Paper 099-2011

COMMAND Your Session: SAS® Commands and Command-Style Macros

Benjamin Neely, RTI International, RTP, North Carolina

ABSTRACT

SAS commands are used to accomplish various tasks within an interactive SAS session. Often, users issue commands and are completely unaware of the command that has been executed. One such example is pressing the print icon on the toolbar. Once this icon is pressed, the active window is printed to the default printer location by submitting the command DMPRINT from the toolbar. When users become familiar with SAS commands, commands can provide a flexible, efficient way to accomplish an array of tasks. Perhaps most importantly, users are not limited to the commands provided in the product installation. The macro facility provides an environment where users can create their own commands to carry out highly customized tasks. Within this paper, I discuss several topics related to SAS commands; specifically, how to submit them from four different locations, how to find a particular SAS command when the documentation is sparse, and how to create customized SAS commands using the macro facility.

INTRODUCTION

SAS users are probably familiar with statements, procedures, and functions, but may not be as familiar with another tool available to them: commands. Commands are used to perform specific tasks – such as printing results or opening a new enhanced editor window. In many instances, these tasks are window-specific. This means that most commands will work only when a specific window is active (e.g. the enhanced editor window). SAS provides over 30 windows and each window has a suite of commands. Many of these windows are not accessible through the menu, but all are accessible using the command language.

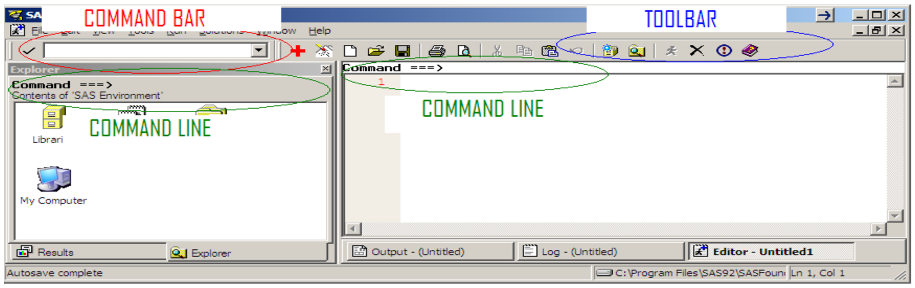
The first section of this paper describes how and where to submit commands. SAS provides four locations where commands may be submitted. All four locations are discussed, as well as their advantages and disadvantages. A potentially frustrating aspect of using the command language is not being able to find the right command for a desired task. In the second section of this paper, readers are pointed toward several resources for finding a suitable command. The final section describes how to create customized commands using the macro facility. It is important to note that SAS support for commands can be dependent on a user's operating system – for this paper Windows commands are used.

HOW AND WHERE TO SUBMIT COMMANDS

SAS provides users with four locations to submit commands: the command bar, command line, toolbar, and keyboard. In each of these locations it is important that users note which window is active because commands are window-specific. Each of these locations has advantages and disadvantages. Display 1 illustrates the three locations that are located on the user interface. Due to the default settings in SAS, the easiest location to submit a command is the command bar. For this reason, the SAS command tutorial begins from this location.

ISSUING A COMMAND FROM THE COMMAND BAR

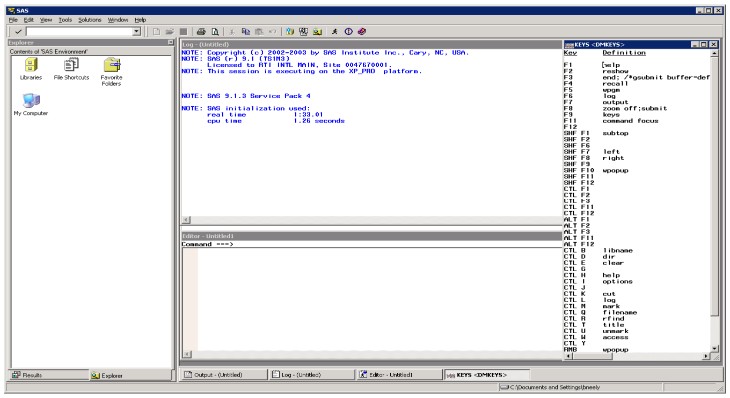
The command bar is a command-line interface useful for issuing commands at startup. Another advantage of this location is the drop down menu which allows users to issue previously used commands. The disadvantage of this location is that users must be aware of which window (i.e. output, log, or enhanced editor) is active as commands are window-specific. Therefore, if a user's log window is active (window header is blue) then typing a command on the command bar will only perform an action supported within the log window. This can sometimes cause confusion. An example of issuing a command from this location is typing the following command: NUM. If the active window is the enhanced editor, then number lines will appear. However, if the active window is the log window then nothing will happen.



