Creating a mailing

Johannes Braams

september 7, 1994

1 Introduction

This package is intended to be used when you want to send a large number of letters, all with (almost) the same text.

This package is based on the former style option `merge` by Graeme McKinstry, but is is a reimplementation with a different user interface.

2 The user interface

`\addressfile` The commands `\addressfile` takes a filename as its argument. When the file can't be found by LaTeX, the user will be asked to supply a different name.

The address file should have the following format:

```
Name of addressee
Street\\town
Opening text of the letter
(optional definitions)
<blank line>
Name of addressee
Street\\town
Opening text of the letter
(optional definitions)
<blank line>
...
```

The various addresses are separated by a blank line in the file (multiple blank lines in between addresses are allowed). It is also possible to have multiple lines with definitions; they will all be executed.

`\mailingtext` The text of the mailing is entered as the argument of `\mailingtext`. A difference with the original `merge.sty` is that this package allows control sequences in the argument of `\mailingtext`. These control sequences should then be defined in the file with the address information.

`\makemailing` When `\makemailing` is called the letters are produced, combining the information found in the address file with the text of the mailing.

3 The implementation

3.1 User interface

`\addressfile` The argument to `\adressfile` is the name of the file with the address information.

```
\newcommand{\addressfile}[1]{%
  \def\M@filename{#1}}
```

`\mailingtext` The argument to this macro contains the entire text of the mailing. The text may contain control sequences to be able to make variations in the text.

```
\long\def\mailingtext#1{\global\mailing@text={#1}}
```

`\makemailing` The command `\makemailing` will produce the mailing, reading addresses, openings and optional definitions of variable text parts from an external file.

```
\def\makemailing{%
  \M@openadrfile
  \loop
    \read@info
    \if@notready
      \begin{letter}{\M@toname\\\M@toaddress}%
        \opening{\M@opening}%
        \vskip\baselineskip
        \the\mailing@text
      \end{letter}
    \fi
    \if@notready
  \repeat}
```

3.2 Allocations

`\M@adrfile` We need to allocate an input stream for the file with the address information.

```
\newread\M@adrfile
```

`\mailing@text` The contents of the letter are stored in a token register

```
\newtoks\mailing@text
```

`\if@notready` A switch which indicates if the file `\M@adrfile` has been exhausted.

```
\newif\if@notready
\newif\if@notemptyoreof
```

3.3 Internal macros

`\M@openadrfile` The macro `\M@openadrfile` tries to open `\M@filename`. It that doesn't succeed it asks the user to supply a new name. This is done untill a file is found.

```
\def\M@openadrfile{%
  \openin\M@adrfile\M@filename\relax
  \ifeof\M@adrfile
    \loop
      %\PackageWarning{mailing}{I can't find the file \M@filename}
      \typeout{I can't find the file \M@filename!}
      \closein\M@adrfile
      \typein[\M@filename]{Enter a new name}
      \openin\M@adrfile\M@filename
      \ifeof\M@adrfile
    \repeat
  \fi}
```

`\read@info` The macro `\read@info` takes care of the reading of all the information from `\M@adrfile`, needed to format another letter.

```
\def\read@info{%
  \skip@empty@lines
```

The macro `\skip@empty@lines` leaves the non-empty line it found in `\M@lines`. If it found an end of file condition the `\if@notready` flag will be set to `\iffalse`.

```
  \if@notready
    \let\M@toname\M@line
    \read\M@adrfile to\M@toaddress
    \read\M@adrfile to\M@opening
    \test@eof
    \if@notready\read@defs\fi
  \fi
  }
```

`\read@defs` Reads definitions of control sequences from the file `\M@adrfile` until either an empty line is found or the end of file is reached. Each line is stored in a control sequence and it is executed after all definitions are read.

```
\def\read@defs{%
  \def\M@defns{}%
  {\loop
    \endlinechar=-1
    \read\M@adrfile to\M@line
    \endlinechar=`\^^M
```

We need to get the expansion of `\M@line` into the definition of `\M@defns`, not `\M@line` itself. Therefore `\M@line` is expanded before `\M@defns`.

```
    \expandafter\toks@\expandafter\expandafter
      \expandafter{\expandafter\M@defns\M@line}%
    \xdef\M@defns{\the\toks@}%
    \test@emptyoreof
    \if@notemptyoreof
    \repeat}%
  \M@defns
  }
```

`\test@eof` The macro `\test@eof` tests the status of of the input file.

```
\def\test@eof{%
  \ifeof\M@adrfile
    \@notreadyfalse
  \else
    \@notreadytrue
  \fi}
```

`\test@emptyoreof` The macro `\test@emptyoreof` checks whether we reached an empty line *or* the end of the file.

```
\def\test@emptyoreof{%
  \@notemptyoreoftrue
  \ifx\M@line\@empty
    \global\@notemptyoreoffalse
  \fi
  \ifeof\M@adrfile
    \global\@notemptyoreoffalse
    \global\@notreadyfalse
  \fi}
```

`\skip@empty@lines` This macro skips empty lines until it finds either a non-empty line or the end of the file. If necessary it sets the `\if@notready` flag. The last line read is left in `\M@line`.

```
\def\skip@empty@lines{%
  {\loop
    \endlinechar=-1
    \ifeof\M@adrfile
      \global\@notreadyfalse
      \@tempswafalse
    \else
      \global\@notreadytrue
      \global\read\M@adrfile to\M@line
      \ifx\M@line\@empty
        \@tempswatrue
      \else
        \@tempswafalse
      \fi
    \fi
    \if@tempswa
    \repeat}%
  }
```