Enjoying babel

Enrico Gregorio

1 Introduction

Back in the 1980s, when TEX was making its way in the world, it was an all-American piece of software. LATEX was based on Plain TEX and was even more American in style.

For instance, Knuth chose to set the DVI reference point one inch to the right and one inch from the top of the sheet of paper; maybe this is one of the design errors in the TEX family of programs. However, with a judicious setting of `\hoffset` and `\voffset`, users could correctly print TEX output on A4 paper. He did provide tools for typesetting European languages (with all their strange accents) but it was not possible to hyphenate two languages simultaneously.

Overall, however, the situation was not so nice for us Europeans. As of today, the European Union comprises 27 countries and has 22 official languages (in three different alphabets), not counting Luxemburgish and various languages spoken by minorities: in the UK, besides English, there are Scottish Gaelic, Scots, Scottish English, Welsh, Irish, Cornish and Manx; in Spain, besides *Castellano*, there are *Catalá* (Catalan, in three different varieties), *Galego* (Galician) and *Euskara* (Basque) plus some others. There are many countries where two or more languages have official status, possibly only in some regions: this is the case of Italy, where German and French are official languages in two provinces and Slovenian is "almost official" in one province.

Version 3 of TEX was hailed with enthusiasm, as it provided the possibility of hyphenating in 256 languages simultaneously and its 8 bit design allowed for extended sets of characters which made it possible to get rid of explicit accents, with all the related and well known hyphenation problems: in fact TEX does not hyphenate a word containing an explicit accent (past the accent), which is a big nuisance for languages such as French and German and is intolerable for Slavic languages such as Czech and Polish. By the way: do you know the difference between *slovenčina* and *slovenščina*?[1]

Earlier than the introduction of TEX 3, Johannes Braams developed the babel system that permitted substituting the fixed tags in LATEX like 'Chapter' and 'Table of Contents' with localized tags for some European languages. It also provided a method, based on the package german by Bernd Raichle, for inputting accented characters while allowing for good hyphenation.

Of course, before TEX 3, users were limited to hyphenating one language at a time, and special versions of the LATEX (or Plain or $\mathcal{AMS}$-TEX) format had to be prepared. But, at least, one could typeset a book in Italian where chapters were named 'Capitolo' and the table of contents 'Indice'.

LATEX 2_ε improved the situation. It supported package options and the support for babel was integrated by declaring control sequences that contain the fixed tags: for example, `\chaptername` expands to 'Chapter' by default, but babel can easily change its meaning in every language it supports.

The supported languages are many, 44 in the current version, and not only European. It may be surprising to learn that at least as many European languages are *not* supported. Among the main ones, Maltese, Lithuanian and Latvian still lack support (they are all official in the EU); one of the four official languages of Switzerland, Romansh, is missing. But Latin, Esperanto and Interlingua are present.

I should mention Thomas Esser and his teTEX distribution, which made it easy to enable hyphenation rules and format creation for LATEX. The same idea was used by MiKTEX through a menu. TEX Live, also based on the teTEX scripts, offers this facility as well. Moreover, today's fast computers and large memories make it possible to enable all available rules and then forget about the matter.

I'll talk later briefly about Plain TEX or $\mathcal{AMS}$-TEX users, who are not left in the cold, after all. But the bulk of the paper is devoted to babel and LATEX.

2 Calling babel

The babel package is called as usual:

`\usepackage[`⟨*languages*⟩`]{babel}`

where ⟨*languages*⟩ is a comma separated list of languages, whose names can be found in Table 1. You should name all the languages you plan to use in the document, for example

```
\usepackage[italian,english]{babel}
```

if the document has English as its main language, but some parts of it are written in Italian.

I said that the supported languages are 44, but the table has more items. Some names are just synonyms (Hungarian and Magyar, for example) and some denote *dialects*, that is, languages which share hyphenation patterns with others (for example, Austrian is a dialect of German, Acadian and Canadien

[1] *Slovenčina*, or *slovenský jazyk* is the official language of Slovakia. *Slovenščina*, or *slovenski jezik*, is the official language of Slovenia. They are two different countries of the EU and do not share a border.

