Babel, a multilingual style-option system for use with LaTeX's standard document styles*

Johannes Braams

Abstract

The standard distribution of LaTeX contains a number of document styles that are meant to be used, but also serve as examples for other users to create their own document styles. These styles have become very popular among LaTeX users. But it should be kept in mind that they were designed for American tastes and contain a number of hard-wired texts. This article describes a set of document-style options that can be used in combination with the standard styles, which makes the latter adaptable to other languages.

1 Introduction

Although Leslie Lamport has stated [5] that one should not try and write *one* document-style option to be used with *all* the standard document styles of LaTeX, that is exactly what I have done with this system of style options. The reasons for this approach will be explained in section 2.

A lot of the ideas incorporated in this set of files come from the work of Hubert Partl [4], `german.tex`. Some parts in the implementation are different, others are the same. It will be shown that `german.tex` can be modified to fit into this scheme of style options.

2 Why babel?

When I first started using LaTeX I was very happy with just the style files that are distributed with the standard distributions of TeX and LaTeX. That means, as long as I made texts in English I was happy. Then as other users found out about LaTeX and its advantages, they started using it for texts in languages other than English. As I was the most experienced LaTeX user at the time, they came to me and asked me 'When I'm writing a report in Dutch I don't want chapters to be named "Chapter", I want them to be named "Hoofdstuk", how do you change that?'. At that time I didn't know, but I soon found out. The first thing I found was that Leslie Lamport states [2, pages 85–86] that you have to redefine the command `\@chapapp` to get the desired result. This looked rather promising to me, so I had a look at the style files to find out how other such strings as "Figure" might be redefined. It was then that I found out that `\@chapapp` is the *only* string defined this way, whereas all others are hard-wired into the style.

My first solution to this problem was to create a new document style file called `artikel.sty` as a "Dutch" counterpart to `article.sty`. The same was done for `report.sty`. This is exactly what Leslie Lamport suggests [5]. This approach has one major drawback however: you get two copies of basically the same file to maintain. This was discovered when newer releases of the styles reached our site. The standard styles had to be replaced *and* edited all over again to get the "Dutch" versions back. About the same time, in early 1988, a discussion on this subject appeared in TeXhax. One of the persons commenting was Hubert Partl. The method he suggested was to modify the standard document styles by replacing the hard-wired texts by macros such as `\@chapapp`. This led me to my second attempt at a solution. I modified the standard styles (all four of them) as suggested, but while doing that added an option, implemented like the option `draft`, by defining a command `\ds@dutch`.

* During the development ideas from Nico Poppelier, Piet van Oostrum and many others have been used.

This command would set a variable to indicate which language was requested. This variable I used later on in a `\case` statement. In this `\case` statement a choice is made between English, Dutch and possibly other languages for texts such as “Figure” and “Contents”. Unfortunately, some of this implied changing the secondary style files `xxx10.sty`, `xxx11.sty` and `xxx12.sty`. This was unfortunate because one of the research groups in our laboratories complained their document style didn't work properly. It turned out that their style was a modified `article.sty` that had been given a different name, but it still loaded `art10.sty` etc. I found a temporary solution, but I still wasn't exactly happy with the situation. Besides this, the drawback of replacing the document styles with newer versions still existed.

When after a while a new version of the LaTeX distribution arrived at our site, I began to think about a different way to solve the problem. In the meantime Hubert Partl had his `german.sty` published in *TUGboat* [4]. His article pointed the way to a different solution. Triggered by the discussion in TeXhax in early 1989 about how to detect which is the main (primary) style when processing a document, I started work on what is now available as `dutch.sty` version 1.0, dated may 1989[1]. While working on this style option I discovered that some parts could be borrowed from `german.sty`. This ‘discovery’ and some discussions I had with others at EuroTeX89, the fourth European TeX Conference, held in september 1989 in Karlsruhe, led me towards a more universal approach. The basic idea behind it was, starting from the algorithm to detect the main style, to design an approach with one common file that contained macro definitions needed by a number of language-specific style options. Users specify the name of any of these language-specific options as an option to the `\documentstyle` command, and internally the common file is read.

