Paper CT04

SAS Keyboard Shortcuts Enhanced Using %Clipbrd() Macro Function

Jean-Michel Bodart, Business & Decision Life Sciences, Brussels, Belgium

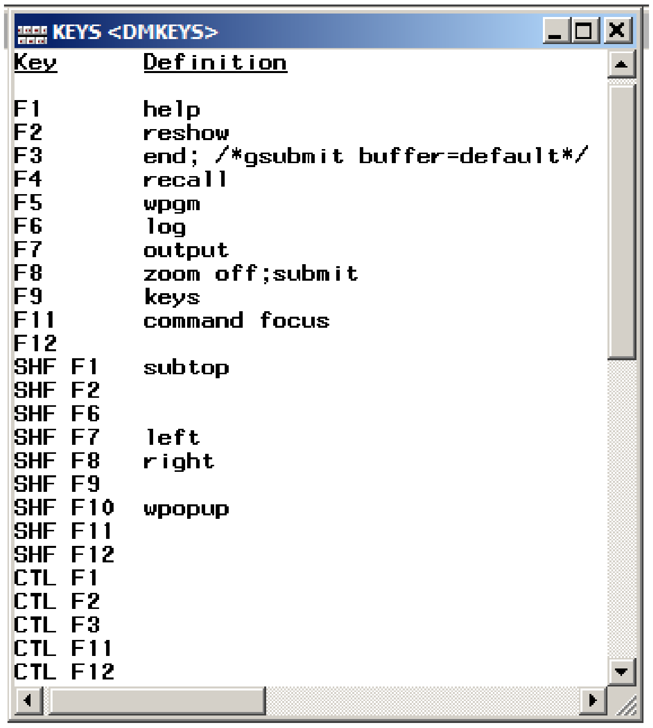
ABSTRACT

Clipboard management in SAS® for Windows® is rather limited: you can copy text from almost any window to the clipboard, cut from and/or paste to Program Editor, Enhanced Editor, Notepad Window, Command Bar, and various dialog boxes, and even submit clipboard contents as code for execution (using GSUBMIT command). SAS Online Help also describes the Clipboard access method of the Filename statement, with examples of reading or writing to the clipboard from a data step, as well as sending a procedure ODS output to the clipboard.

This paper describes the macro-function %Clipbrd() that can be used in user-defined keyboard shortcuts to trigger specific actions from the clipboard contents, such as opening a SAS file in a ViewTable window from its <libname.membername> reference, or an external file in the corresponding windows application from its path, seamlessly resolving any embedded macro-reference. Similar methods should be applicable to non-Windows versions of SAS.

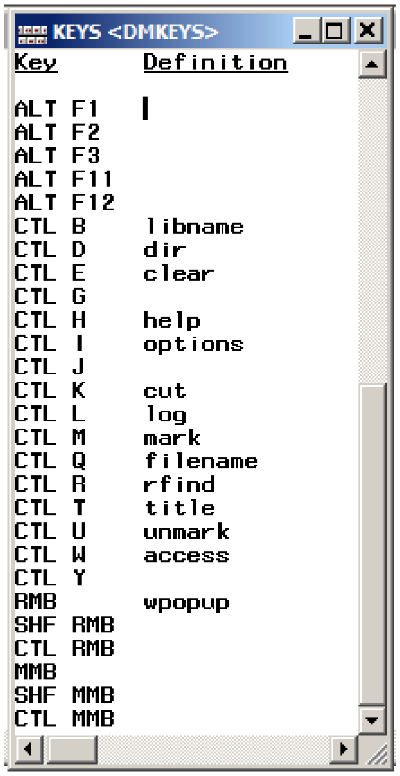
INTRODUCTION

In SAS, keyboard shortcuts can be assigned to function keys (such as F1 which by default opens the help window), or combinations of keys (such as Ctrl+L which by default brings up the Log window) using the KEYDEF command; they can also be modified and stored permanently using the KEYS window (accessed by issuing the KEYS command or via the menu Tools → Options → Keys). These shortcuts definitions can be common to multiple windows (e.g., the Program Editor, Enhanced Editor, Log, Output, Explorer windows) or specific to some other windows (e.g. the Graph and ViewTable windows). The KEYS window shortcuts available under Windows with their default definitions are presented below (where SHF, CTL, ALT, RMB, MMB respectively denote the Shift, Control (or Ctrl) and Alt keys, and the Right and Middle Mouse Buttons):



```
KEYS <DMKEYS>
Key       Definition

F1        help
F2        reshow
F3        end; /*gsubmit buffer=default*/
F4        recall
F5        wpgm
F6        log
F7        output
F8        zoom off;submit
F9        keys
F11       command focus
F12
SHF F1    subtop
SHF F2
SHF F6
SHF F7    left
SHF F8    right
SHF F9
SHF F10   wpopup
SHF F11
SHF F12
CTL F1
CTL F2
CTL F3
CTL F11
CTL F12
```