Table 1: List of babel languages

acadian	bulgarian	frenchb	lowersorbian	russian
afrikaans	canadian	galician	magyar	samin
albanian	canadien	german	malay	scottish
american	catalan	germanb	meyalu	serbian
australian	croatian	greek	naustrian	slovak
austrian	czech	hebrew	newzealand	slovene
bahasa	danish	hungarian	ngerman	spanish
bahasai	dutch	icelandic	norsk	swedish
bahasam	english	indon	nynorsk	turkish
basque	esperanto	indonesian	polish	ukrainian
brazil	estonian	interlingua	polutonikogreek	uppersorbian
brazilian	finnish	irish	portuges	welsh
breton	francais	italian	portuguese	UKenglish
british	french	latin	romanian	USenglish

are dialects of French).[2] What is a dialect? While it is basically the same language as another, they might differ in minor aspects regarding typesetting rules or fixed tags.

In Portuguese typography, the month name in a date is capitalized, while Brazilians use lowercase. In Austria people speak German, but the name of the first month of the year is *Januar* in Germany and *Jänner* in Austria. These two languages can be called also with the `ngerman` or `naustrian` options, which select the "New Orthography" (*Neue Rechtschreibung*) hyphenation.

Other names are there just for backward compatibility: this is the case of `french` and `frenchb`. It is sufficient to look at the beginning of `babel.sty` to realize what each option does. Let's look at the first lines:

```
\DeclareOption{acadian}{\input{frenchb.ldf}}
\DeclareOption{albanian}{\input{albanian.ldf}}
\DeclareOption{afrikaans}{\input{dutch.ldf}}
```

Every language option loads a *language definition file* with extension `.ldf` (in the following, LDF). We see from these lines that `acadian` loads `frenchb.ldf`, and indeed Acadian is for babel a dialect of French. Similarly, Afrikaans is a dialect of Dutch. Conversely, Albanian is a language by itself, and it had better be, since it does not belong to any of the big European language families; the same is true for Basque.

The name of the LDF for French and its dialects is `frenchb` for historical reasons, which apply also to `germanb`: since there are packages around named french and german, the final 'b' was to remind users that they were using babel in the old days when packages were specified as options to `\documentstyle`.

[2] Of course, Canadien is not a dialect of Canadian.

Note that every option loads the corresponding LDF and it is this file's duty to handle double loadings. We'll see later some examples.

The most important thing to remember about language options is that the *last* language loaded is considered the main language of the document. In case there is only one it can be specified as a global option (i.e., as an option to `\documentclass`); other packages, such as varioref, understanding that option can therefore benefit from it. Notice, though, that varioref does not understand all babel's aliases. If there is more than one language, it can happen that a package does not correctly understand the global options: the solution is to specify them as local for each package.[3]

Don't specify a language as a global option and other languages as options to babel. This is a sure cause for head scratching, trying to figure out what went wrong. Try, for example

```
\documentclass[italian]{article}
\usepackage[greek,italian]{babel}
\begin{document}
XYZ
\end{document}
```

Do you see what happens? The option `italian` is not the last option seen by babel, because global options are scanned first.

3 Tags

In Table 2 is the list of fixed tags with their definitions in English. Not all of these tags are used in the standard classes article, report and book. For example, `\proofname` is used by amsthm as the name used in the `\begin{proof}` environment. In Table 3 you

[3] By the way, while writing this paper I discovered two bugs in varioref, version 1.4p: `\extrasbrazil` and `\extrasportuges` were misspelled as `\extrabrazil` and `\extraportuges`.

Table 2: List of tags in English

`\prefacename`	Preface
`\refname`	References
`\abstractname`	Abstract
`\bibname`	Bibliography
`\chaptername`	Chapter
`\appendixname`	Appendix
`\contentsname`	Contents
`\listfigurename`	List of Figures
`\listtablename`	List of Tables
`\indexname`	Index
`\figurename`	Figure
`\tablename`	Table
`\partname`	Part
`\enclname`	encl
`\ccname`	cc
`\headtoname`	To
`\pagename`	Page
`\seename`	see
`\alsoname`	see also
`\proofname`	Proof
`\glossaryname`	Glossary

Table 3: List of tags in Ukrainian

`\prefacename`	Вступ
`\refname`	Література
`\abstractname`	Анотація
`\bibname`	Бібліоґрафія
`\chaptername`	Розділ
`\appendixname`	Додаток
`\contentsname`	Зміст
`\listfigurename`	Перелік ілюстрацій
`\listtablename`	Перелік таблиць
`\indexname`	Покажчик
`\authorname`	Іменний покажчик
`\figurename`	Рис.
`\tablename`	Табл.
`\partname`	Частина
`\enclname`	вкладка
`\ccname`	копія
`\headtoname`	До
`\pagename`	с.
`\seename`	див.
`\alsoname`	див. також
`\proofname`	Доведення
`\glossaryname`	Словник термінів

find the same tags with their contents in Ukrainian; as you can see, different language traditions require also different tags.

