FORMLETT: for Letters to Multiple Receivers

Zhuhan JIANG
Department of Mathematics Statistics and Computing Science, University of New England, Armidale N.S.W. 2351, Australia. Email: zhuhan@neumann.une.edu.au

(Completed on 12 November 1993, current form on 22 January 1995)

In this article, the author explains how to use a form-letter style **formlett.sty**, designed for plain TeX and LaTeX or LaTeX2e. formlett.sty supports different parameter input methods, parameter naming and defaulting mechanism, as well as facilities for previewing parameter positions and printing labels. It is written for the purpose of being powerful, robust and above all easy to use.

Introduction

In this modern world of ours, we often need to send a set of form letters, personalised or non-personalised, to many receivers at the same time. Or perhaps we would like to have a collection of business letters or information brochures which we would like to call upon *instantly* whenever we need them.

Our purpose here is therefore to describe a comprehensive implementation of such a macro system, handling form letters under TeX or LaTeX. The main objective is to provide an easy way to output many form letters with their own parameters, with or without the use of multiple files. There will be a coherent and simple format for putting parameters inside a form letter, with a number of helping facilities for such as naming parameters and previewing their positions. A minimum support for printing mailing labels is also provided.

The concept of macros [1] for form letters is not new: there already exist macros in this connection such as **merge**, **textmerg** and **address** to name a few, see [2,3] for further details. Our stress here is therefore laid on the ease to use, along with the power and the robustness of the macros.

Format

In this section, we always assume that the file formlett.sty or formlett.tex of version 2.2 has already been input. Certain version of formlett.sty can be obtained via anonymous ftp from CTAN sites at macros /generic.

Essentially, each form letter, or letter format or letter template, will be included between \beginletter and \endletter. Letter parameters that will be specified later on for each particular letter can be given by \paras[<m>][<n>], representing the <m>-th parameter of the <n>-the parameter group, at the positions you want them to be. Let a cluster denote a complete collection of parameters that may be specified for a form letter. Then we can specify letter parameters group by group for a cluster such that inside each group the parameters are given *sequentially* and a termination of one group should not affect the resynchronisation for the next group. For instance we can keep the address of a receiver as a single group, as it often contains different number of (parameter) lines.

We often want to treat a particular group of letter parameters somewhat uniformly. This can be achieved by one of the following commands

```
\blockparas[<m>][<n>][<pre>][<post>]
\addressparas[<m>][<n>][<width>][<indent>]
```

They represent parameters in the <n>-th group, from the <m>-th to the last parameter of that group. For \blockparas, tokens <pre> and <post> are those to be added in front and behind respectively each of the legitimate parameters mentioned above. If <post> is \relax, it will then not be appended at the end. Instead, each chosen parameter will be put into a {}- pair before preceded by the token <pre>. The reason for this exception is that while <pre> and <post> are meant to be the front and end macros for each selected parameter, in the case of <post> being \relax, we may regard <pre> as *acting upon* each chosen parameter.

The command \addressparas, however, uses \blockparas indirectly and will put the chosen parameters into a box of width <width>. If any selected parameter is longer than <width>, then the line will be wrapped around, with all wrapped potions being indented by an <indent>. This may be used to deal with very long address lines. Incidentally, we may use \textbox or \addressbox or \ADDRESSBOX to control the width of a single parameter. The general format reads

```
\textbox[<width]{<materials>}
\addressbox[<width>][<indent>]{<materials>}
\ADDRESSBOX{<width>}{<indent>}{<materials>}
```

where the meaning of the macro parameters are self-explanatory.

For the macro parameters, we have for convenience provided the following defaults for those in the squared brackets

```
<m>=1
<n>=1
<pre>=\noindent
```

```
<post>=\par
<width>= 8truecm
<indent>= 1.5em
```

This way, for instance, \paras[1][1] is equivalent to any of the following commands

```
\paras, \paras[1], \paras[][1],
\paras[1][], \paras[][]
```

and the following three commands are also the same

```
\addressbox{one address line}
\addressbox[8truecm]{one address line}
\addressbox[][1.5em]{one address line}
```

Incidentally this is the reason why we used the second parameter <n> to represent the group number: the defaults can often be dropped off more frequently. By default, all letter parameters specified will be put into a group environment before putting them into the form letter, unless \localparasfalse is issued beforehand.