3 LaTeX and document-style files

Before the I discuss some of the code in the babel system I would like to discuss the document-style mechanism used by LaTeX. Every LaTeX document should start with a line like:

```
\documentstyle[opt1,opt2,...]{docstyle}
```

This line of code instructs LaTeX to first load the file `docstyle.sty`. When that is done the ‘options’ are processed *in the order specified*, by reading the files `opt1.sty`[2], `opt2.sty`, etc. This implies that definitions, made in the file `docstyle.sty` can be overridden in one of the option files. It is even possible to redefine code from the very kernel of LaTeX, but you have to know what you are doing.

Some care has to be taken in writing document-style options, because a number of problems can occur. First of all, if a document-style option should be modest in size; if it tries to redefine most of the code in `docstyle.sty` I think you should write (and maintain) your own, complete, document style. Next, as it was possible to override definitions from the main file in an option file, it is of course also possible to override definitions made in another option file. When this happens, your document might depend on the order in which you have specified your document-style options.

This mechanism of overriding definitions from the main document style is exploited in the babel system. The macros that contain the hard-wired texts are redefined in the common part of babel, replacing each of these texts with a unique macro. These macros have to be defined in the language-specific files.

[1] This file is available from `listserv@hearn.bitnet` as file `dutch.old`.

[2] Except when the documentstyle defines the control sequence `\ds@`⟨*opt1*⟩; in that case this control sequence will be executed.

4 LaTeX and multilingual documents

In a european environment it sometimes happens that one wants to write a document that contains more than *one* language. I have an example of a document, published by the EEC, that contains 9 (nine) different languages. Also in linguistics one can find documents written in more than one language, i.e. to compare two languages.

If you have to write such a multilingual document you should try to conform to the typographical conventions in use for each language. A well known example is the type of quotation marks used. TeX supplies the user with “quoted text”, but a Dutch user might want to have „quoted text”, whereas a German text should contain „quoted text“ and a frenchman would perhaps like to see something like «quoted text». These language specific conventions should be implemented in a document-style option file for each language. These files should then be useable with *all* document styles.

In such a multilingual document a user would specify the languages used as options to the `\documentstyle` command. He would also want a mechanism to be able to switch between these languages in a simple way. When he would use TeX version 3.0 for the processing of his document, he would also want the hyphenation to come out right for the different languages.

5 Overview of the babel solution

5.1 The core of the system

The problems described in sections 3 and 4 can be solved using the babel system of document-style options.

The core of this system currently performs three functions.

1. It defines a user interface for switching between languages;
2. It contains code to dynamically load several sets of hyphenation patterns;
3. It ‘repairs’ the document styles provided in the standard distribution of LaTeX.

Obviously part 2 can only be used while running iniTeX to create a new format, whereas part 3 should *not* be read by iniTeX. Part 3 should even disappear when LaTeX version 3.0 arrives, as the style files supplied with the new LaTeX will no longer be language specific. Part 1 can either be loaded into the format with multiple hyphenation patterns, or it can be read while processing a document.

For this reason the core of the babel system is stored in two separate files, `babel.switch`, containing parts 1 and 2, and `babel.sty` which contains part 3. The file `babel.sty` will instruct LaTeX to load `babel.switch` if necessary, the file `babel.switch` checks the format to see if hyphenation patterns *can* be loaded.

5.2 Language specifics

The language switching mechanism contains a couple of hooks for the developers of language-specific document-style options.

First of all the macro `\originalTeX` should be defined. Its function is to disable special definitions made for a language to bring TeX into a ‘defined’ state. A language-specific document-style option might, for example, introduce an extra active character. It would then also modify the definitions of `\dospecials` and `\@sanitize`. Such an option would then define a macro to restore the original definitions of these macros and restore the extra active character to its normal category code. It would then `\let \originalTeX` to this ‘restoration’ macro.

To enable the language-specific definitions three macros are provided in the switching mechanism, `\captions`⟨*language*⟩, `\date`⟨*language*⟩ and `\extras`⟨*language*⟩.

The macro `\captions`⟨*language*⟩ should provide definitions for the macros that replaced the hard-wired texts in the document style and the macro `\date`⟨*language*⟩ should provide a definition for `\today`. The real fun starts with the macro `\extras`⟨*language*⟩. This macro should activate all definitions needed for ⟨*language*⟩.

