PharmaSUG 2020 - Paper AP-264

Hidden Gems in SAS Display Manager: Old Wine in New Bottle

Ramki Muthu, Veristat LLC, Southborough, MA 01772 USA

ABSTRACT:

Clinical SAS programmers tend to spend more than 90% of their work time using the SAS enhanced editor and other display manager windows, and it is imperative to maximize the available functionalities for their routine task. While most of the programmers are already familiar handling the SAS editor, and the topic was reviewed earlier, this paper attempts to hunt some hidden (or under-utilized) features in SAS display manager for Windows. Briefly, starting with few common etiquettes this paper discuses: Summary of keyboard shortcuts that programmers should be aware, Window management to make a split view of a program and utilizing tile vertical options, Using command bars to display required variables from a dataset, sub setting the records and precisely moving to the desired record (e.g. from 1st to 28474th record) using the command prompt. With few examples, it also discusses how we could utilize keyboard macros, add tool icons for common tasks, and suggest how to learn new (or unfamiliar) SAS procedures.

INTRODUCTION:

SAS is the proprietary software that has been extensively used in Clinical SAS programing to report clinical trial data to FDA and other regulatory agencies. The SAS windowing environment is built with BASE SAS core and other components on top of it to access, analyze and present data for various business solutions. The entire system is built with excellent graphical user interface with different sub-windows such as enhanced editor, log, view table and result window for the user to interact with SAS system. Even though every programmer is familiar with each window, it also has many hidden gems that are not commonly visible to many programmers. Surprisingly with authors knowledge, the applications of display manager are not highlighted in any of the SAS trainings as well. The details for each window and its usage are integrated within the SAS help documentation, which can be easily reached by hitting F1 key (or type help in command line) when any of the specific window is active. Try pressing F1 key from an *unedited line* within the SAS editor that directs to SAS help documentation in "Using the Enhanced Editor". Additionally, we can also get additional help in editor window by doing the same for many SAS keywords: such as name of the functions (e.g. place the cursor before the name of the function and hit F1), name of the procedure (e.g. place cursor in front of sort for PROC SORT procedure and hit F1), any keywords which appear in light blue color (e.g. RETAIN, MERGE, ARRAY, DO, etc.) takes directly to the specific help location which is readily available than searching the help documentation using key words. However, not many of us are aware of how to navigate to the specific help documentation other than making a quick search. Most of us stick to what we know and don't have time to explore new features. Carpenter 2012 has comprehensively explored this topic first [1] and followed by others. This paper reiterates some of the concepts along with few new discussions on how and where we could use these tips in our clinical SAS programing tasks. The paper is arranged in three sections namely: 1. Getting into different windows, 2. User defined keyboard abbreviations for SAS programing, 3. Adding user defined task-based icons. The author assumes that the reader of this paper has a basic/intermediate Clinical SAS programing experience and recommends *practicing* the tips with SAS rather than binge reading the entire paper. The symbol * is used to indicate that it is a new discussion or could be a frequently used tip per authors knowledge.

1. GETTING INTO DIFFERENT WINDOWS

1.1 View Table: Open any SAS dataset to activate the view table window and then hit F1 key (or type help in command line), which navigates to the help documentation on how to use the view table window. One of the most common tasks that we do with the view table is to show only selected variables for that we use Hide/Unhide column to pick the variables to show/hide. If the main view table is in maximized view, then

the resulting Hide/Unhide sub window pops up in maximized window showing only few variables in the dataset. With a click of Restore Down from top right-hand side corner of the window (or window key + down arrow) we can view list of more variables (Fig 1A). Also, when we try to rearrange the order, for example as show in Fig. 1B if the intention is to bring BRTHDT above AGE, instead of mouse clicking the up arrow key multiple times, I find it useful to single click the up arrow key and quickly move the variable with keyboard Enter key to move the variable to the desired location.

