@begindocumenthook\@undefined\def\@begindocumenthook{}\fi\@onlypreamble\@begindocumenthook\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

We also have to mimick \LaTeX{}'s \AtBeginDocument; for this to work the user needs to add \begindocument to his file.
\def\begindocument{%\@begindocumenthook\global\let\@begindocumenthook\@undefined\def\do#1{\global\let#1\@undefined}\@preamblecmds\global\let\do\noexpand\\ifx\@begindocumenthook\@undefined\def\@begindocumenthook{}\fi\@onlypreamble\@begindocumenthook\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\def\@endofldf{}
\@onlypreamble\@endofldf
\let\bbl@afterlang\@empty
\chardef\bbl@hymapopt\z@
\LaTeX{} needs to be able to switch off writing to its auxiliary files; plain doesn’t have them by default.

\begin{verbatim}
\ifx\if@filesw\@undefined
   \expandafter\let\csname if@filesw\expandafter\endcsname\csname iffalse\endcsname
\fi

Mimick \LaTeX{}’s commands to define control sequences.
\begin{verbatim}
\def\newcommand\@star@or@long\new@command
\def\new@command#1{\@testopt\@newcommand#10}
\def\@newcommand#1[#2]{\@ifnextchar\[\@xargdef#1[#2]\]{\@argdef#1[#2]}}
\long\def\@argdef#1[#2]#3{\@yargdef#1\@ne{#2}{#3}}
\long\def\@xargdef#1[#2]\[#3\]#4{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter #1\csname\string#1\expandafter\endcsname{#3}}\expandafter\@yargdef\csname\string#1\endcsname\tw@{#2}{#4}}
\long\def\@yargdef#1#2#3{\@tempcnta#3\relax\advance\@tempcnta\@ne\let\@hash@\relax\edef\reserved@a{\ifx#2\tw@ \[\@hash@1\]\fi}\@tempcntb#2\@whilenum\@tempcntb<\@tempcnta\do{\edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}}\let\@hash@##\long\def\@xargdef\@star@or@long\provide@command
\def\provide@command#1{\begingroup\escapechar\m@ne\xdef\@gtempa{\string#1}\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command#1}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
\def\DeclareRobustCommand\@star@or@long\declare@robustcommand
\def\declare@robustcommand#1{\edef\reserved@a{\string#1}\def\reserved@b{#1}\edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}\edef#1{\ifx\reserved@a\reserved@b\noexpand\x@protect\noexpand#1\fi\noexpand\protect\expandafter\csname\string#1\endcsname}}
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\edef\reserved@a{\ifa\tw@ \[\@hash@1\]\fi}\@tempcntb=#2\@whilenum\@tempcntb<\@tempcnta\do{\edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}}\let\@hash@##
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\edef\reserved@a{\reserved@a\[\@hash@1\]}\expandafter\@ifundefined\reserved@a{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a
\def\DeclareRobustCommand\@star@or@long\declare@robustcommand
\def\declare@robustcommand#1{\edef\reserved@a{\string#1}\def\reserved@b{#1}\edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}\edef#1{\ifx\reserved@a\reserved@b\noexpand\x@protect\noexpand#1\fi\noexpand\protect\expandafter\csname\string#1\endcsname}}
\end{verbatim}
\end{verbatim}

109
The following little macro \in@ is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean \in@; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \bbl@tempa.

\def\bbl@tempa{\csname newif\endcsname\ifin@}
\ifx\in@\@undefined
\def\in@#1#2{%\def\in@@##1#1##2##3\in@@{%\ifx\in@##2\in@false\else\in@true\fi}\in@@#2#1\in@\in@@}
\else
\let\bbl@tempa\@empty
\fi
\bbl@tempa

La\TeX{} has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain \TeX{} we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

\def\@ifpackagewith#1#2#3#4{#3}

The La\TeX{} macro \@ifl@aded checks whether a file was loaded. This functionality is not needed for plain \TeX{} but we need the macro to be defined as a no-op.