What about changing or improving them? Suppose a document is written part in English and part in Italian. We would like to define a command to refer to sections in an abstract way, with text of the form

```
As we saw in \secref{sec:a} ...
...
Abbiamo visto nella \secref{sec:b} ...
```

in such a way that the command expands to 'Section 2' in English and to 'Sezione 2' in Italian. The definition is straightforward:

```
\newcommand{\secref}[1]{\secname~\ref{#1}}
```

But how to include `\secname` in the babel tags? It's a matter of saying, in the preamble of the document,

```
\newcommand{\secname}{}
\addto\captionsenglish{%
  \renewcommand{\secname}{Section}}
\addto\captionsitalian{%
  \renewcommand{\secname}{Sezione}}
```

We first introduce to LaTeX the command `\secname`; it is babel's job to provide the correct definition when the user chooses the English or the Italian language: babel orders LaTeX to execute `\captions`⟨*lang*⟩ whenever the ⟨*lang*⟩ is selected and the tags need to be changed. The `\addto` trick simply appends the second argument (a token list) to the replacement text of the control sequence given as first argument.

In the same vein, if we need to change a tag, say we want 'Elenco delle illustrazioni' instead of the default for `\listfigurename`, we can say

```
\addto\captionsitalian{%
  \renewcommand{\listfigurename}{%
    Elenco delle illustrazioni}}
```

It is better if these definitions to complement `\captions`⟨*lang*⟩ are given using only 7-bit input, so that they do not depend on the overall encoding of the document. In this way you will be able to simply copy those definitions from one document to another without worrying about the encoding; this is even more important if a personal style file is made.

The package has another facility: for each requested ⟨*lang*⟩, the macro `\extras`⟨*lang*⟩ is defined. It contains commands to be executed every time the ⟨*lang*⟩ is selected. A stupid example could be to typeset every part in Italian in bright red:

```
\addto\extrasitalian{\color{red}}
```

There is a companion macro `\noextras`⟨*lang*⟩ that contains things to be undone when passing from a language to another and this change is not protected by a group or environment. For example, correct hyphenation in Italian requires that the straight quote be considered for hyphenation, i.e., it must have a nonzero `\lccode`. Otherwise, phrases such as `dell'amicizia` would not be hyphenated fully as `del-l'a-mi-ci-zia` but only as `del-l'amicizia`. Therefore `italian.ldf` contains the instructions

```
\addto\extrasitalian{\lccode`\'=`\'}
\addto\noextrasitalian{\lccode`\'=0 }
```

because the \lccode of the straight quote must be reset to zero for other languages. If we were foolish enough to choose to typeset Italian in red, we should undo the choice when returning to other languages, so that we should say

```
\newcommand{\defaultcolor}{\color{black}}
\addto\noextrasitalian{\defaultcolor}
```

At \begin{document}, LaTeX will execute both \extras⟨*lang*⟩ and \captions⟨*lang*⟩, for the default ⟨*lang*⟩, so the modifications stated in the preamble will be active from the beginning.

Other facilities include the setting of dates. For every language there is a macro \date⟨*lang*⟩. When a different language is selected, LaTeX executes this command, which should redefine \today. So, say we want to use abbreviated month names in Italian: we issue in the preamble

```
\renewcommand{\dateitalian}{%
 \renewcommand{\today}{%
  \number\day~\ifcase\month\or gen.\or
   feb.\or ...\or dic.\fi\ \number\year}}
```

(the definition is incomplete to save space). The names of the months are not tags, because the date format can be very different between languages.

4 Language selection

Assume we have made our choice of the languages for the document. How to change from one to another? There are many ways, each solving a particular problem. The main language of the document is selected implicitly, because LaTeX issues a

```
\selectlanguage{⟨main-lang⟩}
```

command, where ⟨*main-lang*⟩ is the last chosen language option, as seen before.