Display 1. SAS User Interface

ISSUING A COMMAND FROM THE KEYBOARD

The keyboard is a great location for issuing commands that are frequently used, especially if the task needs to be accomplished quickly. Commands can be issued from the keyboard by assigning shortcuts to certain keys so that when these keys are pressed, a command is executed. The disadvantage of the keyboard method is the need to remember which key issues which command. For this reason, it is suggested that users exercise parsimony when assigning commands to keys. By default, SAS assigns predetermined commands to keys. Users are able to view these preloaded commands by typing the command KEYS in the command bar and pressing enter. The KEYS window can be seen in Display 2.



```
NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) 9.1 (TS1M3)
      Licensed to RTI INTL MAIN, Site 0047670001.
NOTE: This session is executing on the XP_PRO  platform.



NOTE: SAS 9.1.3 Service Pack 4

NOTE: SAS initialization used:
      real time           1:33.01
      cpu time            1.26 seconds
```

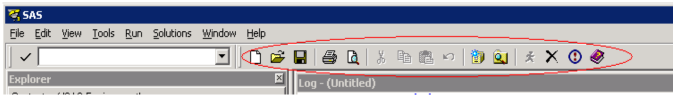
```
KEYS <DMKEYS>
Key         Definition

F1          help
F2          reshow
F3          end; /*gsubmit buffer=def
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
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
```

Display 2. KEYS Window in SAS

Towards the bottom of the KEYS window is a command listed as LIBNAME and should be assigned to the keys CTRL B. This illustrates the relationship between a command and the keyboard. To issue the command LIBNAME, all users have to do is hit the CTRL button and B at the same time. This will bring up the LIBNAME window which is helpful in exploring various data libraries. Users also have the ability to change key / command relationships by entering any command next to a key and saving the window (File -> Save). This allows each user to customize their keyboard to issue their most frequently used commands. Each task can then be accomplished efficiently.

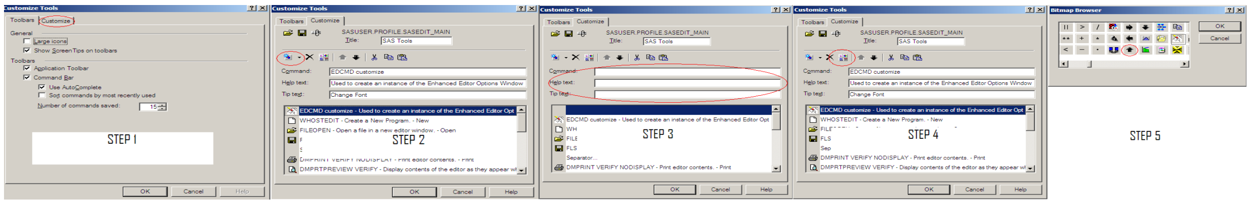
ISSUING A COMMAND FROM THE TOOLBAR

Issuing commands from the toolbar is quick and easy. Instead of selecting a key, users select an icon from the toolbar which submits and executes a command. This provides a major advantage over using the keyboard – the toolbar includes an icon that helps identify the submitted command (See Display 3). For example, the white blank document creates a new editor window. Every time this tool is used the command *WHOSTEDIT* is submitted. The disadvantage of using this method is that the user is limited by the number of icons that can fit on the toolbar.