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

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

\ifx@tempcnta\@undefined\csname newcount@endcsname@tempcnta\relax\fi
\ifx@tempcntb\@undefined\csname newcount@endcsname@tempcntb\relax\fi

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

\ifx\bye\@undefined
13.4 Encoding related macros

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

\def\DeclareTextCommand{% 
\@dec@text@cmd\providecommand 
\}
\def\ProvideTextCommand{% 
\@dec@text@cmd\providecommand 
\}
\def\DeclareTextSymbol#1#2#3{% 
\@dec@text@cmd\chardef#1{#2}#3\relax 
\}
\def\@dec@text@cmd#1#2#3{% 
\expandafter\def\expandafter#2% 
\expandafter{\csname#3-cmd\endcsname 
\expandafter#2% \csname#3\string#2\endcsname 
}% 
\let\@ifdefinable\@rc@ifdefinable 
\expandafter#1\csname#3\string#2\endcsname 
\}}
\textbf{Theorem:} A function $f: \mathbb{R} \to \mathbb{R}$ is called \textit{even} if \[ f(x) = f(-x) \] for all $x \in \mathbb{R}$ and \textit{odd} if \[ f(x) = -f(-x) \] for all $x \in \mathbb{R}$.

\textbf{Definition:} Let $f: \mathbb{R} \to \mathbb{R}$ be a function. We say that $f$ is \textit{continuous} at $a$ if for every $\varepsilon > 0$, there exists a $\delta > 0$ such that for all $x$, if $0 < |x - a| < \delta$, then $|f(x) - f(a)| < \varepsilon$. If $f$ is continuous at every point in its domain, then we say $f$ is \textit{continuous} on its domain.

\textbf{Example:} Consider the function $f(x) = x^2$. This function is continuous on $\mathbb{R}$.

\textbf{Exercise:} Prove that the function $f(x) = x^3$ is odd.

\textbf{Proof:} Let $x \in \mathbb{R}$. Then
\[ f(-x) = (-x)^3 = -x^3 = -f(x) \]

This shows that $f(x) = x^3$ is odd.\]
Currently we only use the \LaTeX{} method for accents for those that are known to be made active in some language definition file.

The following control sequences are used in babel.def but are not defined for plain \TeX{}.

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

13.5 Babel options

The file babel.def expects some definitions made in the \LaTeX{} style file. So we must provide them at least some predefined values as well some tools to set them (even if not all options are available). There in no package options, and therefore and alternative mechanism is provided. For the moment, only \texttt{\babeloptionstrings} and \texttt{\babeloptionmath} are provided, which can be defined before loading babel. \texttt{\BabelModifiers} can be set too (but not sure it works).
14 Tentative font handling

A general solution is far from trivial:

- \addfontfeature only sets it for the current family and it’s not very efficient, and
- \defaultfontfeatures requires to redefine the font (and the options aren’t “orthogonal”).

15 Hooks for XeTeX and LuaTeX

15.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

\LaTeX sets many “codes” just before loading hyphen.cfg. That is not a problem in luatex, but in xetex they must be reset to the proper value. Most of the work is
done in \(\LaTeX\)\text{ini}, so here we just “undo” some of the changes done by \LaTeX. Anyway, for consistency Lua\TeX{} also resets the catcodes.

\begin{verbatim}
2766 ⟨⟨Restore Unicode catcodes before loading patterns⟩⟩ ≡
2767 \AddBabelHook{xetex}{loadkernel}{%
2768 \begingroup
2769 % Reset chars "80-"C0 to category "other", no case mapping:
2770 \catcode`@=11 \count@=128
2771 \loop\ifnum\count@<192
2772 \global\uccode\count@=0 \global\lccode\count@=0
2773 \global\catcode\count@=12 \global\sfcode\count@=1000
2774 \advance\count@ by 1 \repeat
2775 % Other:
2776 \def\0 #1 {%
2777 \global\uccode"#1=0 \global\lccode"#1=0
2778 \global\catcode"#1=12 \global\sfcode"#1=1000 }
2779 % Letter:
2780 \def\L #1 #2 #3 {\global\catcode"#1=11
2781 \global\uccode"#1="#2
2782 \global\lccode"#1="#3
2783 % Uppercase letters have sfcode=999:
2784 \ifnum"#1="#3 \else \global\sfcode"#1=999 \fi }
2785 % Letter without case mappings:
2786 \def\l #1 {\L #1 #1 #1 }
2787 \l 00AA
2788 \L 00B5 039C 00B5
2789 \l 00BA
2790 \l 00D7
2791 \l 00DF
2792 \l 00F7
2793 \L 00FF 0178 00FF
2794 \endgroup
2795 \input #1\relax}
2796 ⟨⟨/Restore Unicode catcodes before loading patterns⟩⟩