Such a command can be issued everywhere; it changes everything to the new language: tags, typographical choices, shorthands and, of course, hyphenation rules. Therefore, after

```
\selectlanguage{portuges}
```

every following chapter will be tagged as '*Capítulo*'; after

```
\selectlanguage{french}
```

the typographical rules for French will be active. For example,

```
\selectlanguage{french}
Il dit: \og Qu'est-ce que tu veux?\fg
```

will be typeset as

Il dit : « Qu'est-ce que tu veux ? »

The correct spaces before the colon and the question mark will be automatically inserted, as required by the French tradition.

The language selection can act on various aspects regarding typesetting:

1. tags and dates,
2. typesetting conventions,
3. input conventions,
4. hyphenation.

The command \selectlanguage acts on all four aspects. The same holds for its environment form \begin{otherlanguage}. Input such as

```
\begin{otherlanguage}{turkish}
...
\end{otherlanguage}
```

is equivalent to \selectlanguage{turkish}, but confines the changes to the duration of the environment, in the usual way. The *-form environment \begin{otherlanguage*} acts only on typesetting and input conventions and hyphenation. It has a command form, for setting a small piece of text:

```
\foreignlanguage{⟨lang⟩}{⟨text⟩}
```

is largely equivalent to

```
\begin{otherlanguage*}{⟨lang⟩}
⟨text⟩
\end{otherlanguage*}
```

but the environment form allows many paragraphs.

The last environment is \begin{hyphenrules}; it acts only on the hyphenation rules. Usually, among the loaded hyphenation rules there is a set with no rule at all, commonly called nohyphenation. So, if we have text in an unsupported language, we can use this empty set of rules.

5 Other commands

The macro \languagename expands to the name of the current language. The command \iflanguage takes as arguments

1. a language name,
2. a token list to be executed if the current language is the same as the first argument,
3. a token list to be executed otherwise.

6 Input conventions

Before LaTeX supported 8-bit input via the inputenc package, people had a hard time with all the encodings used by different operating systems. Only 7-bit-clean input was guaranteed to be interpreted in the same way on all platforms. With TeX 2 it was even impossible to directly input characters in the upper half of an 8-bit code page.

During this time, the package german introduced a convention for inputting accented characters by preceding them with a double quote:

```
sch"oner G"otterfunken Stra"se
ba"cken Schi"ffart
```

could be used instead of the more awkward

```
sch\"oner G\"otterfunken Stra{\ss}e
ba{\ck}en Schi{\ff}ahrt
```

after having defined

```
\def\ck{\discretionary{k-}{k}{ck}}
\def\ff{ff\discretionary{-}{f}{}}
```

and similar commands for other sequences.

Braams developed this scheme further, making it easy to define similar shorthands for all languages. Nowadays, with the development of encodings such as UTF-8, these conventions are less important. However, UTF-8 is not yet widespread and is intrinsically foreign to standard TEX, so occasionally they can still be useful.

Suppose I have to use a Latin 1 keyboard, but need to type text in Czech: most of the diacritics used by Czech are not directly accessible with Latin 1. Fortunately, it is fairly easy to set up suitable "double quote" conventions.

The only letters that can take different diacritics are the 'e' (háček and acute accent) and the 'u' (ring and acute accent). Since Latin 1 keyboards have vowels with the acute accent, except for 'y', we don't need anything special for the other five.

Let's analyze the Czech alphabet. It uses four kinds of diacritics: the *háček* (as in 'č'), the acute accent (as in 'ý'), the ring (as in 'ů') and the apostrophe. The háček is produced with `\v`; the Czech support by babel provides `\q` for the apostrophe and `\w` for the the ring.

Let's decide to use the double quote for inputting most diacritics. In a `.sty` file to be loaded after babel we can write the following code.

```
\initiate@active@char{"}
\addto\extrasczech{%
  \languageshorthands{czech}}
\addto\extrasczech{\bbl@activate{"}}
\addto\noextrasczech{\bbl@deactivate{"}}
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\dq{"}}\x
```

Here `\initiate@active@char`, `\bbl@activate` and `\bbl@deactivate` are standard babel functions; we add `\languageshorthands` to `\extrasczech` in order to declare the use of the defined shorthands. The last three lines are a trick to define `\dq` as a double quote with category code 12.