Display 3. SAS Toolbar

Like keyboard commands, users can customize the toolbar. Customization is as easy as right-clicking the blank space to the right of the existing tools and selecting CUSTOMIZE. This will bring up the CUSTOMIZE TOOLS window (See Display 4 – Step 1). After the Customize Tools window is instantiated, select the CUSTOMIZE tab (Step 1) and then click on the ADD TOOL icon (Step 2) – this will bring up a blank tool (Step 3). Users must then fill in three fields: Command, Help Text, and Tip Text. In the "Command" field, enter the command to be executed every time the tool is pressed. The "Help text" field should contain enough information to help anyone understand the tool. Finally, the "Tip text" field is the text that is shown when users hover over the icon. Next, users should select the CHANGE ICON tool (Step 4) to instantiate the Bitmap Browser window (Step 5). This window contains hundreds of graphics that can be used to help remember the command that is issued when the customized tool is pressed. Once a graphic is chosen, click OK -> Click the Hard Disk Icon to save -> OK. The new tool will appear in the toolbar and can be used to submit a user-defined command.



Step 1:
General
[ ] Large icons
[✓] Show ScreenTips on toolbars
Toolbars
[✓] Application Toolbar
[✓] Command Bar
[✓] Use AutoComplete
[ ] Sort commands by most recently used
Number of commands saved: 15

Steps 2 and 4:
SASUSER.PROFILE.SASEDIT_MAIN
Title: SAS Tools
Command: EDCMD customize
Help text: Used to create an instance of the Enhanced Editor Options Window
Tip text: Change Font

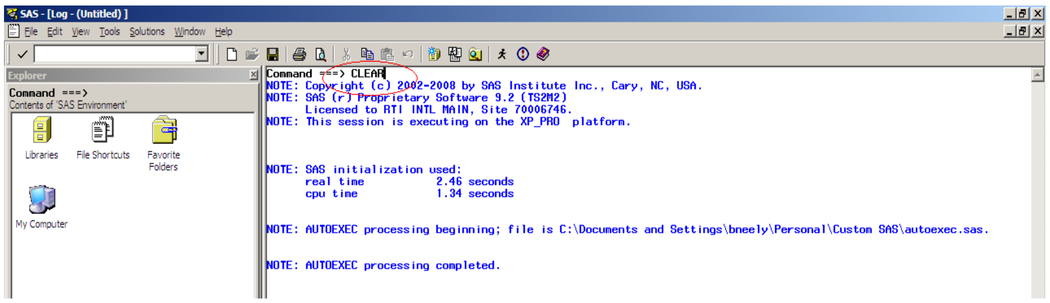
EDCMD customize - Used to create an instance of the Enhanced Editor Opt
WHOSTEDIT - Create a New Program. - New
FILEOPEN - Open a file in a new editor window. - Open
DMPRINT VERIFY NODISPLAY - Print editor contents. - Print
DMPRTPREVIEW VERIFY - Display contents of the editor as they appear wh

Step 3:
SASUSER.PROFILE.SASEDIT_MAIN
Title: SAS Tools
Command:
Help text:
Tip text:

Display 4. Customize Tools Window – How to Customize a Tool Step-by-Step

ISSUING A COMMAND FROM THE COMMAND LINE

In many SAS installations, the default action is to hide the command line (see Display 1). Users can choose to unhide this command-line interface by typing the command PMENU OFF in the command bar and pressing enter. After this command has been submitted, the reader's user interface should contain a command line on all default windows (i.e. results, explorer, output, log, and enhanced editor). Issuing a command from the command line is very similar to using the command bar. To retrieve the command line, type the command PMENU OFF in the command bar at startup and hit enter. To issue commands using the command line, type the desired command to the right of ==> and hit enter. For example, users often wish to delete the information typically found in the SAS Log window after startup before continuing with their program. To clear this window, use the command *CLEAR* on the command line in the log window. The Log window should be blank once executed.



```
Command ===> CLEAR
NOTE: Copyright (c) 2002-2008 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.2 (TS2M2)
      Licensed to RTI INTL MAIN, Site 70006746.
NOTE: This session is executing on the XP_PRO  platform.



NOTE: SAS initialization used:
      real time           2.46 seconds
      cpu time            1.34 seconds


NOTE: AUTOEXEC processing beginning; file is C:\Documents and Settings\bneely\Personal\Custom SAS\autoexec.sas.


NOTE: AUTOEXEC processing completed.
```