6 The user interface

The user interface to the babel system is quite simple. He should specify the languages he wants to use in his document in the list of document-style options. For instance, for a document in which both the English and the Dutch language are used, the first line could read:

```
\documentstyle[a4,dutch,english]{artikel1}
```

Please note that in this case the Dutch-specific definitions are inactive when LaTeX has finished processing document-style option files.

If the user then wants to switch from English to Dutch he would include the command

```
\selectlanguage{dutch}
```

before starting to write Dutch.

If a user wants to write a document-style option of his own he might like to define a macro that checks which language is in use at the time the macro is executed. For this purpose the macro `\iflanguage{`⟨*language*⟩`}{`⟨*then-clause*⟩`}{`⟨*else-clause*⟩`}` is available.

7 Implementation of the core of the system

In this section I would like to discuss some parts of the implementation of the babel system. Not all code will be shown, because some parts of it are just series of slightly modified code from the standard document styles. The files are fully documented and interested readers can print them if they have access to the `doc` option, described by Frank Mittelbach.

The description of the macros that follows is based on an environment using TeX 3.x, together with a version of `lplain.tex` based on `plain.tex` version 3.x. The actual implementation allows for other situations as well, i.e a version of `babel.sty` for TeX 2.x will be available.

7.1 Switching languages

For each language to be used in a document a control sequence of the form `\l@`⟨*language*⟩ has to be defined. This will either be done while loading hyphenation patterns or while loading the language-specific file. The implementation of `\selectlanguage{`⟨*language*⟩`}` and `\iflanguage{`⟨*language*⟩`}{`⟨*then case*⟩`}{`⟨*else case*⟩`}` is based on the existence of `\l@`⟨*language*⟩.

To switch from one language to another the macro `\selectlanguage` is available. Its definition can be seen in figure 1. The first action it takes is to check whether the ⟨*language*⟩ is known, if it is not an error is signalled. If the language is known `\originalTeX` is called upon to reset any previously set language-specific definitions. Next the register `\language` is updated and the three macros that should activate all language-specific definitions are executed. Finally the macro `\originalTeX` receives a new replacement text in order to be able to deactivate the definitions just activated.

The macro `\iflanguage` (see figure 2) will issue a warning when its argument is an 'unkown' language. It then goes on to compare the value of `\language` and `\l@`⟨*language*⟩ and executes either its secon or third argument.

```
\def\selectlanguage#1{%
  \@ifundefined{l@#1}
       {\@nolanerr{#1}}
       {\originalTeX
        \language=\expandafter\csname l@#1\endcsname\relax
        \expandafter\csname captions#1\endcsname
        \expandafter\csname date#1\endcsname
        \expandafter\csname extras#1\endcsname
        \gdef\originalTeX{\expandafter\csname noextras#1\endcsname}
       }
}
```

Figure 1: The definition of `\selectlanguage`.

```
\def\iflanguage#1#2#3{%
  \@ifundefined{l@#1}
    {\@nolanerr{#1}}
    {\ifnum\language=\expandafter\csname l@#1\endcsname\relax
       #2 \else #3
     \fi}
}
```

Figure 2: The definition of `\iflanguage`

7.2 Dynamically loading patterns

With the advent of TEX 3.0 it has become possible to build a format with more than one hyphenation pattern preloaded. The core of the babel system provides code, to be executed by iniTEX *only*, to dynamically load hyphenation patterns. The only restriction is that the implementation of TEX that you use has to have rather high settings of `trie_size` and `trie_op_size` to actually load several hyphenation patterns.

For the purpose of dynamically loading hyphenation patterns a 'configuration file' has to be introduced. This file will be read by iniTEX. Each line should contain either a comment, nothing or the name of a language and the name of the file that contains the hyphenation patterns for that language. In figure 3 an example of such a file, instructing iniTEX to load patterns for three languages, English, Dutch and German.

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

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

Figure 3: An example configuration file

The configurationfile will be read line by line using TEX's `\read` primitive. Because the name of a file might be followed by a space-token and comment (as in the example) a macro to process each line is needed. The definition of this macro, `\process@language`, can be found in figure 4. As can be seen in the definition of this macro, its second argument *always* has to be followed by a space-token. The effect of this is that any trailing spaces are removed. The macro strips all spaces following its arguments. It's first argument is used to define `\l@`⟨*language*⟩. The macro `\addlanguage` is basically a non-outer version of the plain TEX macro `\newlanguage`.