To output a new letter entity, i.e a complete letter, we may use

```
\moreletter
  para-1-1;para-1-2;para-1-3; ...
 +para-2-1;para-2-2;...
 +   ...
 +para-n-1;para-n-2; ... ;para-n-m !
```

where ';' separate parameters inside a same group, '+' separates groups, '!' ends a cluster of parameters, and '...' represent further letter parameters. These three tokens are nothing special: we may change them to for instance '....', '----' and '====' respectively by \blockmarks, and change them back to '; + !' by \defaultmarks. For more general case, use

```
\delimiters{<A>}{<B>}{<C>}
```

where <A>, <B> and <C> are the new toks replacing '; + !' respectively. If we have to use these three special characters or tokens inside our *letter parameters*, then use \pstr, \gstr and \cstr respectively instead.

For the case of many clusters, we may enclose them, separated by any white spaces or empty lines, between \beginpilemode and \endpilemode so that we don't have to put \moreletter in front of each cluster of parameters.

We may also enter letter parameters line by line. In other words, we may essentially replace ';' in the \moreletter format by a normal line break. In this case, we shall put clusters of parameters between \beginblockmode and \endblockmode. Thus the following commands

```
\beginblockmode
   <para-1-1>
   <para-1-2>
     +       letter one

   <para-2-2>
     !

   <addr-1-1>
   <addr-1-2>
     !       letter two
\endblockmode
```

will produce the required form letter for each cluster of parameters. The general rules are as follows:

- If a line contains \endblockmode or ! or +, then everything else on that line is ignored and that line serves as an end/cluster/group marking line in the given priority order.
- Leading and trailing empty lines before or after a complete cluster will be ignored, unless ';' is there to force the next (empty) line as a starting parameter.

Obviously \blockmarks provides a more sensible delimiters for the blockmode, under which we could have

```
\blockmarks\beginblockmode
  .......    force next empty line active

 <para-1-2>
 ------- -  mark end of 1st group

 <para-2-2>
 <para-2-3>
 ========   end of cluster

 <other clusters>
\endblockmode
```

A simpler and perhaps more often encountered scenario is to use names and addresses only. In this case, there will be essentially only *one* group of parameters for each cluster. Thus we may use empty lines to delimit the clusters, and use line by line mode for the individual parameters. This way, each block of consecutive nonempty lines in

```
\beginlinemode
  <name-1>
  <address-1-1>
  <address-1-2>

  <name-2>
  <1st part of addr-2-1>  %
  <continued part of line addr-2-1>

  <name-3>      % letter three
\endlinemode
```

will output a new complete letter, corresponding to the specified letter parameters.

Ideally, the name–address entries will be produced by a database utility. Since they often contain special characters such as '#' and '$' explicitly, we may wish to change such characters in the parameters to the normal printable ones. For this purpose, we could use correspondingly the pair `\beginrawblockmode{<token>}` with `\endrawblockmode` or the pair `\beginrawlinemode{<token>}` with `\endrawlinemode`. If `<token>` is not empty, then we use `<token>` in place of `\endrawblockmode` or `\endrawlinemode`. In other words, `<token>` is the password to leave the raw text mode. Since database utility will in general produce a *fixed* number of `<m>` lines for each cluster of parameters, we may use `\begindatamode[<token>]{<m>}` and `\enddatamode` (or `<token>`) in place of `\beginrawlinemode` and `\endrawlinemode` so that each `<m>` raw text lines, including the empty ones, will eject a form letter.

It is often desirable that a letter form or format and the actual parameters be kept in different files: it will help in archiving and retrieving form letters. Furthermore, if a letter is quite long such that the TEX can't hold it in the memory, then we *have to* keep the letter content in a separate file. Essentially we need to save everything between `\beginletter` and `\endletter` into a separate file, say, `letter.let`, then later on use `\inputletter{letter.let}` to load in the letter format. Should one prefer to keep everything inside a *single* file, however, he may use `\beginfile[`*password*`]{`*file.ext*`}` and `\endfile` (or *password*) to create files such as `letter.let` mentioned above as a temporary scratch file. We note that `letter.let` will not contain `\beginletter` and `\endletter` as `\inputletter` will add them properly for us. Also the extension `.let` is preferred in this connection as it can indicate that this type of format file should not quite be regarded as the normal TEX files.

Likewise, it is also desirable to have a separate address or general letter parameter file. For the address file, we remark that we have to keep the pairing `\beginlinemode` with `\endlinemode` or `\beginblockmode` with `\endblockmode` inside the address file, if we are to use any of them for some content of that file. Moreover, an extension `.adr` is recommended for such files. Thus for a letter format or template saved in `letter.let`, we may choose `letter.adr` as the name for the related address file.