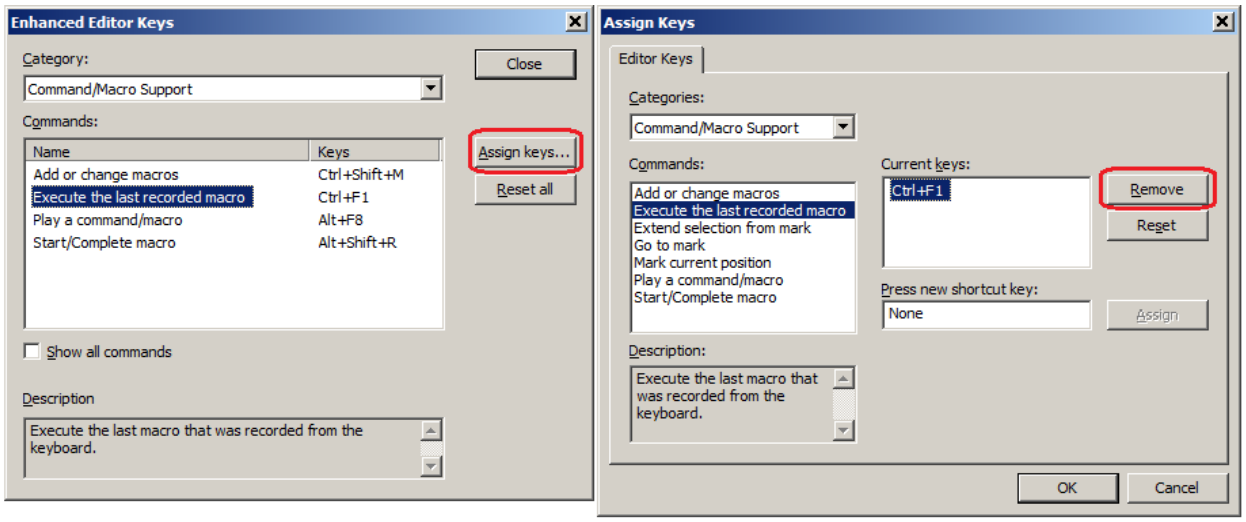
```
KEYS <DMKEYS>
Key       Definition

ALT F1
ALT F2
ALT F3
ALT F11
ALT F12
CTL B     libname
CTL D     dir
CTL E     clear
CTL G
CTL H     help
CTL I     options
CTL J
CTL K     cut
CTL L     log
CTL M     mark
CTL Q     filename
CTL R     rfind
CTL T     title
CTL U     unmark
CTL W     access
CTL Y
RMB       wpopup
SHF RMB
CTL RMB
MMB
SHF MMB
CTL MMB
```

This type of shortcuts can be used to execute SAS commands (the same commands that can be entered in the SAS Command Bar) and can resolve SAS macro code; otherwise they can also be used to insert pre-defined text strings at the current cursor location (when the definition consists of a tilde character (~) followed by the text string).

Some function keys (or combinations thereof) are assigned (and reserved) by the operating system (e.g. in Microsoft® Windows, Alt + F4 closes the current application).

On the other hand, the Enhanced Editor comes with a specific set of keyboard shortcuts that can be modified using menu: Tools → Options → Enhanced Editor Keys; this type of shortcuts can be used to send various commands to the editor and to execute keyboard macros, but not SAS macros, so we will not discuss them further, except to mention that when defined, they override the shortcuts assigned to the same keys in the KEYS window, preventing their execution from the Enhanced Editor. By default, the Keyboard Shortcuts for the Enhanced Editor include (and override) the following of the KEYS Window shortcuts: Ctrl + F1, Ctrl + F2, Ctrl + G, Ctrl + Y, F1, F2, Shift + F2. It's possible to "unmask" these KEYS Window shortcuts in the Enhanced Editor by de-assigning these keys from any commands available for Enhanced Editor Keys assignment. To do so, bring up the dialog box using Tools → Options → Enhanced Editor Keys, identify the command associated with the keys you want to free up; press "Assign Keys…"; select the command and the associated keys, and press "Remove".