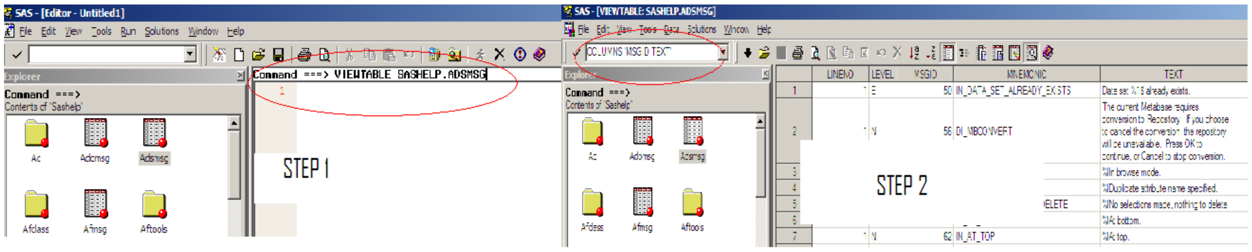
SAS - [Log - (Untitled)]
File Edit View Tools Solutions Window Help
Explorer
Command ===>
Contents of 'SAS Environment'
Libraries
File Shortcuts
Favorite Folders
My Computer

Display 5. Issuing the command CLEAR from the Command Line in the Log Window

Although the command line shares many of the same advantages and disadvantages as the command bar, it has one major advantage – it accepts the command PREVCMD. This command allows a user to execute a command using the keyboard, toolbar, or a menu-action and immediately recall the submitted command. This allows users to determine the syntax of the command issued without having to search through SAS literature or documentation. This powerful command will be discussed in detail in the section SAS Command Resources.

SAS COMMAND RESOURCES

The best overall resource for SAS commands can be found in the "SAS Command Reference" and "SAS Window Reference" of the *SAS Help and Documentation* compiled HTML file (just click the Help tab at the top left of the SAS user interface, then SAS Products -> Base SAS). Within the "SAS Command Reference" tab, users can find descriptions, syntax, and command arguments for many SAS commands. This is a very good starting place for novice command users. In addition, the "SAS Window Reference" is a good resource for learning about all of the windows available in SAS and how to instantiate an instance of each one using the command language. I've found the command VIEWTABLE to be an invaluable resource (and all of the associated window-specific commands). Display 6 is an example of using the VIEWTABLE command to review the data set SASHELP.ADSMSG.



```
Command ===> VIEWTABLE SASHELP.ADSMSG
```

COLUMNS 'MSGID TEXT'

	LINENO	LEVEL	MSGID	MNEMONIC	TEXT
1	1	E	50	IN_DATA_SET_ALREADY_EXISTS	Data set %1$ already exists.
2	1	N	56	DI_MBCONVERT	The current Metabase requires conversion to Repository. If you choose to cancel the conversion, the repository will be unavailable. Press OK to continue, or Cancel to stop conversion.
3					%1In browse mode.
4					%1Duplicate attribute name specified.
5				ELETE	%1No selections made, nothing to delete
6					%1At bottom.
7	1	N	62	IN_AT_TOP	%1At top.

SAS - [Editor - Untitled1]
File Edit View Tools Run Solutions Window Help
Explorer
Command ===>
Contents of 'Sashelp'
Ac
Adomsg
Adsmsg
Afclass
Afmsg
Aftools
STEP 1
SAS - [VIEWTABLE: SASHELP.ADSMSG]
File Edit View Tools Data Solutions Window Help
STEP 2