Then we can write (incomplete for brevity):

Table 4: Improved input for Czech or „Česko"

A	A	"E	Ě	"L	Ľ	"R	Ř	X	X
Á	Á	F	F	M	M	S	S	Y	Y
B	B	G	G	N	N	"S	Š	"Y	Ý
C	C	H	H	"N	Ň	T	T	Z	Z
"C	Č	I	I	O	O	"T	Ť	"Z	Ž
D	D	Í	Í	Ő	Ó	U	U		
"D	Ď	J	J	P	P	Ú	Ú		
E	E	K	K	Q	Q	"U	Ů	"‘	„
É	É	L	L	R	R	V	V	"’	“
a	a	"e	ě	"l	ľ	"r	ř	x	x
á	á	f	f	m	m	s	s	y	y
b	b	g	g	n	n	"s	š	"y	ý
c	c	h	h	"n	ň	t	t	z	z
"c	č	i	i	o	o	"t	ť	"z	ž
d	d	í	í	ó	ó	u	u		
"d	ď	j	j	p	p	ú	ú		
e	e	k	k	q	q	"u	ů	"<	«
é	é	l	l	r	r	v	v	">	»

```
\declare@shorthand{czech}{"c}
  {\textormath{\v{c}}{\ddot c}}
\declare@shorthand{czech}{"C}
  {\textormath{\v{C}}{\ddot C}}
\declare@shorthand{czech}{"d}
  {\textormath{\q{d}}{\ddot d}}
\declare@shorthand{czech}{"D}
  {\textormath{\q{D}}{\ddot D}}
...
\declare@shorthand{czech}{"y}
  {\textormath{\'{y}}{\ddot y}}
\declare@shorthand{czech}{"Y}
  {\textormath{\'{Y}}{\ddot Y}}
\declare@shorthand{czech}{"z}
  {\textormath{\v{z}}{\ddot z}}
\declare@shorthand{czech}{"Z}
  {\textormath{\v{Z}}{\ddot Z}}
\declare@shorthand{czech}{"‘}
  {\textormath{\quotedblbase}
              {\mbox{\quotedblbase}}}
\declare@shorthand{czech}{"’}
  {\textormath{\textquotedblleft}
              {\mbox{\textquotedblleft}}}
\declare@shorthand{czech}{"<}{\flqq}
\declare@shorthand{czech}{">}{\frqq}
```

The `\declare@shorthand` command used here takes three arguments:

1. a language name,
2. a one or two character sequence, the first of which must have been declared as we did before,
3. what we want to substitute when TEX sees that character sequence.

Table 5: Warning message for missing hyphenation patterns

```
Package babel Warning: No hyphenation patterns were loaded for
(babel)                the language 'Albanian'
(babel)                I will use the patterns loaded for \language=0 instead.
```

The macro `\textormath` typesets its first argument in text mode, the second one in math mode. It is customary to make a double quote combination equivalent to the math accent `\ddot` in order to avoid strange error messages.

We take the occasion to also introduce abbreviations for the inverted double quotes in German style, common in Czech, and also the guillemets (`\flqq` and `\frqq` are babel jargon for them). The non-obvious choices are `"y` and `"u` for the 'y with acute accent' and 'u with ring' (and the corresponding uppercase letters). In Table 4 we find a list of characters along with their input.

Users can define new shortcuts on the fly using similar commands. The internal commands are more efficient and can be restricted to one language:

- `\useshorthands{`⟨*chars*⟩`}`, for introducing new shorthand characters.
- `\defineshorthand`, which behaves like the internal command `\declare@shorthand`, but it doesn't take a language as an argument, only the shorthand and its definition.
- `\aliasshorthand{`⟨*char*$_1$⟩`}{`⟨*char*$_2$⟩`}` for making ⟨*char*$_2$⟩ a shorthand equivalent to ⟨*char*$_1$⟩.

Continuing our example, we could add

```
\aliasshorthand{"}{|}
```

to `\extrasczech` and then input `"C` as `|C`. Of course ⟨*char*$_1$⟩ must have already been defined as a shorthand.

Sometimes it is necessary to disable a shorthand character, because of bad interactions with other packages, notably XY-pic. This package does its job by parsing the source looking for special characters. An activated character is likely to disturb this parsing, so users can say

```
\shorthandoff{⟨chars⟩}
\shorthandon{⟨chars⟩}
```

where ⟨*chars*⟩ is the list of characters to disable or enable. For example:

```
\shorthandoff{"^}
```

7 Attributes

Some languages can have *attributes*, which modify their behavior. Currently only `greek` and `latin` use this facility. So "Polytoniko Greek" can be chosen either with

```
\usepackage[greek]{babel}
\languageattribute{polutoniko}
```

or simply by specifying `polutonikogreek` as the language.