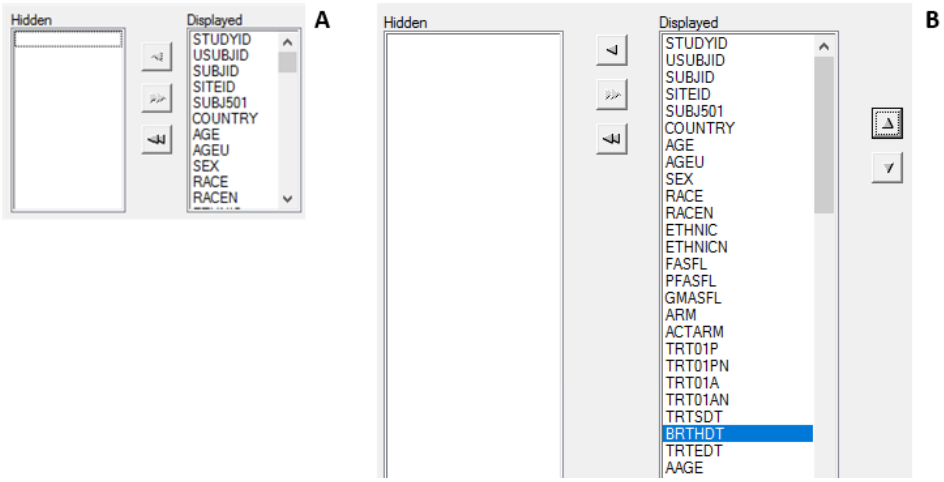

Fig 1. Show/Unhide variables from view table. **A.** Shows the maximized view showing only few variables, **B.** shows the restored view to see more variables.

The command line offers more flexibilities to look at the view tables using multiple commands which is not widely used (at least with the new programmers). By default, SAS has a shorter command line, which I personally prefer to keep it wider. Table 1A shows some examples and a brief summary of 10 common tasks that are frequently used.

#	Task	Command examples	Comments
0.*	Help for different task	help help proc report help sgplot	Help command takes to the relevant SAS help documentation
1.**	View dataset from work library	**vt** adqs **viewt** adqs **viewtable** adqs	vt stands for view table, open adqs from work library (use different libname as necessary)
2.**	Show selected variables	**column** 'usubjid parcat1 paramn param paramcd' **column** 'usubjid parcat1 paramn param paramcd'**;** **columns** "usubjid parcat1 paramn param paramcd" **keep** " usubjid parcat1 paramn param paramcd"; **hide** "paramn"; **unhide** "paramn";	column/ columns/ keep commands are used with a single or double quotation
3.**	Subset a dataset	**where** usubjid 'ABC-XYZ-001-0100' and paramn=1 and avisitn=10 and anl01fl='Y';	Subset from the previous view

…continued

4.	Bring all variables back	**column** _all_	
5. *	View attributes of adqs	**vars** adqs	see below for a specific variable attribute
6.***	Move the view table to 32314 (n^{th}) row of a dataset	**forward** 32313	Moves directly to 32314^{th} row
7.	Move the view table 10 more rows	**forward** 10	Moves from 32314 to 32324
8.	Move the view table 20 rows back	**backward** 20	Moves from 32324 to 32304
9.	Find value | String	**find** ABC-XYZ-001-0100	highlights the find. Rfind command searches for repeat finding
10.*	Form view (like annotated CRF)	**vt** adsl **view=form**	Sometimes based on the data (e.g. subject level information) form view appear better than table view

Table 1A. Illustrates 10 common tasks that can be performed using command line.

The command line only executes the command and it does not show any error or warning note in the log. Hence, we need to type the commands without any spelling mistakes. The commands are not case sensitive, works with or without semicolon, and works with either matching single or double quotations (don't use both) as per the required syntax. Copying apostrophe (') from word, power point won't work and make sure to use ' and not " or '. Table 1B has a brief summary of the pros and cons of using Hide/Unhide column approach *vs* command line. Users have the option of choosing either method based on their preference and task. However, the ability to reuse the command code from the pull-down menu stands out over the Show/Unhide approach especially when working on any derivations that I need to look before and after running the program.