SAS Macros[3] are a well-known, powerful feature of the SAS system that can be used for automating, repeating and/or controlling a program execution. The macro language has its own syntax, and a limited set of statements and built-in functions. Macro functions compute a result based on specified or default arguments, and return (or resolve to) a value, which can then either be assigned to a macro-variable, or used in user-written macro definitions or in open code such as within data steps, global statements and procedure calls. Nesting macro functions is also possible, i.e. the result of an inner macro function becomes the argument of an outer macro function. The macro language allows users to create new macro functions[2] and extend the available functionality.

The macro-function %Clipbrd() was developed as a generic utility function to retrieve the clipboard contents, in such a way that the returned value can be used in keyboard shortcut definitions (the variety that can be defined using the KEYS window). Such keyboard shortcuts would be used to trigger a specific action when the clipboard contains a reference to one or more SAS files or external files. Typically, they would be used with text fragments copied from the SAS Log or from a SAS program and may contain macro-references, as highlighted in log excerpts below:

```
55   /* Analysis Folder on personal drive (version-controlled programming environment */
56   %let analysis_dir=\\fileserver\home\&SYSUSERID\&_compound.\Extension;
57
[..]
Note: (CREATE_FORMATS): Codelist Metadata are retrieved from dataset ref_data.full_codelist_table (dated:
14APR2017:16:17:30.107).
[..]
206  /*****************************************************************************************/
207  /* Save log to a file */
208  /*****************************************************************************************/
209  DM LOG 'FILE "\\fileserver\CRO\ANALYSIS\&_compound.\Data\ADaM\Log\&program..log" REPLACE';
```

The most basic actions to be performed would be to open the corresponding file(s) in the corresponding default Windows application (according to their file extension) or to open the corresponding folder in a Windows Explorer window with the corresponding file selected, but more advanced actions for SAS files could be e.g. to print or retrieve the contents of a dataset; to sort it, summarize it, print it, compare it to a reference dataset, execute a program or scan a log file.

This paper will demonstrate the creation of custom keyboard shortcuts in SAS for Windows, designed to open either a SAS dataset in a ViewTable window, or an external file such as an Excel® workbook or a Word® document in the associated Windows application, when the clipboard contains the corresponding member name of filename.

THE CLIPBOARD ACCESS METHOD OF THE FILENAME STATEMENT

According to SAS Online Help[5], the FILENAME clipboard access method has the following syntax:

```
   FILENAME fileref CLIPBRD <BUFFER=paste-buffer-name>;
```

Where fileref is a valid fileref name, and the optional BUFFER option is not applicable under Windows.

Note however that it is not possible to specify additional options such as the logical record length LRECL (which is by default 256 characters under windows). Even setting OPTION LRECL=32767 (the maximum LRECL value on Windows) does not overcome this limitation. The following simple data step code only retrieves the first 256 characters of each line from the clipboard, and writes them to the log:

```
   option LRECL=32767;
   filename myclip clipbrd;
   data _null_;
      infile myclip length=l;
      length line $32767;
      input line $varying32767. l;
      put l= line=;
   run;
```

It's also possible (although not documented in SAS Online Help) to access the clipboard using **external files functions**[4] in a data step. The following code overcomes the 256 characters by line limitation[1]:

```
   data _null_;
      rc = filename( 'clippy' , ' ' , 'clipbrd' );
      fid = fopen( 'clippy' , 's' , 32767 , 'V' );
      length line $32767;
      do while( fread( fid ) = 0 );
         len = frlen( fid );
         rc = fget( fid , line , len );
         put len= line=;
      end;
      rc = fclose( fid );
      rc = filename( 'clippy' );
   run;
```

The equivalent to the above code can also be implemented as pure macro statements, outside of a data step. In addition, some checks are performed to report any problem in assigning the libref or opening it in read mode.

```
   %macro clip1;
      %local rc fileref fid len line;
      %let rc = %sysfunc( filename( fileref, , clipbrd ) );
      %if &rc eq 0 %then %do;
         %let fid = %sysfunc( fopen( &fileref, s, 32767, v) );
         %if &fid ne 0 %then %do;
            %do %while( %sysfunc( fread( &fid ) )=0 );
               %let len = %sysfunc( frlen( &fid ) );
               %let rc = %sysfunc( fget( &fid, line, &len ) );
               %put len = &len line = %superq( line );
            %end;
            %let rc = %sysfunc( fclose( &fid ) );
         %end; %else %do;
            %put failed to open clipboard fileref.;
         %end;
         %let rc = %sysfunc( filename( fileref ) );
      %end; %else %do;
         %put %sysfunc( sysmsg( ) );
      %end;
   %mend;
   %clip1;
```

Note the above-described methods only retrieve textual contents from the clipboard. If the clipboard only contains graphical elements, SAS fails to open the clipboard fileref. If the clipboard contains both text and images (e.g. a section copied from a Word document), only the text part is retrieved.