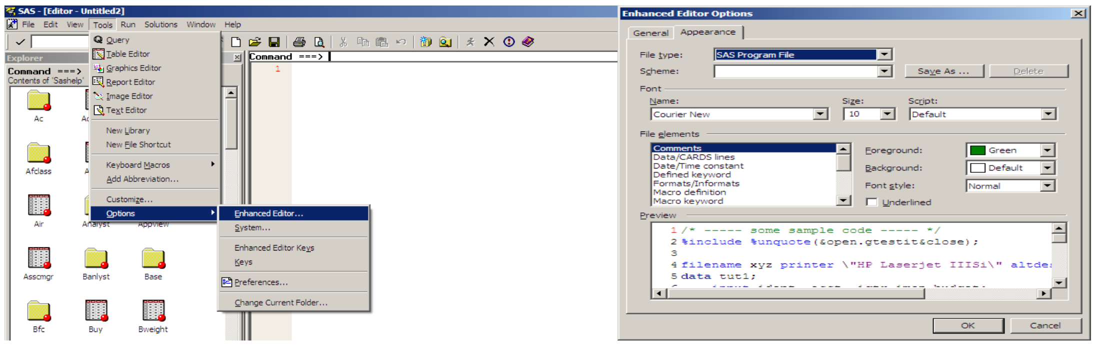
Display 6. Using VIEWTABLE to review the data set SASHELP.ADSMSG

Step 1 illustrates use of the command VIEWTABLE and the optional parameter <data set>. After executing this command, the VIEWTABLE window opens with the data set SASHELP.ADSMSG loaded. Using the window-specific command COLUMN 'MSGID TEXT' in the command bar, users are able to view only the variables that were entered as parameter to the command COLUMN (i.e. MSGID and TEXT). This is a particularly useful feature if a data set contains hundreds of variables.

While the aforementioned resources are very helpful, it is sometimes hard to find the right command to mimic a menu action because command names can sometimes be cryptic. For example, many users may wish to choose custom colors and fonts to use in their enhanced editor. To accomplish this task, users can submit the command EDCMD <*options*> if the enhanced editor window is active. However, browsing through the HTML help file, the functionality of the command EDCMD is not immediately obvious from the name alone. In situations like these, the PREVCMD (or *?)* command may be used to circumvent the issue.

THE PREVCMD/? COMMAND

Earlier in this paper I claimed that the biggest advantage of the command line was the ability to issue the command PREVCMD. According to the help file, PREVCMD may be used to "recall the last command entered on the command line of any window". However, it can also be used to display the commands associated with many menu actions. The user can take advantage of this feature to determine which command can be used to customize the colors and fonts of their program editor. By first running the desired menu action and the entering PREVCMD in the command line, users can not only mimic the menu action, but they can also customize the action through the use of commands. Display 7 illustrates using menu actions to bring up the Enhance Editor Options window.



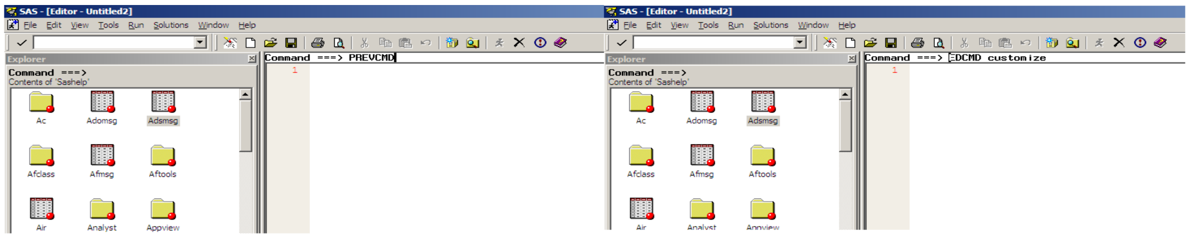
```
Command ===>
```

File type: SAS Program File
Scheme:
Font
Name: Courier New Size: 10 Script: Default
File elements: Comments, Data/CARDS lines, Date/Time constant, Defined keyword, Formats/Informats, Macro definition, Macro keyword
Foreground: Green
Background: Default
Font style: Normal
Underlined
Preview

```
1 /* ----- some sample code ----- */
2 %include %unquote(&open.gtestit&close);
3
4 filename xyz printer \"HP Laserjet IIISi\" altde
5 data tut1;
```