Latin uses the `withprosodicmarks` attribute, which makes `^` and `=` shorthands to typeset accents in Latin poetry and specifying the vowel quantities in order to emphasize the meter. It has also the attribute `medievallatin` for making the lowercase 'u' equivalent to the uppercase 'V' and using traditional ligatures.

8 Problems

The choice of hyphenation rules is done at format creation, based on the file `language.dat`. This file (excluding comments) has the following appearance:

```
english         hyphen.tex
=usenglish
=USenglish
=american
usenglishmax    ushyphmax.tex
dumylang        dumyhyph.tex
nohyphenation   zerohyph.tex
basque          xu-bahyph.tex
bulgarian       xu-bghyphen.tex
...
```

Its format is due to Sebastian Rahtz. Basically it lists on each line a language name along with the hyphenation patterns file; a line can consist also of an equal sign followed by a language name, meaning that this name is an alias for the preceding two-item line. At this level, language names are actually arbitrary strings. It is babel's job to associate each of its supported languages with one of these strings.

A problem comes immediately to our attention: Albanian is supported by babel, but no hyphenation patterns for it are available. A user saying

```
\usepackage[albanian]{babel}
```

will be saluted with a message from TEX which you find in Table 5.

A warning of the same type would appear for any language whose hyphenation patterns were not enabled at format creation time by the system administrator. Some distributions like MiKTEX are pretty conservative in this regard and enable only a few languages; many questions in the discussion forums are about this.

In my opinion this is a design error: in fact Albanian (or the other non-enabled language) would be hyphenated using US English rules which are completely different from those of Albanian. I believe that no hyphenation is better than *wrong* hyphenation. Splitting an Italian word like *cestino* (small basket) as *ces-tino* is a bad grammatical error.[4]

In Albanian the digraph 'rr' is considered a single letter, and must never be divided. And this is only one of the problems which can arise by allowing hyphenation with English rules.

All language definition files begin with something like

```
\ifx\l@italian\@undefined
    \@nopatterns{Italian}%
    \adddialect\l@italian0\fi
```

This could be changed into

```
\ifx\l@italian\@undefined
    \@nopatterns{Italian}%
  \ifx\l@nohyphenation\@undefined
    \adddialect\l@italian\@cclv
  \else
    \adddialect\l@italian\l@nohyphenation
  \fi
\fi
```

Thus we would associate a non-enabled language either to the one with no hyphenation patterns by definition or to language number 255, which is very likely undefined. The `\@nopatterns` error message can be changed by saying that TEX won't use any hyphenation.

Users who can't enable a language, either because they are not the system administrator or there is no hyphenation patterns file, can correct this behavior themselves by finding in the log file the warning similar to that of Table 5; immediately after it they'll find a line such as

```
\l@albanian = a dialect from \language0
```

The first control sequence is the key to the solution. It is now sufficient to write, just after loading babel,

```
\makeatletter
\ifx\l@nohyphenation\@undefined
\chardef\l@albanian=255
\else
\let\l@albanian=\l@nohyphenation
\fi
\makeatother
```

Since commands like `\selectlanguage{albanian}` execute the command

```
\language\l@albanian
```

the trick is done.

[4] The kind of error which made our teachers in primary school shriek in horror.

It is important to note that shorthand characters in a language remain active also in the languages where they are not used in that way; in those cases they expand to the character they denote. This is why they can have unwanted side effects with other packages.

The reason to keep them active is clear: a language changing command can appear in risky places, for example in the `.aux` file, as Braams points out in the babel documentation. The example he makes is the following: a user could use a shorthand in the optional argument of a `\bibitem` command.

However even this doesn't work. Suppose someone writes a document with German as the main language and parts in English. Assume that in the bibliography, written in German, we find

```
\bibitem["UB99]{ub99}
A. "User und E. Benutzer, ``Ein Titel''.
```

A reference like `\cite{ub99}` will come out correctly in a German context, but not in an English context: they will print, respectively, [ÜB99] and ["UB99] (or ["UB99], if OT1 encoding is used). The reference will consistently be resolved correctly only if the author writes

```
\bibitem[\german{"UB99}]{ub99}
```

(where I've used `\german` simply as a shortcut for `\foreignlanguage{german}`).