If the clipboard contains tabular information such as a range of Excel cells, a Microsoft Word table, or tab-separated values in Notepad, each row of the table is retrieved as a separate line; within a row, the contents of the cells (fields) are retrieved as separated not with horizontal tabs, but with sequences of 6 spaces. Therefore, retrieving the exact contents of individual cells can be somewhat inaccurate in case some cell contents have leading or trailing spaces, or contain embedded sequences of 6 spaces or more. Tabular data copied from a HTML file however seem to be handled in different ways depending on the browser used to display them (e.g. Internet Explorer vs Google Chrome).

RETURNING THE CLIPBOARD CONTENTS USING A MACRO FUNCTION

Macro functions, such as %INDEX(), %SUBSTR(), or %LOWCASE(), compute a result based on specified or default arguments, and return (or resolve to) a value, which is then used inside other user-written macro definitions or in open code such as data step, global statements and procedure calls.

User-written macro-functions such as %Clipbrd() are composed of macro-code that is purely processed by the SAS Macro Language Processor (they do not contain any DATA step or PROC step or global statement).

According to the author's experience, the value returned by a user-written macro function will be any text left after processing of the macro code by the macro language processor, that does not contain any (non macro-quoted) semi-colon. If the text left after macro-processing contains non macro-quoted semi-colons and one attempts to call the macro as a macro-function, the text prior to the first semi-colon will become the returned value (i.e. what the macro-function resolves to); the text following it will be considered as non-macro code submitted for execution (e.g. as global statement(s), data step or proc step code) after the returned value has been processed. For that reason, a macro cannot at the same time generate non-macro code for execution, and return a value based on that non-macro code execution. Practically speaking, a macro-function can only execute macro code; it cannot at the same time submit data steps, proc steps or global non-macro statements. However, the returned value can be used to form (part of) a SAS command, a data or proc step, or a global statement.

Note the value returned by a macro-function can only contain printable text characters, i.e. any tab, line-feed or carriage return characters that you may wish to include in the data are automatically converted into spaces.

In order to become a macro-function, the %clip1() macro definition described above must be modified so that:

- The lines of text retrieved from the clipboard are not written to the log, but are concatenated (with optional insertion of line separators) and temporarily stored in a "result" macro-variable
- At the end of the macro execution, the accumulated contents of that "result" macro-variable are referred to but left unprocessed (i.e. they become the return value of the macro-function)
- Except to report processing errors, no information should be printed to the log

Optionally, some additional processing of the text data retrieved from the clipboard might be useful for certain purposes. In %Clipbrd(), this is controlled with macro parameters:

- LINESEP = ***text*** (zero, one or more characters) (Default: a single space)
    - Specifies replacement ***text*** for line breaks in the returned string. Note that any non-printable characters (such as ASCII codes 10 (New Line) and 13 (Carriage Return)) would be automatically converted to spaces by the macro processor.
- TABS = Y/***n***/N/[***chars***]/***string*** (Default: N)
    - This parameter controls how text lines read from the clipboard are split into one or more individual fields. Specify Y to consider sequences of 6 spaces (assumed to originate from tab characters) as field separators in the source text (retrieved from the clipboard). This method can be inaccurate if sequences of spaces also occur in the absence of, or in addition to tabs in the original clipboard contents. When a sequence of more than 6 spaces is found, the first 6 will be replaced by a field separator. If the remaining spaces form a sequence of at least 6, the first 6 consecutive spaces will again be replaced by a field separator, and so on.
    - Specify ***n*** (an integer value >=2) to consider as a single field separator each sequence of at least ***n*** spaces between non-space characters or line boundaries in the source text. In that case, sequences of spaces can never be considered as multiple consecutive field separators.
    - Use N to consider any sequences of spaces or other characters to be part of the text data, not separators. In that case each line is considered equivalent to a single field.
    - Specify [***chars***] (one or more printable characters between square brackets) to have each of the specified characters considered as alternate input field separators.
    - Specify ***string*** (a sequence of characters without square brackets, other than Y or N or a number), to be considered as unique input field separator.
- FSEP = ***string*** (one or more printable characters). (Default: a single space)
    - Specify a ***string*** of one or more characters as output field separators in returned value; they will be used to replace the input field separators specified with the TABS parameter, if any are found. If FSEP is set to an empty string, each field separator identified is simply removed from the returned value (after applying field processing specified by PRXCHG, UNIFW and QUOTE parameters).
- TR = N/H/V (default: N)