Display 7. Using Menu Actions to Instantiate the Enhanced Editor Options Window

Now that we have used the menu action to obtain access to changing colors and fonts, we can submit the command PREVCMD (or equivalently the command *?)* to find the command used to populate the window. Display 8 illustrates use of PREVCMD to find the command executed by the menu action; entering *PREVCMD* in the command line, the command EDCMD *customize* appears letting the user know it is this command that can be used to customize their program editor. Once the command is known, it can be issued from any of the four locations. Knowing the command also allows users to enter parameters next to the command which allow commands to be more flexible and efficient than menu actions alone.



```
Command ===> PREVCMD
```

```
Command ===> EDCMD customize
```

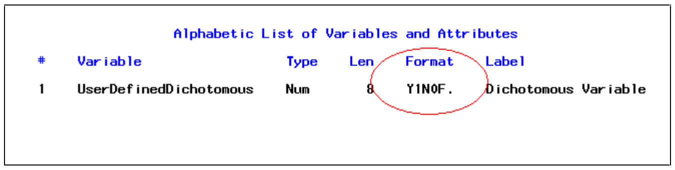
Display 8. Using PREVCMD to find the command executed from a menu action

CREATING CUSTOMIZED COMMANDS USING THE MACRO FACILITY

Accomplishing a task where a command is not already available can prove challenging. Luckily, the macro facility offers an environment where customized commands (called command-style macros) can be created. Command-style macros are macros that are created using the CMD option and are designed to be executed from any of the four command acceptable regions (because they are designed to appear like commands, they do not need to be preceded by a % symbol). However, users must be willing to trade efficiency for this handy tool, as additional system resources are used to process these types of macros. Developing a command-style macro starts with an initial idea (as all macro's do), but must be written with one major concept in mind – the code within the macro must be command line executable. This means that only macro language, SAS code wrapped in a GSUBMIT command, and existing SAS commands are acceptable. The last section of this paper is divided into three sections: motivating an idea for a customized command, writing a macro to carry out the customized task, and executing the command-style macro.

INITIAL IDEA

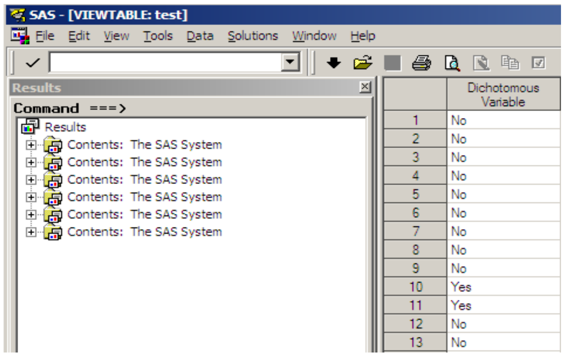
Inherited data sets can sometimes pose a problem if there are existing user-defined formats. These formats can hide the true value of a variable making is hard to subset the data. Output 9 illustrates this problem by providing the PROC CONTENTS output from a data set that contains a user-defined format (Y1NOF).



```
                 Alphabetic List of Variables and Attributes

 #    Variable                  Type    Len    Format    Label

 1    UserDefinedDichotomous    Num       8    Y1NOF.    Dichotomous Variable
```

Output 1. PROC CONTENTS including a user-defined format.

The format Y1N0F is used to "dress up" the values in the variable UserDefinedDichotomous. This is generally a very useful task when presenting results; however, when querying the data, these formats can become a nuisance. For example, in Display 9, the variable's formatted values are "yes" and "no". This format can be informative for presenting results. However, if one were to query this data, what value could be used to subset the data to only “yes”? From the PROC CONTENTS statement above (see Output 1), it is evident that a subset condition of “yes” will not work, because this variable is numeric. This information is not readily available from the output of the VIEWTABLE command. A customized command could be very useful in this situation.



Results
- Contents: The SAS System
- Contents: The SAS System
- Contents: The SAS System
- Contents: The SAS System
- Contents: The SAS System
- Contents: The SAS System

	Dichotomous Variable
1	No
2	No
3	No
4	No
5	No
6	No
7	No
8	No
9	No
10	Yes
11	Yes
12	No
13	No

Display 9. VIEWTABLE window with a data set that contains a user-defined format.