After a letter format is completed for sometime, one may have difficulty in recalling what those parameters should refer to when using the form letter again. For this purpose, we provide a simple mechanism with `\paranames` and `\showparas`. To be more precise, if one puts `\paranames name-1-1; ...; <etc>!` at the very beginning of the letter format, then one can use `\showparas` outside to output a letter with its parameters replaced by their names.

An alternative way, or rather a way of highlighting the positions of the letter parameters is to use `\preview`. Under `\preview`, or `\previewtrue`, all parameters given by `\paras[<m>][<n>]` will be boxed with double borderlines, similar to $\boxed{\boxed{2-3}}$, the number `<n>-<m>` inside (corresponding to `2-3` in the shown box) denotes the `<m>`-th parameter of the `<n>`-th group. However, the parameters given by `\blockparas` or `\addressparas` will be boxed with triple borderlines. Moreover, the macro parameters `<pre>` and `<post>` will also be displayed explicitly. Since `\addressparas` is essentially a special way of using `\blockparas`, it is highlighted via that for the `\blockparas`. In particular, `\blockparas[3][6][\x][\y]` for instance will display under `\preview` the following box

`6 - 3 (+): macro:->\x <<paras>> macro:->\y`

Sometimes we may want to have each given *empty* letter parameter automatically replaced by a particular *default* parameter. This can be easily done by using `\loaddefaultparas` inside your letter format, i.e. essentially between `\beginletter` and `\endletter`, before any calling of the letter parameters via `\paras` etc. Of course, the effect of `\loaddefaultparas` is nil unless a command `\paradefaults` is issued inside or (better still) outside the letter format. We note that `\paradefaults` takes the macro parameters in the exact way as `\paranames` does. One note of caution: `\paradefaults` and `\paranames` always use the default delimiters '`; + !`' no matter how you changed them via `\delimiters`. This is to avoid unnecessary complications.

We note that printing labels is no more than designing a special letter template. The mechanism provided in `formlett` however is to pick a subblock of parameters as the address. Basically all the form letter output inside `\beginlabels` and `\endlabels` will produce labels instead, taking the first parameter group as the address by default (see Appendix for more details). Incidentally, labels can be printed in single or double columns depending upon the choice of label width.

To conclude this section, we remark that if you want to put `\input` *filename* into a letter parameter, make sure that there is a *nontrivial* space following the *filename*. To be on the safe side, always use `\inputfile{`*filename.ext*`}` instead. Also `<!>`, when output to screen, is an invitation to read the log file for a warning message.

Example

The following is an example. If you are running plain TEX, add `\input formlett` at the top. But if you are running LATEX, then just add at the top the line

```
\documentstyle[formlett]{article}
\begin{document}
```

The line numbering on the right hand side is for reference only, as in the other occurrences later on. The example reads as follows.

```
\beginletter
\paranames              % optional
  \tt<<FULL NAME>>;%
  \tt<<ADDRESS-etc>>;%
 +\tt<<GIVEN NAME>>;%
  \tt<<MISSING ITEM>>;%
  \tt<<PHONE NUMBER>>!
\loaddefaultparas       % optional

\NOPAGENUMBERS\parindent=0pt
\noindent{\it\paras[1]}\par
\blockparas[2]\par\bigskip

Dear \paras[1][2],\par\medskip
We have been looking for
\paras[2][2] for quite a while
without any luck, could you help
us out? If so, please ring
\paras[3][2]. \par\medskip
Cheers!\hfill Michael\vfill\eject
\endletter
\preview \showparas
```

The above is a typical letter format or form letter. Command `\preview` in the last line gives

1 – 1

1 – 2 (+): `macro:->\noindent <<paras>> macro:->\par`

Dear 2 – 1,

We have been looking for 2 – 2 *for quite a while without any luck, could you help us out? If so, please ring* 2 – 3.

Cheers! *Michael*

and, the name of the parameters are displayed via `\showparas` as follows

`<<FULL NAME>>`
`<<ADDRESS-etc>>`

Dear `<<GIVEN NAME>>`*,*

We have been looking for `<<MISSING ITEM>>` *for quite a while without any luck, could you help us out? If so, please ring* `<<PHONE NUMBER>>`*.*

Cheers! *Michael*

In other words, we can have a good understanding about what a letter format does *without* reading its source code, which will be very handy for archiving and retrieving letter templates. To provide defaults for letter parameters and output new letters via `blockmode`, we may add

```
\paradefaults       % optional
  To whom this may concern
 +Sir or Madam;something;%
  061-225-9905!

\blockmarks
\beginrawblockmode{}

Mrs L Stenson
#1-20 Sunset Street
Hillside, Norway
------
Louise
a Bible
220-8888
=========

......

Above empty line active

\endrawblockmode
```