9 Double loading

The LDF for Hungarian starts as follows:

```
\@namedef{captions\CurrentOption}{%
  \def\prefacename{El\H osz\'o}%
  \def\refname{Hivatkoz\'asok}%
...
\@namedef{date\CurrentOption}{%
  \def\today{%
    \number\year.\nobreakspace
    \ifcase\month\or
...
```

after the check for the existence of the hyphenation patterns in the format. What does this mean?

This LDF is called if Hungarian is requested with either the option `hungarian` or `magyar`. In the first case, the fundamental macros will be defined as

```
\captionshungarian
\datehungarian
\extrashungarian
\noextrashungarian
```

and with the suffix `magyar` in the second case. Quite recently, Péter Szabó has exploited this possibility with a new implementation of the Hungarian LDF (`http://ctan.org/tex-archive/language/hungarian/babel`).

10 Plain TEX users

In principle, it should be possible to use babel with Plain TEX or formats built upon it like $\mathcal{AMS}$-TEX. However, this is pretty much undocumented, apart from the instructions to build a format by running iniTEX on `bplain.tex` and `\dump`. The user interface is not specified.

Recently I wrote a very primitive Plain TEX package supporting multiple languages. It's a substantially simplified version of babel's machinery and I will use it to try and illustrate how it works.

First we run iniTEX on a file called `hyplain.tex`:

```
\catcode`\{=1
\catcode`\}=2
\catcode`\@=11
\let\orig@input\input
\def\input hyphen {%
  \let\input\orig@input \input hyrules }
\orig@input plain
```

D. E. Knuth has decreed that `plain.tex` cannot be modified except for preloaded fonts. But we can always use some TEX trick; since the file *is* immutable, it will contain the line '`\input hyphen`'; at that point we restore the original meaning of `\input` and input `hyrules.tex` instead of `hyphen.tex`.

The file `hyrules.tex` defines the interface commands. The most important is `\selectlanguage` which, unlike that of babel, requires two arguments: a two letter ISO language code and a two letter country code. For example,

```
\selectlanguage{en}{US}
\selectlanguage{it}{IT}
\selectlanguage{de}{AT}
```

would switch, respectively, to American English, to the Italian of Italy[5] and to the German of Austria.

Users can also define personal language selection commands: define command `\italiano` with

```
\addalias{\italiano}{it}{IT}
```

to make it equivalent to `\selectlanguage{it}{IT}`. Internally, this command calls `\it_IT` and similarly for other combinations. If a language combination is not defined in the user modifiable file `hylang.tex`, a fallback language `\zz_ZZ`, without hyphenation patterns, is selected and a warning message is issued.

Two token lists are associated to each language, similar to the `\extras`⟨*lang*⟩ and `\noextras`⟨*lang*⟩ of babel and an 'undo' token list is maintained. Each time a language is selected, the following happens:

1. what is in the 'undo' token list is executed and the list is cleared (locally),
2. the parameter `\language` is given the appropriate value,
3. the *extras* for the chosen language are executed,
4. the *noextras* are put in the 'undo' token list.

Since assignments to the 'undo' token list are local, this list will always be loaded with the correct commands.

The 'extras' list for a language ought to set the left and right hyphenation minima; this setting has no counterpart in the 'noextras' list. Other things, instead, must be set in both places: for example, the `\lccode` setting of the apostrophe for Italian.

Users can modify `hylang.tex`, adding or deleting languages. It is recommended not to change the first one, so that we are sure that `\language` zero refers always to American English, as in Plain TEX.

The commands are

```
\definelanguage{xx}{YY}{xxhyph}
\refinelanguage{xx}{YY}
  {<something>}{<something>}

\definedialect{yy}{YY}{xx}{XX}
\refinedialect{yy}{YY}
  {<something>}{<something>}
```

where `xx` is the two letter code of a language (I suggest '`nde`' for 'New Orthography German') and `XX` is the two letter country code. Actually these codes could be arbitrary strings, but I believe that we (and babel) would benefit from standardization.

For example, one could write

```
\definelanguage{fr}{FR}{frhyph}
\definedialect{fr}{CA}{fr}{FR}
```

to set up for Canadian French. One could use the nonexistent country code 'ZZ' for an unspecified country: this would be the case for Esperanto.