WRITING A COMMAND-STYLE MACRO

All macros begin with the %MACRO statement, but command-style macros must also contain the CMD option. The CMD option is designed to be used in macros that will be executed from the command-line.

```
%MACRO  macro-name /cmd;
```

In order to solve the mystery of a variables true value, it would be useful to obtain information for a particular user-defined format. This can be accomplished in various ways, one of which is to use PROC SQL to read the meta-data from dictionary.formats.

```
proc sql;
select  count(*), fmtname into :ttl, :list separated by ' '
from dictionary.formats
where index(fmtname,’Y1N0F’)>=1
;
quit;
```

However, the code above will not work as a command-style macro because it cannot be executed from the command line. Remember, there are three options when creating command-style macros: (1) find a way to use only macro language to obtain the necessary information (e.g. %LET, %SYSFUNC(), etc.), (2) use existing SAS commands and modify them in some way using macro language or GSUBMIT wrapped code, or (3) wrap SAS code in a GSUBMIT command. Option (3) is shown below:

```
gsubmit "
proc sql noprint;
select  count(*), fmtname into :ttl, :list separated by ' '
from dictionary.formats
where index(fmtname,%str(%")%upcase(&format.)%str(%"))>=1
;
quit;
"
;
```

Notice that the code above has been altered so that the format is now a macro variable (&format.). This is done so that positional parameters can be used later when the code becomes more developed. It is important to note here that the three methods for creating a command-style macro mentioned above do not have to be exclusive; that is, a combination can be used. One example of macro language and GSUBMIT wrapped code working in collaboration can be seen in the code below:

```
%do i=1 %to &ttl.;
       %global libnms&i type&i passon;
       gsubmit "
       proc sql noprint;
       select libname, objtype into :libnms&i., :type&i.
       from dictionary.catalogs
       where memname='FORMATS' and
       objname=%str(%")%upcase(%scan(&list,&i.))%str(%")
       ;
       quit;
       "
```

```
                        ;
                        %if &&type&i.=FORMATC %then %do;
                               %let passon=$%scan(&list,&i.);
                        %end;
                        %if &&type&i.^=FORMATC %then %do;
                               %let passon=%scan(&list,&i.);
                        %end;
                        gsubmit "
                        proc format library=&&libnms&i...formats cntlout=%scan(&list,&i.);
                        select &passon.;
                        run;
                        "
                        ;
                        gsubmit "
                        title1 %str(%")%scan(&list,&i.) Format Display%str(%");
                        proc print data=%scan(&list,&i.)(keep=start end label) noobs ;
                        run;
                        "
                        ;
                 %end;
```

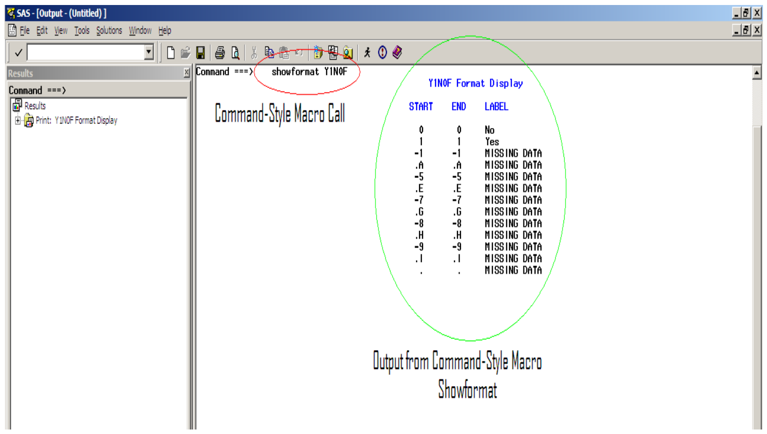
Once the logic portion of the macro is complete, the macro should be closed with a %MEND statement. When the %MEND statement is tacked on, the command-style macro is completed and is ready to be saved. This entire command-style macro can be viewed at the end of this document and hereafter will be referred to as SHOWFORMAT. In summary, a command-style macro is different from a name-style macro in that the CMD option is used, the code within the macro is designed to be submitted from the command line, and the macro can be executed without a preceding percent sign.