#	Show/Unhide	Command Line
1.	Open a dataset in table view and press Alt + t + h to reach Show/unhide screen or use a right click and select 'where' to subset data.	Access from editor by F11 (if this does not work check the key definition and assign Command Focus to F11 key) and enter command. **
2.	Need to use mouse*	Need to type the command
3.	Easier to use and have access to list of variables	Need to remember the commands and know the name of the variables
4.	Once the window is closed, it is gone and need to redo	Using the command line pull down menu we can *reuse* with a single click and modify as needed ***
5.	If we have a where clause, cannot copy paste	We can copy paste directly to the command line and use where clause functions as needed ***
6.	Dealing with dates (e.g. ADT in ISO format), formatted date values are seen and easier to pick	Need to enter the numeric SAS date value. To find the numeric value, open the column attributes (e.g. ADT) with right click and apply best. format.

Table 1B. Pros and Cons of using Show/Unhide *versus* Command line approach.

The form view of a table is almost never used by us (Table 1A #10, Fig 2.), but I believe this provide better insights to view subject level information's (recall scrolling back and forth of a broad ADSL dataset). Interestingly, the form view assigns the width of each variable based on the attribute length of a variable. E.g. in the Fig. 2. When comparing the form length of RACE, TRT01P ($200) and SEX ($1) we can potentially identify the need of longer length is necessary or not. Reducing the unnecessary variable length could potentially save some storage space, especially when carrying ADSL variables to other datasets. In addition, the form view also provide similar feel of looking at annotated CRF forms used in clinical SAS programming. There are few other commands such us hold**, release**, bottom, commands are also useful and worth exploring.

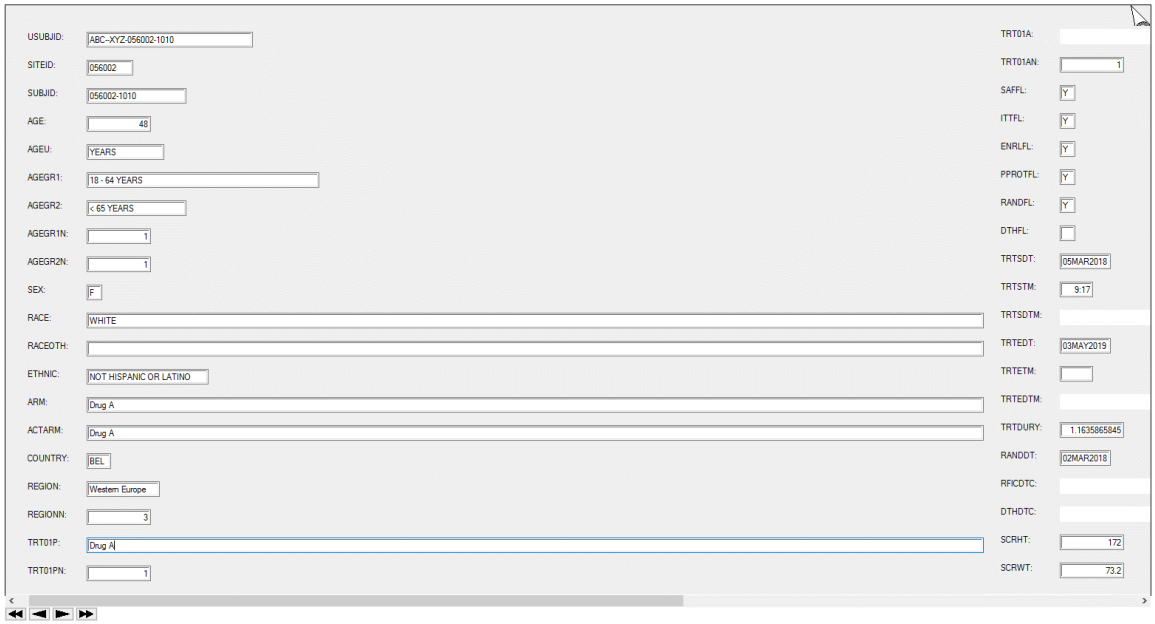

Fig. 2. A representative example of a form view. See how easily readable but focus on how easily to read the order of the variable and access the trailing blanks. E.g. If Race, ARM, ACTARM could assign a smaller length at SDTM level, would save lot of storage place across other SDTM datasets and ADaM datasets*.