We note the parameters contain special characters #. That is the reason why we used `\beginrawblockmode`. Now suppose we have to wrap a very long address line, we could use `\addressbox` to control an *individual* parameter. The following is a typical case

```
\defaultmarks
\moreletter
   S Wales,;%
   \addressbox[2in][1em]{%
   University of Manchester Institute
   of Science and Technology, This is
   a long address line: it will be
   wrapped up automatically.}%
 +;a \TeX\ package \gstr\ manual and
   many more (\cstr)!
```

which is manifested in the following output

S Wales,
University of Manchester Institute
of Science and Technology, This
is a long address line: it will be
wrapped up automatically.

Dear Sir or Madam,

We have been looking for a TEX package + manual and many more (!) for quite a while without any luck, could you help us out? If so, please ring 061-225-9905.

Cheers! *Michael*

Please note that the default parameters have been used there. In the case of putting *only* name and address into a form letter, we may typically use `linemode` via e.g.

```
0057 \beginlinemode
0058   Z Jiang
0059   UNE, Arimdale
0060
0061   T Ribbons
0062   UMIST, Manchester
0063 \endlinemode
0064 \end{document}
```

If one saves lines 4-22 to file `myletter.let`, lines 25-63 or simply lines 57-63 to file `myletter.adr`, then we can for instance produce via LaTeX the form letter for multiple receivers by

```
\documentstyle[formlett]{article}
\begin{document}
\inputletter{myletter.let}
\showparas \preview
\paradefaults To whom it may concern!
\inputfile{myletter.adr}
\beginlabels % 1st parameter group as address
\inputfile{myletter.adr} % for labels
\endlabels
\end{document}
```

If one puts commands `\beginlables` and `\endlabels` at e.g. immediately after line 2 and 63 respectively, then one gets all the posting labels instead. If one puts commands `\beginfile{scratch.adr}` and `\endfile` there respectively, then by adding after `\endfile` the commands

```
\inputfile{scratch.adr}
\beginlabels \inputfile{scratch.adr} \endlabels
```

one gets both form letters and the labels. If one insists on not writing out auxiliary scratch files, then use `\input formlett.sty \initstyle` [*styles*]{`article`}{*preamble*} to replace `\documentstyle [formlett,` *styles*]{`article`} *preamble* `\begin{document}`, which will be valid for LaTeX but ignored for TeX , and will enable one to use `\labelsquit` at the end to read in the current document again with all the letters there converted into the corresponding labels. If you only want to execute certain commands the first time round (i.e. before `\labelsquit` re-reads the file again), use `\firstread`{*commands*} for this purpose.

Macros

One of the main technical features of this set of macros is the extensive use of arrays, both one dimensional and two dimensional. The one dimensional array `STK` that we use is essentially an user-defined stack, while the two dimensional arrays `LET` and `DEF` are more like structure or record in other programming languages. First let us make clear that an array (or stack) of name, say `ABC`, will contain elements `\ABC`m`~` for one dimensional and `\ABC`m`*`n`~` for two dimensional case, where m and n are some non-negative integers representing row and column numbers respectively. The first element, if necessary, will be used to denote the length of that raw. Thus, `\ABC1*0~` for instance will denote the number of elements in the first raw of a two dimensional array `ABC`.

Let us now look at how a mechanism of defaulting *macro* parameters is formulated. For this purpose, we first define a stack `STK` and its stack pointer `\STKcount` by

```
\def\make@STKcount{\csname newcount\endcsname
  \STKcount\global\STKcount=0\relax}
\ifx\STKcount\undefined@\def\next{%
  \make@STKcount}\else\def\next{}\fi\next
\long\def\push#1{\global
  \advance\STKcount1\relax
  \expandafter\gdef\csname STK\the
  \STKcount\string~\endcsname{#1}}
\def\popnil{\expandafter\let
  \expandafter\temp@macro\csname
   STK\the\STKcount\string~\endcsname
  \ifnum\STKcount>0\global\expandafter
    \let\csname STK\the\STKcount
    \string~\endcsname=\undefined@
    \global\advance\STKcount-1%
  \else
    \def\temp@macro{}\global\STKcount=0%
  \fi\relax }
\def\pop{\popnil\temp@macro}
```

The first four lines of code will ensure that the stack pointer will not be flushed if this useful subset of macros is loaded again by for instance another style file in the middle of a TeX document. With the above code, we can stack away anything by `\push`{*anything*} and later use `\pop` to recall them or `\popnil` to remove one element from the stack.