  - This parameter specifies an optional transposition of the data retrieved from the clipboard, performed prior to applying actions specified by parameters PRXCHG, QUOTE, and/or UNQUOTE=Y.
  - Specify H to 'horizontalize' the data by transforming all line separators into field separators.
  - Specify V to 'verticalize' the data by transforming all field separators into line separators.
- PRXCHG = ***/perl regular expression/***
  - Set to a Perl Regular Expression of the form: **s/search_pattern/replace_text/** to perform string search and replace (see SAS function PRXCHANGE()[4] for supported components of Perl Regular Expressions). This is either performed globally on the whole clipboard contents, or separately for each line of text, or for each field, depending on the BY parameter setting.
- QUOTE = Y/N/S (default: N)
  - Specify Y for string quoting (adding double quotes) around the whole clipboard contents, the individual text lines, or individual fields depending on the BY parameter setting. Existing double quotes within the input text are doubled, similarly to SAS function QUOTE()[4].
  - Use N (default) to add no quotes.
  - Use S to add single quotes instead. Existing single quotes within the input text are doubled.
- DEQUOTE= Y/N (default: N)
  - Use Y to remove surrounding matching quotes around text, similar to SAS function UNQUOTE()[4]. Between the matching quotes, pairs of consecutive quotes are reduced to one quote, and non-matching quotes are retained. Text after the second matching quote (ending quote) is removed.
- UNQUOTE= Y/N/F (default: N)
  - Use N to maintain macro quoting and prevent contents such as unmatched symbols (parentheses, quotes) or special characters/ keywords found in the data from being inappropriately interpreted by the macro processor, causing syntax errors and/or unexpected results.
  - Specify Y for macro unquoting, to allow resolution of macro references embedded in the clipboard data – but make sure the clipboard does not contain possibly harmful contents. The macro unquoting is performed at the same time of the addition of the double/single quotes, if QUOTE is set to Y or S, so as to limit the impact of unmatched symbols.
  - Specify F for final macro unquoting of the returned value. This can be useful in case the returned text used within non-macro code results in an ERROR message: "`Statement is not valid or it is used out of proper order`" although the syntax of the code appears correct, and the presence of macro-quoting seems to be the only reason for the error.
- BY = L/F/T (default: L)
  - If L, specifies that the actions of parameters PRXCHG, QUOTE, and/or UNQUOTE=Y should be applied on individual Lines; if F, on individual Fields, or if T, on the Total clipboard contents.
- PREFIX = ***string*** (default: empty string)
- SUFFIX = ***string*** (default: empty string)
  - If a non-empty ***string*** is specified as PREFIX or as SUFFIX, the ***string*** is respectively pre-pended or appended to the individual Lines, Fields or Total contents according to parameter BY setting (L, F or T), after the actions specified by parameters PRXCHG, QUOTE, and/or UNQUOTE=Y have been applied. To avoid the need for macro-quoting, special characters can be specified as keywords between hash signs, that will be replaced by the corresponding characters: #c# (comma), #sc# (semicolon), #s# (space), #cs# (comma+space), #sq# (single quote), #dq# (double quote), #lp# (left parenthesis), #rp# (right parenthesis), #pc# (%), #am# (&), #h# (#).

UPDATING THE CLIPBOARD CONTENTS USING A MACRO FUNCTION?

A macro that accesses the clipboard could also replace its content after processing it, instead of actually returning the processed contents. The clipboard access method also allows writing text back to the clipboard, including line breaks and tabs as field separators, that would otherwise be lost in the returned value of a macro-function.

Unfortunately, any horizontal tab characters (the separator that should be used for pasting the different fields into distinct cells of an Excel spreadsheet or a Word table) used within macro code are automatically converted to spaces by the SAS macro processor, except if directly passed enclosed between double quotes. As a result, the tab character cannot be written directly to the clipboard using only macro code. Within a data step, it would be straightforward to preserve tab characters using external file functions FPUT() and FWRITE()[4].

When using only macro-code, a special technique must be used: the tab characters are first written enclosed between quotes at pre-calculated positions in the File Data Buffer (FDB) (using %SYFUNC(FWRITE()) function calls), then the field values are written between the tab characters into the FDB, with the first and last character of the value overwriting the previously written quotes. This implies that a tab cannot appear as first or last character of a line, as would be needed after a first empty field or before a last empty field. In such cases the first and last empty fields must be replaced by a single space. When all values and separators have been written to the FDB, its contents are written to the clipboard using %SYSFUNC(FWRITE()) function calls, which also add newline characters.