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

Figure 4: The definition of `\process@language`.

The second argument of `\process@language` is the name of the file containing the hyphenation patterns. Before the file can be read, the register `\language` has to updated.

The configuration file is read in a `\loop` (see figure 5). When a record is read from the input file a check is done whether the record was empty. If it was not, a space token is added to the end of the string of tokens read. The reason for this is that we have to be sure there always is at least *one* space token present. When that has been taken care of the data just read can be processed. The last thing to do is to check the status of the input file, in order to decide whether TEX has to continue processing the `\loop`. When all patterns have been processed the value of `\language` is restored.

```
\loop
  \read1 to \@config@line
  \ifx\@config@line\empty
  \else
    \edef\@config@line{\@config@line\space}
    \expandafter\process@language\@config@line
  \fi
  \ifeof1 \@morefalse \fi
  \if@more\repeat
\language=0
```

Figure 5: Reading the configuration file line by line

7.3 'Repairing' LATEX's standard document styles

A large part of the core of the babel system is dedicated to 'repair' the standard document styles. This means redefining the macros in table 1.

macro	article	report	book	letter
`\fnum@figure`	×	×	×	×
`\fnum@table`	×	×	×	×
`\tableofcontents`	×	×	×	
`\listoffigures`	×	×	×	
`\listoftables`	×	×	×	
`\thebibliography`	×	×	×	
`\theindex`	×	×	×	
`\abstract`	×	×	×	
`\part`	×	×	×	
`\chapter`		×	×	
`\appendix`		×	×	
`\cc`				×
`\encl`				×
`\ps@headings`				×

Table 1: macros that need to be redefined for the four standard document styles.

As an example of the way the macros have to be redefined, the redefinition of `\tableofcontents` is shown in figure 6.

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

Figure 6: An example of redefining a command

The standard styles can be distinguished by checking the existence of the macros `\chapter` (not in `article` and `letter`) and `\opening` (only in `letter`). The result of these checks is stored in the macro `\doc@style`. When `\doc@style` already exists (which is the case when for instance `artikel1.sty` is used [7]) it is not superseded (see figure 7).

```
\@ifundefined{doc@style}
    {\def\doc@style{0}
        \@ifundefined{opening}
            {\@ifundefined{chapter}
                {\def\doc@style{1}}
                {\def\doc@style{2}}
            }{\def\doc@style{3}}
    }{\relax}
```

Figure 7: Determining the main document style

8 Implementing a language specific document-style option file

To illustrate the way a language specific file can be implemented the file `dutch.sty` is discussed here. Note that not all of the code containde in the file `dutch.sty` is shown here, only those parts that are of interest for the scope of this article are included. If the reader would like to see the complete code, he can print all files in the babel system, using the file `doc.sty`, described by Frank Mittelbach in [6].

8.1 Compatibilty with plain TEX

The file `german.tex` [4] was written in such a way that it can be used by both plain TEX users and LATEX users. This seemed a good idea, so all files in the babel system can be processed by both plain TEX and LATEX. But some of the "useful hacks" from LATEX are used, so for a plain TEX user they have to be defined. For this purpose the format is checked at the start of a language specific file. If the format is `plain` an extra file, called `latexhax.sty` is read.

This file should be read only once, so another check is done on the existence of one of the commands defined there.

A new group is started to keep the definition of the macro `\format`, which is used in the following if statement, local. When the current format turns out to be plain TEX the file `latexhax.sty` has to be read. But the definitions in that file should remain valid after the group is closed. This could be accomplished by making all definitions

```
{\def\format{plain}
\ifx\fmtname\format
    \expandafter\ifx\csname @ifundefined\endcsname\relax
      \gdef\next{latexhax.sty}
      \aftergroup\input\aftergroup\next
    \fi
\fi}
```

Figure 8: Conditonal loading of `latexhax.sty`

`global`, but another solution is to tell TEX to process the file `latexhax.sty` *after* the current group has been closed. The command `\aftergroup` puts the next token on a list to be processed after the group.