So, how can we make defaults for certain empty macro parameters? The simplest case is perhaps

```
\long\def\get@nepara[#1][#2]{{%
  \def\next@{#2}%
  \ifx\next@\empty\push{#1}\else
  \push{#2}\fi}\ag@in}
\long\def\get@para\left@#1\right@{%
  \def\check@{%
  \ifx[\next@
  \def\full@####1{\get@nepara[#1]####1}%
  \else
  \def\full@{\get@nepara[#1][#1]}\fi
  \full@}%
\futurelet\next@\check@}
```

Basically, `\get@onepara[<A>][<B>]` will push the default `<A>` to the stack `STK` unless `<B>` is nonempty. And `\get@para\left@<A>\right@` will look at the next token to be read, if it is character '`[`', then reads in a macro parameter in the form of `[<B>]`

and then push <B> to the stack STK, otherwise push the default <A> to that stack. For the more general case, we need to define

```
\long\def\do@nepara
  \left@#1\right@#2\p@r@end{%
  \gdef\p@r@data{#2}\global\advance
  \p@r@count1\get@para\left@#1\right@}
\def\ag@in{\ifx\p@r@data\empty
  \def\next@{\relax\getp@r@s\run@CMD}%
  \else\def\next@{\expandafter\do@nepara
       \p@r@data\p@r@end}%
  \fi\next@}
\def\run@CMD{\csname STK\the\STKcount
  \string~\endcsname}
\newcount\p@r@count
\long\def\st@ckparas#1\p@r@end{%
  \global\p@r@count=0%
  \gdef\p@r@data{#1}\ag@in}
```

so that \st@ckparas\left@<A>\right@... \left@<B>\right@... \p@r@end will look for parameters enclosed in [] one by one, exhausting all the defaults paired by \left@ and \right@. If future tokens inside [] are not found, or empty, the default tokens inside the corresponding pair \left@\right@ will be used. The new or default tokens will be pushed to the general-purpose stack STK for a later use, while \p@r@count records the total number of the default macro parameters. We note that on exit of \st@ckparas, the control is passed to \runCMD.

Once all macro parameters are pushed to the stack, we may transfer them to separate toks \p@r@one to \p@r@nine (an easier-to-use format) defined by

```
\newtoks\p@r@one\newtoks\p@r@two
\newtoks\p@r@three\newtoks\p@r@four
\newtoks\p@r@five \newtoks\p@r@six
\newtoks\p@r@seven\newtoks\p@r@eight
\newtoks\p@r@nine
\def\clrp@r@s{\global\p@r@one={}%
 \global\p@r@two={}\global\p@r@three={}%
 \global\p@r@four={}\global\p@r@five={}%
 \global\p@r@six={}\global\p@r@seven={}%
 \global\p@r@eight={}\global\p@r@nine={}}
```

where \clrp@r@s clears all the parameter toks. We can then retrieve parameters from the stack STK to the parameter toks via \getp@r@s with the following code

```
\newcount\temp@count
\def\getp@r@s{\temp@count=\p@r@count
{\loop
 \ifnum\temp@count>0
 \expandafter\let\expandafter
 \t@macro\csname STK\the\STKcount
   \string~\endcsname
 \ifcase\temp@count
 \or\global\p@r@one=\expandafter{\t@macro}%
 \or\global\p@r@two=\expandafter{\t@macro}%
 \or\global\p@r@three=\expandafter{\t@macro}%
 \or\global\p@r@four=\expandafter{\t@macro}%
 \or\global\p@r@five=\expandafter{\t@macro}%
 \or\global\p@r@six=\expandafter{\t@macro}%
 \or\global\p@r@seven=\expandafter{\t@macro}%
 \or\global\p@r@eight=\expandafter{\t@macro}%
 \or\global\p@r@nine=\expandafter{\t@macro}%
 \else
  \errmessage
  {Parameter capacity exceeded.}%
 \fi \global\expandafter\let
 \csname STK\the\STKcount
 \string~\endcsname=\undefined@%
 \global\advance\STKcount-1%
 \global\advance\temp@count-1\relax
 \fi
 \ifnum\temp@count>0\repeat}}
```