1.2. Enhanced Editor Window: Though primarily this window is used to edit SAS programs, there are numerous tools and customizations are available for the user to interact with SAS. As shown in the introduction, practice to use the F1 key to get specific help from SAS documentation and here we focus on the features that are not widely used.

1.2.1. Common keyboard short-cuts that everyone should use: Hemedinger's old blog summarizes some of the most common shortcuts that everyone should be using [2]. SAS programmers needs to be aware and comfortable in using these short cuts to change case, commenting out the code, shuffling between the matching parenthesis (Ctrl + [, Ctrl +]) and matching DO/END (Alt + [, Alt +]) keywords. In addition to the common shortcuts, it is worth considering few more as discussed below. We often work on lengthy SAS codes and make updates at different places. I find it useful to create a bookmark within the editor to easily identify the specific line where we make the update. In addition, we could also toggle between the next or previous bookmarks within the program (e.g. we can toggle with a key stroke from line 234 to line 9 and vice versa). However, note the bookmarks will be lost once we close the program.

Action	Shortcut
Bookmark a line	Ctrl + F2 on an unmarked line
Unmark a line	Ctrl + F2 on a marked line
Go to next bookmark	F2
Go to the previous bookmark	Shift + F2

```
24 proc copy in=ads out=work;
25      select adsl adlb;
26 run;
```

Fig 3. Short cuts to create bookmark and a sample code how a bookmark look within SAS Editor. *This tip is expected to be useful to shuffle between multiple lines quickly***.

***1.2.2. Split View and Window management*:** Additionally, when working on lengthy programs, I often find it useful to make a split view of the same program so that I can look at different portion of the program. For example, if we want to see line number 25 and 269 of a program in the same screen without any cluttering (See Fig. 4) that can be accomplished by creating a new window of the same program and comparing side by side. 1. Pick a sample SAS program 2. Under Window menu click New Window. 3. If multiple windows are open minimize all and then select only the windows you want to see. 4. Under Windows menu click Tile Vertically or Tile Horizontally as you prefer. Fig 2 shows an example of how a horizontal comparison looks. Any changes you make on either window gets updated in the same program and both windows have the same file name with a numeric index at the end. The tile option not only be used for same program, we can compare different windows and different programs as needed**.

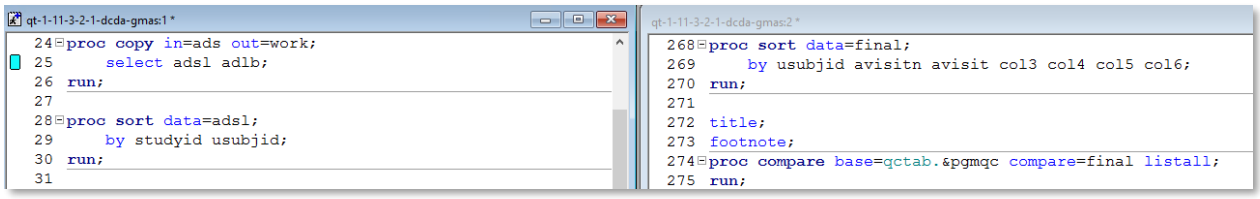

Fig 4. Split view of a sample program comparing side by side horizontally. This tip is expected to view different parts of same program**.

***1.2.3. Vertical Selection*:** In the editor the copy-paste functionalities works horizontally, meaning copying and pasting editor text by a word or lines of SAS code, there are circumstances where we need to process the text vertically. For example, when working with formats or DO loops, updating variable meta-data information's using RETAIN statement, or copying texts from Excel we make similar formatting update with a large or small chunk of code. Consider we are working on SDTM questionnaire datasets and we have many QSTESTCDs and if we need to create a format, we could make a PROC PRINT of a distinct values of QSTESTCD. Copying and pasting the list from the output window to the editor end up as in Fig 5.❶. The unwanted text can be easily selected by pressing the Alt key and selecting the text using the mouse ❷. The selected text can be copied or deleted as per the need. We can also use tab to make uniform indentation. However, unlike some advanced text editors (e.g. Notepad++, Ultraedit), SAS enhanced editor still has some limitations in creating multiple line edits (e.g. editing all the selected text to 'ABCD' or making numeric increments like ABCD1, ABCD2 and so on).