The macro `\refinelanguage` refers to the language by its codes; then in the third argument one puts the 'extras' and in the fourth the 'noextras'. This command can be given as many times as one desires, since new lists are appended to the existing ones. The macro `\refinedialect` is the same as `\refinelanguage`. Dialects do not inherit extras: the interface is primitive, just the way Plain TEX devotees are used to.

For example, my settings for Italian are these:

```
\definelanguage{it}{IT}{ithyph}
\refinelanguage{it}{IT}
  {\lccode`\'=`\'}{\lccode`\'=0 }
```

11 Advanced babel programming

babel works in a similar way to HyPlain. It has of course many more features: for example, functions to save the meaning of commands or the value of variables. In the LDF for Italian we can see

[5] Italian is an official language also in Switzerland, where different typography conventions could be used.

```
\addto\extrasitalian{%
  \babel@savevariable\clubpenalty
  \babel@savevariable\widowpenalty
  \babel@savevariable\finalhyphendemerits
  \clubpenalty3000 \widowpenalty3000
  \finalhyphendemerits50000000 }%
```

When the language changes from Italian to another one, the values of the listed parameters[6] are restored and possibly changed again by the new language: babel uses the 'noextras' token list and its internal mechanism to restore a clean setting and then it applies the 'extras' for the new language. Let's see how it's done:

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

Let the variable name be `\foo`. The macro appends to the replacement text of `\originalTeX` (which corresponds to the 'undo' list in HyPlain) the tokens `\foo=`. This is done inside a group in order to be sure not to clobber the value of `\toks@`. The `\edef` is done when the value of `\toks@` is what has just been set; after that token list, the present value of `\foo` is put. Then `\x` is executed, which closes the group and redefines `\originalTeX`.

A very similar trick is performed when we say `\babel@save\baz`, where `\baz` is a command. First `\baz` is made equivalent to a command of the form `\babel@1234` (where `1234` stands for the actual value of a counter reserved by babel). The same thing happens as before, so `\originalTeX`'s replacement text will end with

```
\let\baz=\babel@1234
```

(there is no problem in interpreting that strange token, because it has already entered the scanning mechanism). Finally, the counter is stepped, providing a fresh number for the next `\babel@save`.

We can apply this method to modify the behavior of a command without forcing users to change their input. A silly example is the following:

```
\makeatletter
\addto\extrasitalian{%
  \babel@save\emph\let\emph\textbf}
\makeatother
```

In this way, typing `\emph{ciao}` in an Italian context would print the word in bold face, while keeping the abstract nature of the command. This could be obtained also by

```
\let\origemph\emph
\renewcommand{\emph}{%
  \iflanguage{italian}%
    {\textbf}{\origemph}}
```

but the method with `\babel@save` is of course more robust and does not require a long chain of nested `\iflanguage` calls if we need different effects for several languages.

I've said before that `\declare@shorthand` takes as the first argument a language name, but this is not strictly true. There is the concept of 'shorthand group'. In the present version of babel there are three levels: (1) user, (2) language, and (3) system. The package checks in that order when it is resolving a shorthand.

Let's make an example: German uses the double quote as a shorthand character, for instance `"A` to get 'Ä'. It is not necessary to define every combination `"`⟨*char*⟩, because there is already a definition of the active double quote at the system level (it expands to a double quote, of course).

The default system level shorthands are `"`, `'`, `` ` ``, and `~`. When an LDF introduces a new shorthand character, it ought to define its behavior at the system level. For example the LDF for Esperanto says

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

because it uses `^` for shorthands. The same is true of `frenchb.ldf`, where there is

```
\declare@shorthand{system}{:}{\string:}
```

along with similar lines for `!`, `?` and `;`. If a user says `\defineshorthand{"A}{\hat{A}}`, this shorthand would take precedence over a possible definition of `"A` by the LDF. If the LDF defines a `"` shorthand, this takes precedence over the system one.

With the development of input encoding support, especially Unicode, for TEX these devices are less useful, because it is possible to input directly any character. On the other hand, other babel features remain invaluable.

The most recent versions of pdfTEX allow a different treatment for the typographic conventions of French, for example, making it possible to reduce the number of active characters. Some support for this is available through the microtype package.

◇ Enrico Gregorio
Dipartimento di Informatica, Settore di Matematica
Università di Verona, Italy
Enrico dot Gregorio (at) univr dot it
http://profs.sci.univr.it/~gregorio

[6] The setting of `\clubpenalty` is wrong, it should refer to `\@clubpenalty`. A bug report has been mailed.