We are now ready to use the above mechanism to default macro parameters for some already defined macros. Suppose we have already defined a macro \crudemac[<A>][<B>] which takes two macro parameters <A> and <B>, and we want to define a new macro \smartmac which behaves like \crudemac but will default the macro parameters <A> and <B> to for instance 9 and 88 respectively. For this purpose, we may define \smartmac via

```
\def\smartmac{\push{%
 \edef\next@@{\noexpand
 \crudemac[\the\p@r@one][\the\p@r@two]}%
 \popnil\clrp@r@s\next@@}\st@ckparas
 \left@9\right@\left@88\right@\p@r@end}
```

This way, all the followings

```
\crudemac[9][88], \smartmac[9][88],
\smartmac, \smartmac[], \smartmac[][],
\smartmac[9], \smartmac[9][], \smartmac[][88]
```

are the same under the assumption that we shall not use characters '[' and ']' for the macro parameters themselves, and that \crudemac will not change any \catcode inside. The reason for this latter condition lies in the fact that when \smartmac looks at the next character, the character is in a sense already read. If \crudemac changes the \catcode of that particular character, then the one that is already read via \futurelet will have a misleading \catcode. If one has to change \catcode inside but wants to avoid this possible misbehavior, always put full number of empty brackets '[]' for the defaults, or put a \relax immediately afterwards. Better still, make the last parameter to appear in a mandatory '{}' form rather than the '[]' form via the #{ mechanism [1], so that those square brackets can be dropped off.

For example, we as a by-product also defined a 100% verbatim mode by

```
\begin@@rawlist[A][B][C][D][E][F]G
 <general text>
\endrawlist
```

where, for the line numbering of the raw text, A, ..., G represent respectively the initial value, increment step, number of digits, font, horizontal position of text, horizontal shift of numbering, and the password to leave raw text mode. Incidentally, `\linecount` records the current line numbering count. Our purpose is to define `\beginrawlist` so that its format is

`\beginrawlist`$[A][B][C][D][E][F]\{G\}$

in which A to F are *optional* while $\{G\}$ is mandatory even though G can be empty — implying `\endrawlist` is the password to leave the raw-text mode. For this purpose, we first define

```
\def\begin@rawlist[#1][#2][#3][#4][#5][#6]#{%
    \begin@@rawlist[#1][#2][#3][#4][#5][#6]}%
```

to make '{}' pair compulsory for `#7`, then we provide the defaults for A to F as `\linecount`, `1`, `0`, `\tiny`, `0pt` and `0pt` respectively via

```
\def\beginrawlist{\push{%
 \edef\next@@{\noexpand\begin@rawlist
 [\the\p@r@one][\the\p@r@two][\the\p@r@three]%
 [\the\p@r@four][\the\p@r@five][\the\p@r@six]}%
 \popnil\clrp@r@s\next@@}%  end of push
 \font\tiny@rm=cmr5%
 \edef\temp@macro{%
  \noexpand\left@\the\linecount\noexpand\right@
  \noexpand\left@1\noexpand\right@
  \noexpand\left@0\noexpand\right@
  \noexpand\left@\noexpand\tiny@rm
          \noexpand\right@
  \noexpand\left@0pt\noexpand\right@
  \noexpand\left@0pt\noexpand\right@}%
  \expandafter\st@ckparas\temp@macro\p@r@end}%
```

Likewise, we could also similarly define `\printfile[...]`{*filename.ext*} so that '`[...]`' takes a same defaulting scheme.

There are several tiny but very useful macros in `formlett` which are used again and again. One of such macros is for testing if one string contains another string. The following `\test@str` tests if string `#1` is contained in string `#2`

```
\newif\iftemp@if \newtoks\temp@toks
% return \temp@iftrue if yes,
%  \first@half,\second@half are global
\long\def\test@str#1#2{%
  \long\def\strip@endmark##1\s@fetymarkI
    #1\s@fetymark{\gdef\second@half{##1}}%
  \long\def\strip@markI##1\s@fetymarkI
    \s@fetymark{\gdef\first@half{##1}}%
  \long\def\p@rse##1#1##2\s@fetymark{%
    \gdef\first@half{##1}%
    \gdef\second@half{##2}%
    \ifx\second@half\empty
     \strip@markI##1\s@fetymark\temp@iffalse
    \else
     \strip@endmark##2\s@fetymark\temp@iftrue
    \fi}%
    \temp@toks={#2\s@fetymarkI#1\s@fetymark}%
    \expandafter\p@rse\the\temp@toks }%
```

The `\first@half` will contain the tokens upto but excluding the substring `#1`, while the `\second@half` will contain the tokens after the *first* appearance of `#1`. Should `#2` not contain string `#1` at all, then `\first@half` is the whole string `#2` whereas `\second@half` is empty.

