

Creating a mailing*

Johannes Braams

2024/07/20

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 a reimplementaion 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.

*This file has version number v1.0c, last revised 2024/07/20.

3 The implementation

3.1 User interface

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

```
1 \newcommand{\addressfile}[1]{%
2   \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.

```
3 \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.

```
4 \def\makemailing{%
5   \M@openadrfile
6   \loop
7     \read@info
8     \if@notready
9       \begin{letter}{\M@toname\\M@toaddress}%
10      \opening{\M@opening}%
11      \vskip\baselineskip
12      \the\mailing@text
13      \end{letter}
14      \fi
15      \if@notready
16      \repeat}
```

3.2 Allocations

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

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

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

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

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

```
19 \newif\if@notready
20 \newif\if@notemptyoreof
```

3.3 Internal macros

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

```
21 \def\M@openadrfile{%
22   \openin\M@adrfile\M@filename\relax
23   \ifeof\M@adrfile
24     \loop
25       %\PackageWarning{mailing}{I can't find the file \M@filename}
26       \typeout{I can't find the file \M@filename!}
27       \closein\M@adrfile
28       \typein[\M@filename]{Enter a new name}
```

```

29     \openin\M@adrfile\M@filename
30     \ifeof\M@adrfile
31     \repeat
32     \fi}

```

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

```

33 \def\read@info{%
34     \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`.

```

35     \if@notready
36         \let\M@toname\M@line
37         \read\M@adrfile to\M@toaddress
38         \read\M@adrfile to\M@opening
39         \test@eof
40         \if@notready\read@defs\fi
41     \fi
42 }

```

`\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.

```

43 \def\read@defs{%
44     \def\M@defs{}%
45     {\loop
46         \endlinechar=-1
47         \read\M@adrfile to\M@line
48         \endlinechar='\^M

```

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

```

49     \expandafter\toks@\expandafter\expandafter
50     \expandafter{\expandafter\M@defs\M@line}%
51     \xdef\M@defs{the\toks@}%
52     \test@emptyoreof
53     \if@notemptyoreof
54     \repeat}%
55     \M@defs
56 }

```

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

```

57 \def\test@eof{%
58     \ifeof\M@adrfile
59     \@notreadyfalse
60     \else
61     \@notreadytrue
62     \fi}

```

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

```

63 \def\test@emptyoreof{%
64     \@notemptyoreoftrue

```

```

65 \ifx\M@line\@empty
66   \global\@notemptyoreoffalse
67 \fi
68 \ifeof\M@adrfile
69   \global\@notemptyoreoffalse
70   \global\@notreadyfalse
71 \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`.

```

72 \def\skip@empty@lines{%
73   {\loop
74     \endlinechar=-1
75     \ifeof\M@adrfile
76       \global\@notreadyfalse
77       \@tempswafalse
78     \else
79       \global\@notreadytrue
80       \global\read\M@adrfile to\M@line
81       \ifx\M@line\@empty
82         \@tempwattrue
83       \else
84         \@tempswafalse
85       \fi
86     \fi
87     \if@tempswa
88     \repeat}%
89 }

```