EXECUTING THE COMMAND-STYLE MACRO ON THE COMMAND LINE

By default, SAS is not set up to look for command-style macros, so the user must reconfigure this option by submitting the following statement:

```
options cmdmac;
```

If users plan to use a lot of command-style macros, it is suggested that this statement is included in an autoexec file. This option specifies that the macro processor examine the first word of every windowing environment command to see whether it is a command-style macro (NOTE: this will reduce efficiency). Next, the command-style macro must be compiled. Once these steps have been completed, users are able to call their command-style macro from any area on the SAS user interface that accepts SAS commands. Display 10 is an example of submitting the SHOWFORMAT command-style macro from the command line.


SAS - [Output - (Untitled)]
File Edit View Tools Solutions Window Help
Results
Command ===>
Results
Print: Y1N0F Format Display
Command-Style Macro Call
Output from Command-Style Macro Showformat

```
Command ===>   showformat Y1N0F

              Y1N0F Format Display

          START     END     LABEL

              0       0     No
              1       1     Yes
             -1      -1     MISSING DATA
             .A      .A     MISSING DATA
             -5      -5     MISSING DATA
             .E      .E     MISSING DATA
             -7      -7     MISSING DATA
             .G      .G     MISSING DATA
             -8      -8     MISSING DATA
             .H      .H     MISSING DATA
             -9      -9     MISSING DATA
             .I      .I     MISSING DATA
              .       .     MISSING DATA
```

Display 10: Command-Style Macro Call and Output from SHOWFORMAT.

The output from this macro allows us to accomplish our original task – subset the aforementioned data to only “yes”. We now know that this should be accomplished by setting the variable UserDefinedDichotomous equal to 1.

CONCLUSION

SAS commands allow programmers to become more efficient at navigating, customizing, and manipulating a SAS session. If command documentation is hard to find, utilizing the power of the PREVCMD/? command allows users to find the desired command through menu actions. The ability to create command-style macros and execute them as any other SAS command, limits programmers only to their imaginations. Hopefully this paper will serve as a catalyst for other SAS users to begin creating innovative command-style macros that will allow programmers and analysts to become more efficient.

REFERENCES

- Carpenter, Art. 2004. *Carpenter's Complete Guide to the SAS® Macro Language, Second Edition.* Cary, NC: SAS Institute Inc.

- Fairfeld-Carter, Brian and Hunt, Stephen. "*A Hacker's Guide to SAS® Environment Customization*" SAS Users Global International 30 Proceedings. April 2005.

ACKNOWLEDGMENTS

The author wishes to extend special thanks to Lance Couzens, Matthew Nelson, and Franklin Sell for their help, support, and willingness to share ideas and solutions to interesting problems.

CONTACT INFORMATION

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

Nigel Benjamin Neely
RTI International
3040 Cornwallis Road
Research Triangle Park, NC 27709-7417
Work Phone: (919) 541-7417
E-mail: bneely@rti.org

SOURCE CODE

```
%macro showformat(input) /cmd;
       gsubmit "
       proc sql noprint;
              select  count(*), fmtname into :ttl, :list separated by ' '
              from dictionary.formats
              where index(fmtname,%str(%")%upcase(&input.)%str(%"))>=1
              ;
       quit;
       "
       ;

       %do i=1 %to &ttl.;
       %global libnms&i type&i passon;
              gsubmit "
              proc sql noprint;
                     select libname, objtype into :libnms&i., :type&i.
                     from dictionary.catalogs
                     where memname='FORMATS' and
                     objname=%str(%")%upcase(%scan(&list,&i.))%str(%")
                     ;
              quit;
              "
              ;
              %if &&type&i.=FORMATC %then %do;
                     %let passon=$%scan(&list,&i.);
              %end;
              %if &&type&i.^=FORMATC %then %do;
                     %let passon=%scan(&list,&i.);
              %end;
              gsubmit "
              proc format library=&&libnms&i...formats cntlout=%scan(&list,&i.);
                     select &passon.;
              run;
              "
              ;
              gsubmit "
              title1 %str(%")%scan(&list,&i.) Format Display%str(%");
              proc print data=%scan(&list,&i.)(keep=start end label) noobs ; run;
              "
              ;
       %end;
%mend;
```