The small macros such as the above one and the stack utilities, though built for the main purpose of `formlett`, provide also a good 'infrastructure' for other house-keepings. One of the sidekicks of this type is our macro for commenting out blocks of text inside a TEX file. Basically, `\begincomment`{*password*} will set *all* characters of code `0` to `\MAX@CHR@CODE` (=`255` by default) to catcode 11 or 12, and look for the *password*, or `\endcomment` in the absence of a password, to quit the comment mode. In comparison to a very compact macro `comment.sty` (currently at version 3.0) written by Victor Eijkhout, quitting comment mode via a password here seems more flexible.

As for the other macros in `formlett`, it is difficult to explain them without printing out all of them. So instead, we shall explain simply one of the strategies in parsing the letter parameters. Suppose W is a list of parameters separated by '`+`' and ended by '`!`', i.e. W is like `<A>+<B>+...+<S>!`. Then it will be easy to parse all the parameters one by one (to a stack for instance) if we can know whether there is precisely one parameter left over. To detect it, we use

```
\newif\ifl@stline
\long\def\testl@stline
  #1+#2!#3\s@fetymark{%
 \def\next{#2}\ifx\next\empty
 \l@stlinetrue\else\l@stlinefalse\fi
 \def\next{#3}\def\nextsample{+!}%
 \ifx\next\nextsample\l@stlinefalse\fi}
```

Then the command `\testl@stline` W`+!\s@fetymark` will set `\l@stlinetrue` if there is only one parameter left in W, and will set `\l@stlinefalse` if otherwise.

Finally, for those wizard users who want to do everything their own way, we just note that if for instance the 3rd letter parameter of the 2nd group of a cluster is given as `<A>`, then `\LET2*3~` will contain `\b@group\relax<A>\e@group` right after a cluster is read in. `\DEF2*3~`, on the other hand, contains the corresponding default parameter in the same fashion. Furthermore, the command `\checkparas`$[m][n]$`{LET}` will copy the content of `\paras`$[m][n]$, minus the 'wrapping' extra tokens `\b@group\relax` and `\e@group`, to `\cachedata` and set `\ifemptyparas` to true or false depending on whether the content is empty or not. This way, a user may even change the characteristics of his letter template by first testing the content of the supplied individual parameters. However, we note that if `\loaddefaultparas` is executed, then the `LET` array, when some of its elements are not supplied, will contain the corresponding elements of the `DEF` array. Hence care must be exercised under such circumstances, when interpreting the `\cachedata` generated by `\checkparas[`m`][`n`]{LET}`. If necessary, we may use `\delparadefaults` to delete current default parameter array `DEF` so as to conduct `\checkparas{LET}` more precisely. So if anyone wants to manipulate furthermore letter parameters inside a letter template, he or she will now at least know where to look at.

References

[1]. Knuth D E, *The TEXbook*, Reading, Mass., Addison-Wesley, 1992.

[2]. Piff M, Text merges in TEX and LATEX, *TUGboad*, 13(4):518, 1993.

[3]. Damrau J and Wester M, Form letters with 3-across labels capacity, *TUGboat*, 13(4):510, 1991.

Appendix

In the followings, we give a brief summary of the new commands given by `formlett` version 2.2.

Let m and n be numbers, p, q and r be dimensions, A, B, P, G, C and T be tokens, and X be a box. Furthermore, we shall denote by R a full set of letter parameters ended by '`!`', with '`;`' separating parameters inside a same group and '`+`' separating different parameter groups. We moreover denote R by F, when '`; + !`' there can be replaced by 'P G C' respectively if `\delimiters`$\{P\}\{G\}\{C\}$ is issued. In the commands tabulated below, the macro parameters contained in squared brackets support default. In particular, the defaults are m=`1`, n=`1`, p=`8truecm`, q=`1.5em`, r=`3pt`, A=`\noindent`, B=`\par`, `\previewfalse`, `\localparastrue` and `\nodefaultsfalse`.