8.2 Switching to the Dutch language

In section 7.1 the names of macros needed to switch to a language have been described. In figure 9 these macros and their definition are shown for the Dutch language.

```
\def\captionsdutch{\gdef\refname{Referenties}%
                   \gdef\abstractname{Samenvatting}%
                   \gdef\bibname{Bibliografie}%
                   ...
                   \gdef\pagename{Pagina}}

\def\datedutch{%
    \gdef\today{\number\day~\ifcase\month\or
                januari\or februari\or maart\or april\or
                mei\or juni\or juli\or augustus\or
                september\or oktober\or november\or december\fi
                \space \number\year}}
\begingroup \catcode`\"\active

\gdef\extrasdutch{%
    \global\let\dospecials\dutch@dospecials
    \global\let\@sanitize\dutch@sanitize
    \catcode`\"\active
    \gdef"{\protect\dutch@active@dq}
    \gdef\"{\protect\@umlaut}
}\endgroup

\def\noextrasdutch{%
    \catcode`\"12
    \global\let\dospecials\original@dospecials
    \global\let\@sanitize\original@sanitize
    \global\let\"\dieresis
}
```

Figure 9: The macros needed to switch to the Dutch language

The definitions of `\captionsdutch` and `\datedutch` are pretty straightforward and need not be discussed. The macro `\extrasdutch` will be discussed in some more detail.

First, because for Dutch (as well as for German) the `"` character is made active, the LaTeX macros `\dospecials` and `\@sanitize` have to be redefined to include this character as well. The new definitions are implemented as two special commands, so we globally `\let` the originals to their new versions. Then the `"` character is made active and is defined. Then, to prevent an error when `\"` appears in a moving argument, the macro `\"` is redefined and made robust. All this is done inside a group to keep the category code change for the `"` character local.

The macro `\extrasdutch` has a counterpart, `\noextrasdutch`, that cancels the extra definitions made by `\extrasdutch`. It changes the `\catcode` of the `"` character back to 'other' and globally `\let`s the macros `\dospecials` and `\@sanitize` to their original definitions. The original definition of `\"` is restored as well.

In figure 10 the code needed to redefine `\dospecials` and `\@makeother` is shown.

```
\begingroup
  \def\do{\noexpand\do\noexpand}%
  \xdef\dutch@dospecials{\dospecials\do\"}%
  \expandafter\ifx\csname @sanitize\endcsname\relax
% do nothing if \@sanitize is undefined...
  \else
     \def\@makeother{\noexpand\@makeother\noexpand}%
     \xdef\dutch@sanitize{\@sanitize\@makeother\"}%
  \fi
\endgroup

\global\let\original@dospecials\dospecials
\global\let\original@sanitize\@sanitize
```

Figure 10: Code needed for the redefinition of `\dospecials` and `\@makeother`.

8.3 An extra active character

All the code disccussed sofar is necessary because we need an extra active character. This character is then used as indicated in table 2. One of the reasons for this is that in the Dutch language a word with an umlaut can be hyphenated just before the letter with the umlaut, but the umlaut has to disappear if the word is broken between the previous letter and the accented letter.

`"a`	`\"a` which hyphenates as `-a`; also implemented for the other letters.
`"|`	disable ligature at this position.
`"-`	an explicit hyphen sign, allowing hyphenation in the rest of the word.
``"` ``	lowered double left quotes (see example below).
`"'`	normal double right quotes.
`\-`	like the old `\-`, but allowing hyphenation in the rest of the word.

Table 2: The extra definitions made by `dutch.sty`

In [3] the quoting conventions for the Dutch language are discussed. The preferred convention is the single-quote Anglo-American convention, i.e. 'This is a quote'. An alternative is the slightly old-fashioned Dutch method with initial double quotes lowered to the baseline, „This is a quote”, which should be typed as ``"`This is a quote"'``.