To update the clipboard contents instead of returning a value, use the following parameter:

- UPDATE = Y/N
    - If N (default), the clipboard contents are passed back as return value of the function.
    - If Y, the processed clipboard contents are written back to the clipboard, and can then be manually pasted where desired. The field values separator is specified by FSEP as usual, except that if FSEP is specified as \t, tabs are used as separators. LINESEP should be set to \n in order to generate newline characters. When UPDATE=Y, the macro-function returns an empty value, so %Clipbrd() can be called either as a macro statement (with no return value to be retrieved) or as a macro-function.

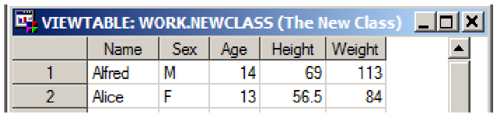
The UPDATE=Y option is designed mainly for use with FSEP = \t (so tabs are used as field separators) and the data are meant to be pasted into a spreadsheet or Word document table. If the data are meant to be pasted into the Program Editor or another character-based text editor that does not support tables, it would be best to right-pad all values with spaces to a common width (= the longest value in a specific field) so that they are nicely aligned. This can be requested by setting to Y an additional parameter UNIFW = Y/N (default= N).

USING MACROS IN KEYBOARD SHORTCUTS

The KEYS window shortcuts can execute SAS commands, including macro-variables references and macro-functions; they can even submit simple macro statements, non-macro code, or even calls to complex macros that will generate data and/or proc steps. The main limitation is the size of the SAS commands that can be assigned to these keyboard shortcuts: the maximum length appears to be 80 characters (under SAS for Windows, Version 9.3).

A command commonly used in keyboard shortcuts is one that opens the last created dataset in a VIEWTABLE window, displaying as title the libname and dataset name in uppercase, followed by the dataset label (if any is defined) between parentheses. The last created dataset is retrieved from the automatic macro-variable &syslast, e.g.:

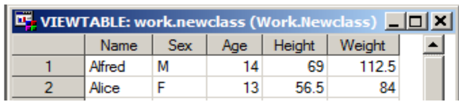
```
   vt &syslast
```


VIEWTABLE: WORK.NEWCLASS (The New Class)

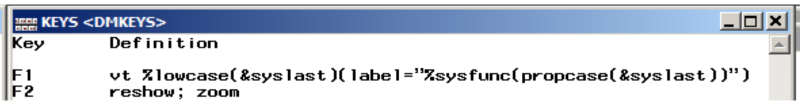
	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	113
2	Alice	F	13	56.5	84

To illustrate the use of standard macro-variables and macro-functions in a command assigned to the F1 key shortcut, the following fancy variation displays the libname and dataset name in lowercase, with libname and dataset name in proper case used as label:

```
   vt %lowcase(&syslast)(label="%sysfunc(propcase(&syslast))")
```


VIEWTABLE: work.newclass (Work.Newclass)

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84


KEYS <DMKEYS>

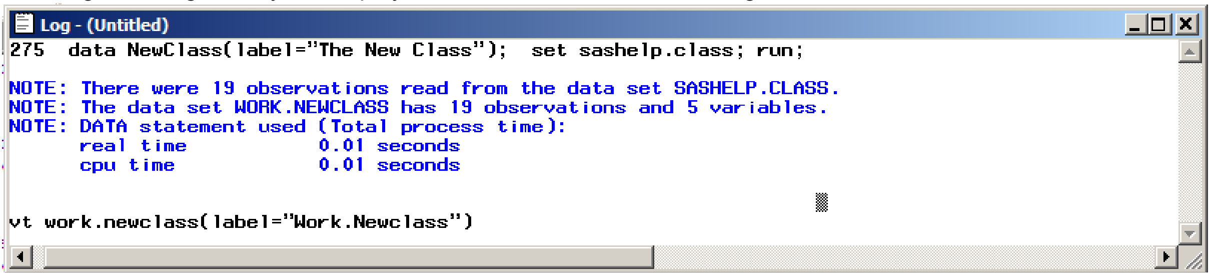
```
Key      Definition

F1       vt %lowcase(&syslast)(label="%sysfunc(propcase(&syslast))")
F2       reshow; zoom
```

In case of unexpected results, it is possible to check how SAS resolves the macros used in the command, by adding a %put (macro-statement) in front of the command assigned, and a semi-colon at the end:

```
   %put vt %lowcase(&syslast)(label="%sysfunc(propcase(&syslast))");
```

Pressing the assigned key will display the resolved command in the log window:


Log - (Untitled)

```
275  data NewClass(label="The New Class");  set sashelp.class; run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.NEWCLASS has 19 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


vt work.newclass(label="Work.Newclass")
```