\paras[m][n]	m^{th} parameter of n^{th} group
\blockparas[m][n][A][B]	m^{th} to the last parameter of n^{th} group, each preceded by A and followed by B, wrapped by {} if B=\relax
\addressparas[m][n][p][q]	m^{th} to the last parameter of n^{th} group, each put into a box of width p with indent q for wrapped portions
\loaddefaultparas	fill empty parameters with defaults
\checkparas[m][n]{T}	m^{th} parameter of n^{th} group copied to \cachedata; \ifemptyparas is true if element is empty; T is often LET or DEF
\moreletter F	use parameters F to output a new letter
\paranames R	use R as parameter names
\paradefaults R	use R as default parameters
\delparadefaults	delete default parameters
\delimiters{P}{G}{C}	use P, G, C as delimiters
\defaultmarks	use '; + !' as delimiters
\blockmarks	use '....', '----', '====' as delimiters
\preview	highlight parameter positions
\showparas	display parameter names, if any
\inputletter{*file.ext*}	input letter content
\inputfile{*file.ext*}	input *file.ext*
\beginletter \endletter	delimiters for letter content (template)
\beginpilemode \endpilemode	normal letter parameters cluster-wise
\beginblockmode \endblockmode	for line-by-line blocks of parameters, empty lines active within each cluster
\beginlinemode \endlinemode	for line-by-line parameters, empty lines delimit clusters
\beginrawblockmode{T} \endrawblockmode	raw text mode; nonempty T replaces \endrawblockmode to mark end
\beginrawlinemode{T} \endrawlinemode	raw text parameters and active spaces etc
\begindatamode[T]{m} \enddatamode	m raw text lines for one form letter
\begincolumns[a][b] \endcolumns	a-columns separated by distance b
\begincomment{T} \endcomment	comment out lines, nonempty T replaces \endcomment to mark last *full line* for comment
\beginfile[T]{*file.ext*} \endfile	write text verbatim to file *file.ext* (empty implies scr@tch@.tex), nonempty T replaces \endfile to mark last full line
\PAGENO=1	page number reset to 1
\NOPAGENUMBERS	no page numbers
\textbox[p]{*text*}	*text* into box of width p
\boxmore[r]{X}	add borderline to box X at a distance r
\addressbox[p][q]{*text*}	*text* into box of width p, with wrapped options indented by q
\previewfalse	given parameters instead of highlighted boxes will be shown
\localparastrue	letter parameters put into {} before displayed
\nodefaultsfalse	take defaults for empty letter parameters
\previewtrue \localparasfalse \nodefaultstrue	*cf.* above

Table continues in the next page

{ `\beginrawlist[`a`][`b`][`c`][`d`][`e`][`f`]{`g`}` { `\endrawlist`	
defaults: a=`\linecount`, b=`1`, c=`0`, d=`\tiny@rm`, e=`0pt`, f=`0pt`, g=`\endrawlist`	verbatim listing, line count from a, step b, digits c, font d, text position e, numbering shift f, end mark g
`\printfile[`a`][`b`][`c`][`d`][`e`][`f`]{`*file.ext*`}`	
defaults: c=`6` *etc*	verbatim listing of *file.ext*
{ `\beginlabels[`a`][`b`][`c`][`d`][`e`][`f`]` { `\endlabels`	
defaults: a=`20pt`, b=`\tt\raggedright`, c=`1`, d=`1`, e=`2.6in`, f=`2em`	form letters become labels: address taken from c^{th} to last parameter of d^{th} group, with width e, indent f, borderspace a and font toks b
`\firstread{`T`}`	toks T will not be read if the file is re-read via `\labelsquit`
`\initstyle[`a`]{`b`}{`c`}`	initiation for `\labelsquit`,with styles a, documentstyle b and preamble c
`\initclass[`a`][`o`]{`b`}{`c`}`	similar to `\initstyle` (o is LaTeX2e options), but retains native LaTeX2e when applicable
`\labelsquit[`a`][`b`][`c`][`d`][`e`][`f`]{`*file.ext*`}`	
defaults: see that for `\beginlabels`	quit after converting letters to labels by reading the current document or *file.ext*
`\newarray\`abc	make array named abc along with `\`abc and `\check`abc
`\delarray\`abc	delete array named abc
`\`abc`(`m`)={`T`}`	assign tokens T to `\`$abc(m)$; `\`$abc(m) \equiv$ `\`$abc(m_1, ..., m_k)$ for m=1 + $\sum_1^k (m_i - 1) *$ `\dataheight`$^{k-i}$
`\`abc`(`m`)`	value of `\`$abc(m)$, if not followed by '`=`'
`\dataheight`	current array dimension: *cf.* `\`$abc(m)$
`\readarray{`N`}{`a_1`&..&`a_n`}`	equivalent to `\`N`(1)=`a_1, .., `\`$N(n)$=a_n
`\check`abc`(`m`)`	copy `\`$abc(m)$ to `\cachedata`; set `\ifemptydata` to true/false accordingly
`\readstyles{`$f_1, ..., f_n$`}`	read files f_1 to f_n with default extension `.sty`; `@`'s catcode set to `11` during input