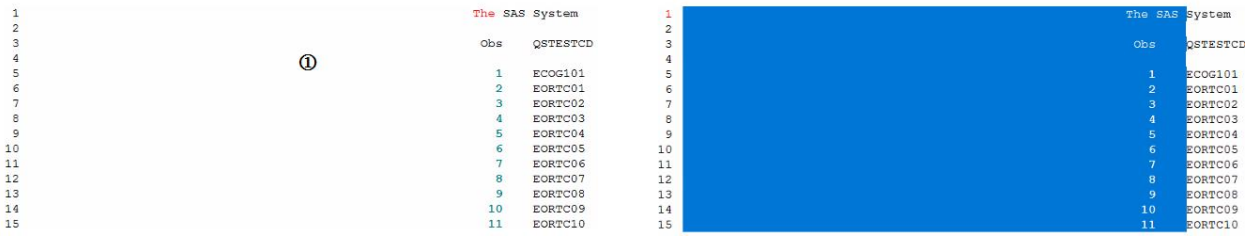

Fig 5. Vertical selection of texts to make a copy-paste or delete. In the left-hand side, the unwanted text can be selected by pressing Alt key and selecting the text using the mouse*.

2. USER DEFINED KEYBOARD ABBREVIATIONS:

The enhanced editor offers many built in keyboard macros (press t + m + u) under different category. The user defined category allows user to store different SAS code snippets with an abbreviation for each code snippet. So that when typing the abbreviation within the editor pulls the SAS code specific to the abbreviation. The abbreviation can be added with Ctrl + Shift + a (remember '**a**' for **a**dd code snippet) and to see the list of abbreviations by Ctrl + Shift + m (remember '**m**' for **m**acro code snippet list). This approach has a wide range of applicability based on the user namely, 1. It allows users to generate repetitive code with a keystroke, 2. Improves programming efficiency and avoid typos, 3. Allows users to learn and try different code. However, a word of caution is not to use too much of this technique, which may lead the user to so dependent on this.

Building list of abbreviations are completely subjective to the user, and a user may tend to add more abbreviations. Earlier, Carpenter 2012 [1] had shown how to use this technique and later few others had shown a list of suggestions [2-6]. My perspective on using this tool is restricted to the following criteria. Develop an abbreviation if it is: 1. Frequently used on daily basis, 2. Unique code that is not frequently used, but has some potential advantage.

2.1. Abbreviation based on frequent usage:

***1. Frequency Count:** Counting has been one of the most frequent tasks that we perform frequently, and the following abbreviation is a workhorse for those tasks.

```
/*dm 'next vt:&syslast; end;';*/ ❸
proc summary data=data_in nway missing;
      class var1 var2; ❶
      output out=data_out (drop=_type_ rename=(_freq_ = count));
run;
dm "vt &syslast" continue; ❷
```

Using the CLASS statement ❶ subgroup analysis can be performed with one or more variables (var1 var2 etc.) without the need of prior sorting. Moreover, the NWAY option allows us to get the summary of highest order of interaction (_TYPE_=0) and unique counts. By storing this piece of code within an abbreviation (e.g. as 'count') we can recall this code wherever we need to perform similar tasks. We only need to update the name of the dataset and variables names. We can utilize the dm (display manager) statement and &syslast macro variable which stores the name of the last created dataset to open ❷ or if it is open, then close ❸, summarize and then open. We can also use this DM statements as a separate abbreviation, to open the last created dataset. The SUMMARY procedure offers many more options, which we may forget over the time if we don't use it. For e.g., DESCENDING (to get the counts with descending order), GROUPINTERNAL (we can apply format to the classification variable), MLF (apply multi label format in the summary), ORDER (by freq or format) can be used (see Carpenter, 2010 [7]). Storing a brief summary of this note as another abbreviation (e.g. count1) can be of greater help for the programmers.