Another useful shortcut is one that opens an external file (usually an output that SAS has just created) in the default Windows application associated with that filetype. The full path including filename and extension can be specified via a user-assigned macro-variable with a standard name. In this example, an excel output is created in the SAS WORK library path (retrieved using the PATHNAME() function):

```
   %let output=%sysfunc(pathname(work, L))\test.xls;
   proc export data=sashelp.class outfile="&output" dbms=EXCEL replace;
      sheet = "Class";
   run;
```

The following code could be submitted from the SAS editor to open the external file "&output" in the default Windows application associated with that filetype:

```
   x start " " "&output" & exit;          or     %sysexec(start " " "&output" & exit);
```

The x statement, the equivalent x command, and the %sysexec() standard macro-function all pass their argument as an external command to the operating system. The whole command does not have to be enclosed in quotes. But the filenames and paths containing spaces must be enclosed in double quotes. The Windows operating system executes the external command in a "Command Prompt" window. The addition of " & exit" at the end of the command ensures that the “Command Prompt” window automatically closes (even if SAS system option XWAIT is active) when the previous part of the command completes. Under Windows, the start command opens the specified file in the application associated by default with that file type, but when its first argument (here, a single space between double quotes) is between quotes, it is assumed to be the title for the application to be started, while the second argument (here the external file specified by macro-variable &output), quote or not, is considered as the file to be opened.

The X command as well as the %sysexec() macro-call can be directly used in a SAS command or assigned to a KEYS window shortcut[A]. In KEYS window shortcuts, multiple SAS commands must be separated by semicolons, but a final semicolon (after the last command or a single command) is not required.

The following command will open the specified (text-based) output file in SAS Enhanced Editor:

```
   wedit "&output"
```

Finally, the following can be used as a shortcut command to open a new Windows Explorer window with the specified output file selected in its parent folder:

```
   %sysexec(start explorer /select,"&output" & exit);
```

or `x start explorer /select,"&output" & exit;`

USING THE CLIPBOARD CONTENTS IN KEYBOARD SHORTCUTS

When any text corresponding to a dataset name (including libname if applicable), or an external file with its path and extension, is copied into the clipboard from a SAS Editor, Enhanced Editor, Log window or any other source, no matter whether it contains references to SAS macro-functions or macro-variables (provided they are defined in the current SAS session), that text can be retrieved from a call to the %Clipbrd() macro-function included in a KEYS window shortcut command. The user only has to select the appropriate text, copy it (usually using the standard Ctrl + C shortcut), and press the corresponding SAS-assigned function key or keyboard shortcut, to open the dataset or external file in the appropriate application, without having to know its exact location, to resolve any embedded macro reference, to find the file or dataset in SAS or Windows explorer and to double-click on it, or to type the SAS command including the dataset name or filename in a SAS command line or Command Bar.

The shortcut commands described previously can be adapted to use the dataset or external file specified by the contents of the clipboard instead of using the macro-variables &syslast or &output:

- Open a dataset in ViewTable window:
  `vt %clipbrd();` or `%clipbrd(prefix=vt#s#, suffix=#sc#);`[B]
- Open an external file in the associated (default) Windows application:
  `x start " " %clipbrd(unquote=Y, quote=Y) & exit;`
  or `%clipbrd(prefix=x start " "#s#, unquote=Y, quote=Y, suffix= & exit#sc#);`[B]
- Open an external (e.g. text, program, log) file in a new SAS Enhanced Editor window:
  `wedit %clipbrd(dequote=Y, quote=Y, unquote=Y)`
  or `%clipbrd(prefix=wedit#s#, dequote=Y, quote=Y, unquote=Y, suffix=#sc#)`[B]
- Open the location of an external file in a new Windows Explorer window, with that file selected:
  `%sysexec(start explorer /select,%clipbrd(unquote=Y, quote=Y) & exit)`
  or `%clipbrd(prefix=x start explorer /select#c#, unquote=Y, quote=Y, suffix=#am# exit#sc#)`[B]

Note that external files may have been copied with or without double quotes to the clipboard, but double quotes will be mandatory if the path/filename contain spaces, therefore it is prudent to have any existing double-quotes removed (using parameter DEQUOTE=Y), then new double-quotes added (with QUOTE=Y). Resolving embedded macros would require UNQUOTE=Y, but if a filename or path contains the “%” or “&” character such as in “R&D” or ”%change” you may need to use UNQUOTE=N to prevent SAS from attempting macro-variable resolution.