8.3.1 Supporting macro definitions

The definition of the active " character needs a couple of support macros. The macro `\allowhyphens` is used make hyphenation of word possible where it otherwise would be inhibited by TEX. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`.

```
\gdef\allowhyphens{\penalty\@M \hskip\z@skip}
```

Then a macro is defined to lower the Dutch left double quote to the same level as the comma. It prepares a low double opening quote in box register 0. This macro was copied form `german.tex`.

```
\gdef\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@}
```

The macro `\set@low@box` is used to define low opening quotes. Since it may be used in arguments to other macros it needs to be protected.

```
\gdef\dlqq{\protect\@dlqq}
\gdef\@dlqq{{%
  \ifhmode
    \edef\@SF{\spacefactor\the\spacefactor}
  \else
    \let\@SF\empty
  \fi
  \leavevmode\set@low@box{''}
  \box\z@\kern-.04em\allowhyphens\@SF\relax}}
```

For reasons of symmetry we also define `"'`. This command is defined similar to `\dlqq`, except that the quotes aren't lowered to the baseline.

```
\gdef\@drqq{{%
  \ifhmode
    \edef\@SF{\spacefactor\the\spacefactor}
  \else
    \let\@SF\empty
  \fi
  ''\@SF\relax}}
```

The original double quote character is saved in the macro `\dq` to keep it available.

```
\begingroup \catcode`\"12
  \gdef\dq{"}
\endgroup
```

The original definition of `\"` is stored as `\dieresis`. The resason for this is that if a font with a different encoding scheme is used the definition of `\"` might not be the plain TEX one.

```
\global\let\dieresis\"
```

In the Dutch language vowels with a dieresis or umlaut accent are treated specially. If a hyphenation occurs before a vowel-plus-umlaut, the umlaut should disappear. To be able to do this, the hyphenation break behaviour for the five vowels, both lowercase and uppercase, could be defined first in terms of `\discretionary`. But this results in a large `\if`-construct in the definition of the active ".

As both Knuth and Lamport have pointed out, a user should not use " when he really means something like `''`. For this reason no distinction is made between vowels and consonants. Therefore one macro, `\@umlaut`, specifies the hyphenation break behaviour for all letters.

```
\def\@umlaut#1{%
    \allowhyphens%
    \discretionary{-}{#1}{\dieresis #1}%
    \allowhyphens}
```

The last support macro to be defined is `\dutch@active@dq`.

```
\gdef\dutch@active@dq#1{%
     \if\string#1`\dlqq{}%
\else\if\string#1'\drqq{}%
\else\if\string#1-\allowhyphens-\allowhyphens%
\else\if\string#1|\discretionary{-}{}{\kern.03em}%
\else\if\string#1i\allowhyphens\discretionary{-}{i}{\dieresis\i}%
                  \allowhyphens%
\else\if\string#1j\allowhyphens\discretionary{-}{j}{\dieresis\j}%
                  \allowhyphens%
\else \@umlaut{#1}\fi\fi\fi\fi\fi\fi}
```

The macro reads the next token and performs some appropriate action. If no special action is defined, it will produce an umlaut accent on top of argument 1.

The last definition needed is a replacement for `\-`. The new version of `\-` should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TeX in this respect is very unfortunate for languages such as Dutch and German, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TeX can generate from the hyphenation patterns.

```
\def\-{\allowhyphens\discretionary{-}{}{}\allowhyphens}
```

8.4 Activating the definitions

The last action that should be performed by a language specific file, is activating it's definitions. Before doing that the macro `\originalTeX` should be definined.

```
\@ifundefined{originalTeX}{\let\originalTeX\relax}{}
```

Also, the macro `\l@`⟨*language*⟩ should be defined. If it hasn't already been defined, this means that no hyphenation patterns were loaded for this language.

```
\@ifundefined{l@dutch}{\addlanguage{dutch}}{}
\selectlanguage{dutch}
```

9 Conclusion

In this article a system of document-style option files has been presented that supports the multilingual use of LaTeX. Some of the code involved has been discussed. The actual files will me made available through the international networks. They will be stored in the fileserver in the Netherlands (address: `LISTSERV@HEARN.BITNET`), the file `babel readme` will explain what you need to get to be able to use the system. The system was developed using the `doc` option, so the files available are fully documented.

References

[1] Donald E. Knuth, *The TeXbook*, Addison-Wesley, 1986.

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

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

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

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

[6] Frank Mittelbach, *The doc-option*, *TUGboat* 10 (1989) #2, p. 245–273.

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

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