***2. QC count:** During the validation of production outputs, we frequently end up differences in number of records and the following abbreviation may be used. The user needs to identify the matching key variables that can go to the class statement ❶ and then update the name of the datasets.

```
***Production;
proc summary data=data_prod❷ nway missing;
  class ❶;
  output out=prod (drop=_type_ rename=(_freq_ = pcount));
run;

***QC;
```

```
proc summary data=data_qcd❷ nway missing;
  class ❶;
  output out=qc (drop=_type_ rename=(_freq_ = qcount));
run;

data diff;
  merge prod qc;
  by ;
  if pcount ne qcount;
 run;
 dm "vt &syslast" continue;
```

3. Ruler: The following can be used as an editor, e.g. when we want to restrict variable descriptive names (labels) to 40 characters. The enhanced editor offers other alternative to identify the line number and column location in the lower right corner (LN x, Col x), however having a ruler as below makes a visual boundary.

```
          1         2         3         4
1234567891123456789212345678931234567894
Rand. Admin of Neoadjuvant Therapy (N)
Primary stratification Best Response
```

2.2. Unique Abbreviation:

***1. Macro Resolution*:** Sometimes when working with large SAS macros and if we encounter any issues, it is difficult to precisely locate the location from the log output window. The line and column number that shows in the log window is not useful. For situations like that for debugging purpose it is better to get the SAS code generated by SAS macro processor and the following code that uses the MPRINT and MFILE options can be used for that purpose (remember to use F1 for more details). The output from macro processor can be saved either in an unformatted SAS or TXT file. The other alternative using OPTIONS NOSYMBOLGEN NOMPRINT NOMLOGIC and execute the macro to see the resolved code in the log window (remember to turn off to the default OPTIONS).

```
***Get the resolved SAS Code;
filename mprint 'Enter the full path name here, followed by file name.
eg. Z:\Z:\xxx\...\check.sas';
*it can output either as text or sas file;
options mprint mfile;
* execute your macro eg. %npct(datain=adlb, denom=denom3, ....);
filename mprint clear;
options nomprint nomfile;
```

***2. Convert dataset to CSV file*:** We rarely export SAS dataset to CSV files, but that can be accomplished by ODS statement and PROC REPORT. However, there is an in-built SAS macro available for that purpose that may be useful (Frey, G. 2005). In the below code the CARS dataset from SASHELP library can be exported to Z:\ location.

```
%ds2csv(data=sashelp.cars, runmode=b, csvfile=Z:\cars.csv);
```

****3. Distinct levels of a classification variable*:** It is not a rare task, but often we calculate distinct values using PROC SQL or other methods. Interestingly, we can also accomplish this with PROC FREQ using the NLEVELS option (SAS Note 30867). In the code below, calculate the distinct levels of all ADSL (Subject

level dataset used in clinical trial applications) variables. The output contains the distinct values (NLEVELS) for each variable as well number of missing (NMISSLEVELS) and non-missing values (NNONMISSLEVELS). We often don't need the levels for all variables, so to summarize levels for few variables, _all_ in the table statement ❶ can be replaced with the list of variables (e.g. USUBJID AGE).

```
***Check levels for all variables;
ods output nlevels=levels;
proc freq data=ads.adsl nlevels ;
 tables _all_; ❶
run;
```

***2.3. Macro recording*:** In addition to the user defined keyboard abbreviations, using different commands available (press t + m + u) we can also build a macro that executes sequential editor activities that we intend to do. Carpenter 2012 shown how to alphabetically sort selected lines in the editor. You can try this by selecting a list of variables or any text in the editor and execute a command '*Sort the selected lines*' from the Run Macro commands (press t + m + u). I find this useful in the places where I have a long list of chemotherapy or other names with the IN operator, to sort alphabetically and to identify any duplicates. Similarly, Eslinger 2018 has shown another macro in her SAS blog to select a whole block of code from the PROC statement down to the RUN statement.