Now, the code.
\end{verbatim}

\begin{verbatim}
2797 ⟨⟨/xetex⟩⟩
2798 \def\BabelStringsDefault{unicode}
2799 \let\xebbl@stop\relax
2800 \AddBabelHook{xetex}{encodedcommands}{%
2801 \def\bbi{\tempa[#1]}
2802 \if\bbi@empty
2803 \def\XeTeXinputencoding"bytes"%
2804 \else
2805 \def\XeTeXinputencoding"#1"%
2806 \fi
2807 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
2808 \AddBabelHook{xetex}{stopcommands}{%
2809 \xebbl@stop
2810 \let\xebbl@stop\relax}
2811 ⟨⟨/Restore Unicode catcodes before loading patterns⟩⟩
2812 ⟨⟨/Font selection⟩⟩
2813 ⟨⟨/xetex⟩⟩
\end{verbatim}
15.2 LuaTeX

This part relies on the lua stripts in luatex-hyphen by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard. Élie also improved the code below.

```latex
\begin{verbatim}
\directlua{
require("luatex-hyphen")
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.callback)
    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
function Babel.addpatterns(pp, lg)
    local lg = lang.new(lg)
    local pats = lang.patterns(lg) or ''
    lang.clear_patterns(lg)
    for p in pp:gmatch('[^%s]+') do
        ss = ''
        for i in string.utfcharacters(p:gsub('%d', '')) do
            ss = ss .. '%d?' .. i
        end
        ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
        ss = ss:gsub('%.%%d%?$', '%%.')
        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
        if n == 0 then
            tex.sprint(\[
                \textbf{New pattern: } p
            \])
        else
            tex.sprint(\[
                \textbf{Renew pattern: } p
            \])
        end
    end
end
\end{verbatim}
```
lang.patterns(lg, pats)
end
}
\endgroup
\def\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook{luatex}{encodedcommands}{%
  \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
  \ifx\bbl@tempa\bbl@tempb\else
    \directlua{Babel.begin_process_input()}
  \fi}
\AddBabelHook{luatex}{stopcommands}{\luabbl@stop
  \let\luabbl@stop\relax}
\AddBabelHook{luatex}{patterns}{%
  \ifcsname lu@texhyphen@loaded@	he\language\endcsname \else
    \global@namedef{lu@texhyphen@loaded@	he\language}{}%
    \directlua{
      luatexhyphen.loadlanguage([[\string#1]],
        '\the\language')}%}
\AddBabelHook{luatex}{adddialect}{%
  \ifdefined{bbl@patterns@}{}{%
    \begingroup
      \@expandtwoargs\in@{\number\language,}{,\bbl@pttnlist}%
      \ifin@else
        \ifx\bbl@patterns@\@empty\else
          \directlua{ Babel.addpatterns{
            [[\bbl@patterns@]], \number\language }%
        \fi
        \@ifundefined{bbl@patterns@#1}{}{
          \directlua{ Babel.addpatterns(
            [[\space\csname bbl@patterns@#1\endcsname]],
              \number\language )}%
        \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
      \fi
    \endgroup}}
\AddBabelHook{luatex}{everylanguage}{%
  \directlua{
    processnow = (tex.language == 0) or
    (luatexhyphen.lookupname([[\string#1]]) == nil)}%
  \ifnum0\directlua{tex.sprint(processnow and "0" or "1")}\relax
    \global@namedef{lu@texhyphen@loaded@	he\language}{}%
  \fi}
\AddBabelHook{luatex}{loadpatterns}{%
  \ifnum0\directlua{tex.sprint(processnow and "0" or "1")}\relax
    \input #1\relax
\belpatterns  This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones and \bbl@patterns<lang> for language ones. We make sure there is a space between words when multiple commands are used.

16 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.
17 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