FURTHER ENHANCEMENTS

To avoid creating too many shortcuts variations based on %Clipbrd(), and instead have a few multi-purpose shortcuts, a macro-parameter CHK_EXIST = Y/N/EXT/SAS/MEM/DATA/VIEW/CAT/ACCESS/CENTRY/LIBREF/FILEREF/URL/***centry_type*** (default = N) could be added in order to make the macro %Clipbrd() try and identify whether the clipboard contents refer to one or multiple existing SAS files, catalog entries or external files, librefs, filrefs, or URLs, or even fragments of path and filenames split across multiple lines that should first be re-concatenated; that they exist and are of the specified types, and if so include them in the returned string.

Another parameter CHK_OPEN = Y/N/WIN/EXPL/SAS (default: N) could specify whether a command to open the retrieved files or SAS elements should be included in the return value, and whether they should be preferentially opened respectively in SAS, in the associated Windows application, selected in Windows Explorer or simply opened in SAS using a “preferred” way implemented in the macro, but that could be customized using a configuration file.

[A] The equivalent statements X and %sysexec() could also be called in a sas program; this can be handy for testing purposes.

[B] It is possible to have %Clipbrd() build and return the complete SAS command, by specifying its starting and ending components using the PREFIX and SUFFIX parameters; doing so, if multiple files or datasets are retrieved (from multiple fields or lines in the clipboard), multiple complete SAS commands can be generated and concatenated.

To simplify calls, presets of parameter values commonly used together for specific tasks could be defined.
Finally the user could have the possibility to review, adjust and confirm the parameters values to be applied in specific circumstances in a macro window (using %WINDOW[3] and %DISPLAY[3]) after %Clipbrd() has been started by a generic shortcut, with a parameter CONFIRM =Y/***names of parameters to be confirmed*** (default: empty).
The following command could then be assigned to a very generic KEYS window shortcut, or even to a Customized Tool in the SAS Application Toolbar (using menu Tools → Customize):

```
   %clipbrd(chk_exist=Y, chk_open=Y);
or %clipbrd(chk_exist=Y, chk_open=Y, confirm=Y);
```

CONCLUSION

User-defined macro-functions can be used to add new functionality to the SAS programming environment. The macro-function %Clipbrd() takes advantage of the macro-language equivalent to the CLIPBOARD access method of the FILENAME statement and makes the textual contents of the clipboard available as its returned value.
The retrieved contents can then be easily used in building SAS commands, and associated to keyboard shortcuts using the KEYS window, or to a Customized Tool in the SAS Application Toolbar.
The most immediately useful shortcuts are related to opening SAS files and external files using the appropriate applications and tools, and examples appropriate for the Windows environment have been presented; it should however be relatively easy to adapt them to other environments.
An additional benefit of the macro is the possibility to re-format text contents by applying quoting, unquoting, macro-variables and macro-functions resolution, pattern substitution through Perl Regular Expressions, transposition to vertical or horizontal structure, and space padding to uniformized field widths.
Some of these benefits are available via the returned value of the macro-function only in a rudimentary way (it lacks direct support for both newline and tab characters), but also and more powerfully via the clipboard update method (which supports both newline and tab characters).
The possibility to convert (tab-)delimited data into text-aligned, fixed width columns (and the other way around) can also facilitate transfers of information between table-oriented applications and documents (such as Word and Excel), and character-oriented editors such as SAS Enhanced Editor; this can be quite handy to copy specifications such as variable attributes definitions (especially labels) and formats definitions into well-structured SAS programs.
Further possible enhancements have also been outlined.

REFERENCES

[1] Tabachneck, Arthur and Kastin, Matt, “Copy and Paste from Excel to SAS” (Paper RF-04-2013), MWSUG24, 2013.
[2] Bodart, Jean-Michel. "Top 10 uses of macro %varlist - in proc SQL, Data Step and elsewhere”, PhUSE 2016, CS09.
[3] SAS Institute, "SAS 9.3 Macro Language: Reference"
[4] SAS Institute, “SAS 9.3 Functions and Call Routines: Reference”
[5] SAS Institute, “SAS 9.3 FILENAME, CLIPBOARD Access Method”

RECOMMENDED READING

The full macro code is available at http://www.phusewiki.org/wiki/index.php?title=SAS_macro-function_%25CLIPBRD

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jean-Michel Bodart
Business & Decision Life Sciences
Rue Saint-Lambert 141
1200 Brussels (Belgium)
Work Phone: +32 (0)2 774 11 00
Fax: +32 (0)2 774 11 99
Email: jean-michel.bodart@businessdecision.be
Web: www.businessdecision-lifesciences.com