****1. Vertical to Horizontal*:** In addition to the commands, the macro recording also records various editing activities (e.g. type, delete, move the cursor up or down etc.). Assume we have a long list of variables in the editor in vertical order and we like to bring the variables horizontally and separate each variable with a single space (or separated by a comma for SQL applications). We can do this task once and save those activities as a macro for future use. To keep it simple, let's assume we have a 3-4 variable names in the editor, then follow the below sequence. 1. Go the Keyboard Macros and click Record New Macro. 2. We need to make sure to remove any trailing space. Select all your variables of interest and then Run Macro command (press t + m + u) '***R**emove trailing whitespace*'. If the category is All then keep pressing **R** until you reach the specific command of your interest which is much quicker than searching each category. 3. Manually edit the variables in the order you want. Meaning, go to the end of the first variable, give a single space (or command and space) and then press delete to bring the second variable and so on. 4. Stop recording the macro and save with a name (v2h) and description (Convert Vertical to Horizontal order). In the future, we can accomplish this task by running this keyboard macro. Once you are familiar with this you can save this macro by editing with more variables so that the macro can handle few or many variables at once.

3. ADDING USER DEFINED TASK BASED ICONS

It is possible to add different icons and make different customization to the tool bar and direct SAS to perform specific tasks [1]. Briefly, right click the tool bar, then from customize tab pick an icon and add a command of interest. The command executes either a DM (display manager) statement or GSUBMIT command. The GSUBMIT commands are stored in the Windows clipboard and is used to submit the SAS statements. Carpenter 2012 had shown how to create an icon to clear all the temporary datasets from the work library by executing a PROC DATASET procedure using GSUBMIT command. Here I extend the discussion with few more examples, which I believe is relevant for our programming activities. Please note different icons can be added to each sub window namely editor, view table, log or result window.

#	Sub Window	Task with a click	Command
1.***	Editor	Submit and focus the log	clear log; clear output; submit; log;
2. **	Editor	Open last created dataset	gsubmit "dm 'viewtable &syslast'; "
3.	Editor	PROC PRIINT last executed dataset	gsubmit "proc print data=&syslast; run;"
4. *	Editor (and/or) View Table	Close 10 or less opened VIEW TABLE	gsubmit "%macro closevt; %do i=1 %to 10; dm 'next viewtable:; end;'; %end; %mend; %closevt"
5.	Editor	Clear Title Footnote	gsubmit "title; footnote;"
6. *	Editor	Discussion notes, like sticky note	notepad
7. **	Log	Check log error	FIND "ERROR"
8. **	Log	Check log warning	FIND "WARNING"
9.	View Table	Scroll left	left;
10.	View Table	Scroll right	right;
11.	View Table	Scroll Up	up;
12.	View Table	Scroll down	down;

Table 2. Examples of adding few user defined task-based icons to the tool bar.

The default running man icon submits the SAS program and it is always expected the user to open the log window to watch for any issues, sometimes we forget to do this. Marisol 2013 suggested to add a DM key (not discussed here but encouraged to explore) command for this purpose. Similar task can also be performed by adding a tool icon to the tool bar (#1 of Table 2) which forces users to see the SAS log every time after the submission. Many companies use different log checker program to check any potential log issues, by scanning the log txt files for specific keywords. However, those log files are created by batch submission and we need to go outside of SAS window every time. Using the FIND command, we can quickly scan the log window to check for few important keywords (See # 7, #8 Table 2). However, note the FIND command won't accept more than one key word and also if you are at the end of the log window it does not go to the top by default, we need scroll to the top of the window to start finding those key words in the log window. The other commands such as left, right, up, down (see #9 to #12) may be used to explore the view table. Please note the left scrolling using the left command did not work as expected in SAS 9.4. Additionally, we can also explore the GSUBMIT command from a buffer to submit the SAS statements that are enclosed in quotation. In Table 2. Number of different examples are given that used GSUBMIT.

In #2 of Table 2, used a DM statement to view the last created SAS data with a single click without fiddle through the file explorer. In #4, a series of DM commands wrapped with a small macro can also be executed to close many opened view tables with a single click. Similarly, a SAS procedure (#3) or some global SAS commands to clear the title, footnote (#5) or altering the default options can be added.

Unfortunately, the GSUBMIT command only allows up to 512 characters, so if we need to include longer code, we cannot use GSUBMIT, but we can explore %INCLUDE statement. For discussion purpose let us take a sample SAS code that was shown above for frequency count (from section 1.2.4.1) and build an icon for similar task. The code is modified to enhance our user experience (See Figure 6.). The %WINDOW ❶ and %DISPLAY❷ allows to pop-up a new window and ask the user to enter the required parameters.

Figure 6. The simple frequency count program is modified to enhance the user experience (overlaid) using %WINDOW ❶and %DISPLAY ❷ statements***.

To place this task to the tool bar, we need to save the code in a defined location and then add an icon using the following command: submit " %include 'X:\My SAS Files\9.4\prompt.sas';". Where %include statement includes the PROMPT.SAS file from the defined location and submit command submits the program in the buffer. The main advantage of this approach over running the keyboard macro is we do not need to call and update the abbreviation, we only need to enter the required parameters and the code is submitted from the tool bar (of course we can see it in the log window) without moving to a different window. Unlike GSUBMIT there is no character limit for this approach we have more flexibility to explore for different tasks.

Lastly, the NOTEPAD window which is a simpler version of enhanced editor is almost not used by many of us. The simplicity adds few advantages and we can make use of it. Consider using NOTEPAD as a note taking tool. It allows us to type anywhere in the window, meaning if you want to type something at row 10 and column 20, we don't need to press enter 10 times and space bar 20 times rather we can move the mouse and start typing. Though we can save different NOTEPAD files, interestingly if we close the SAS window without closing the NOTEPAD window, SAS will bring the unsaved NOTEPAD next time we login to SAS. This allows us to take some random commands like a sticky note can be added during the team meeting or to develop complex logics. The command 'notepad' as in Table 2 Task #6 can be added to the tool bar.

CONCLUSION:

The display manger is a powerful tool that allows user to make lot of customized changes based on their programming requirements. Getting comfortable with the display manager tasks will improve productivity and give better insights to the programmers.

REFERENCES:

*Useful, ** Recommended, ***Frequently used

1. Carpenter, A. L. (2012). SAS Global Forum 2012. Doing More with the SAS® Display Manager: From Editor to ViewTable - Options and Tools You Should Know.
2. Hemidinger, C. (2013) SAS Blog. 5 keyboard shortcuts in SAS that will change your life https://blogs.sas.com/content/sasdummy/2013/10/29/five-keyboard-shortcuts/

3. Frey, G. (2005) MidWest SAS Users Group. SAS Excels! https://www.lexjansen.com/mwsug/2005/Information_Systems/IS200.pdf
4. SAS Usage Note 30867: Modernizing Your SAS Code: PROC FREQ Applications http://support.sas.com/kb/30/867.html
5. Eslinger, J (2018) Efficiency at your fingertips: Keyboard macros and function keys. https://blogs.sas.com/content/sgf/2018/02/16/efficiency-at-your-fingertips-keyboard-macros-and-function-keys/
6. Marisol 2013 South Central SAS User Group meeting. http://www.scsug.org/wp-content/uploads/2013/11/Shortcuts-to-Save-Time-While-Working-with-SAS%C2%AE-Marisol-Rivera-Barragan.pdf
7. Carpenter, A. L. (2010) SAS Global Forum 2010. The MEANS/SUMMARY Procedure: Doing More

ACKNOWLEDGMENTS:

The author like to thank Nirav Patel and Verienne Emile for their constant encouragement throughout the process. Any brand and product names are trademarks of their respective companies.

CONTACT INFORMATION:

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

Ramki Muthu
muthuramki@gmail.com
https://www.veristat.com/contact-us