PhUSE EU Connect 2024

Paper CT07

How to optimize your SAS code: Tips and Tricks

Deepak Ananthan, Zifo RnD Solutions, Chennai, India
Sreelekha Pasupuleti, Zifo RnD Solutions, Chennai, India
Vaisint Singh Muthudurai, Zifo RnD Solutions, Chennai, India

ABSTRACT

The SAS language serves as a widely utilized statistical analysis programming tool. This paper aims to highlight various helpful tips and tricks that can streamline your programming experience. This paper will explore a collection of functions and statements designed to enhance coding efficiency, minimize runtime, and facilitate easier error debugging. It will also discuss methods for programming that facilitates seamless updates in the future, along with certain macro functions aimed at reducing the manual workload involved in producing RTF output.

INTRODUCTION

Efficiency is crucial in data analysis, and SAS (Statistical Analysis System) is a powerful tool for managing complex data and performing intricate statistical analyses. However, the effectiveness of SAS code hinges on the techniques and practices used. Optimizing SAS code accelerates processing, improves accuracy, and enhances data analysis tasks. This paper focuses on refining SAS programming skills, emphasizing practical strategies to improve coding efficiency, reduce execution time, and simplify debugging. By adopting best practices and utilizing advanced features, programmers can streamline workflows, write cleaner code, and minimize resource consumption. Special attention is given to macro functions for automating repetitive tasks and generating outputs in formats like RTF (Rich Text Format) with less manual effort.

In this paper, we will explore a range of strategies for optimizing our code, focusing on aspects such as environment setup, general programming best practices, and techniques for enhancing quality and saving time.

ENVIRONMENTAL SETUP

1. Creating a directory

Creating a directory is a straightforward process, and we can accomplish this using the command x mkdir. However, it's important not to manually input the directory path, as it may include user-specific details, such as "S:\Name.Initial\Study_Name". Instead, we recommend retrieving the directory location dynamically with the %sysget function. The appropriate syntax for this is %sysget(sas_execfilepath);. This method ensures that you obtain the current path accurately without any user-specific information.

Input:

```
1  %let filepath = %sysget(sas_execfilepath);
2
3  %put &filepath;
```

Output:

```
13   %let filepath = %sysget(sas_execfilepath);
14
15   %put &filepath;
S:\Samyuktha.M\dm.sas
```

Creating a SAS code that consolidates all global macro information—such as the MedDRA version and SDTM version—enables us to manage these settings efficiently from a centralized location. By utilizing the %include statement to reference this program just once, we can seamlessly access all the macros defined within it. This method simplifies our workflow, enhances consistency, and makes it easier to manage updates and modifications across our codebase.

```
1  /* Concomitant Medication Coding Dictionary version number
2  %let CMDICT              = 2022;
3
4  /* Adverse events/Medical History Coding Dictionary version number
5  %let AEMHDICT            = 25.0;
```

```
1  %include "&progpath.\macrovar.sas";
```

2. Configuring settings in SAS library

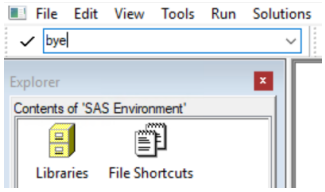
- **encoding = any;** - This parameter defines the encoding settings for the library, allowing for flexible handling of various character encodings
- **options fmtsearch;** - This option specifies the format search path for the library, ensuring that SAS can locate and utilize the appropriate formats for data processing.
- **access = readonly;** - This setting restricts access to the library, permitting only read operations and preventing any modifications to the data contained within.

3. Additional tips

- To clear the log and output windows, you can use the following code. This will help ensure that your workspace is clean and free of any previous messages or results, making it easier to focus on the current task at hand.

```
1  dm log "OUT;CLEAR;LOG;CLEAR;";
```

- To close the SAS window, you can simply type 'bye' in the search bar. This command will terminate your current SAS session and close the application, ensuring that all your work is saved, and the program is properly excited.

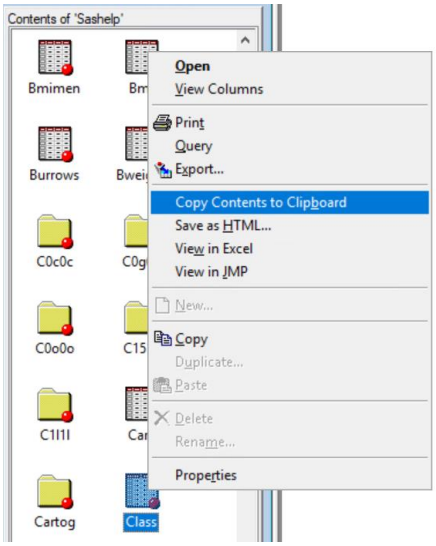


TIPS FOR PROGRAMMING

1. **Excel import:** You can import SAS datasets without needing to enter the complete file name. Occasionally, deviation files include dates in their names, but you can utilize wildcards, such as "?", to import Excel files without having to specify the entire name. This flexibility makes it easier to manage and access your data. Additionally, it enables effortless handling of new files without the need to alter existing code, ensuring you can integrate new data smoothly into your workflow while maintaining efficiency and consistency in your processes.

```
proc import datafile = "&SOURCEPATH.\ExternalData\&STUDYID._Randomization_&EXT.*?*.xls"
     dbms = xls out = &OUT. replace;
quit;
```

2. **Copying data from SAS to Excel:** Typically, you use the "Copy Contents to Clipboard" feature to transfer and view your SAS datasets in an Excel file. However, there are instances when the content fails to copy properly into Excel. To avoid this issue, you can execute the statement **ods select all;** before copying the SAS datasets. This ensures that all relevant output is properly prepared for copying, allowing for a seamless transfer to Excel.



```
21   filename _temp_ clipbrd;
22   ods noresults;
23   ods listing close;
24   ods html file=_temp_ rs=none style=minimal;
NOTE: Writing HTML Body file: _TEMP_
25   proc print data=Sashelp.'Class'N noobs;
26   run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds


27   ods html close;
28   ods results;
29   ods listing;
30   filename _temp_;
NOTE: Fileref _TEMP_ has been deassigned.
```

3. **Observation count:** The **sysnobs** option reports the number of records in the most recently created dataset. This information is particularly helpful for performing data condition checks, as it allows you to determine whether the dataset contains any observations. Additionally, you can use this feature to conditionally execute certain parts of a macro only if records are present, ensuring that your code runs efficiently and effectively based on the dataset's contents.

```
data check;
  set sashelp.class;
  if SEX eq "F";
run;

%let OBS = &sysnobs.;

%put &OBS.;
```

```
31   data check;
32     set sashelp.class;
33     if SEX eq "F";
34   run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.CHECK has 9 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


35
36   %let OBS = &sysnobs.;
37
38   %put &OBS.;
9
```

4. **Using Labels as headers when exporting SAS datasets to Excel:** When exporting SAS datasets to Excel, the default behaviour is for variable names to appear as headers. However, if you prefer to use labels as headers instead, you can achieve this by utilizing the label option in SAS. This function allows you to assign descriptive labels to your variables, which can then be used as headers in your exported Excel file.

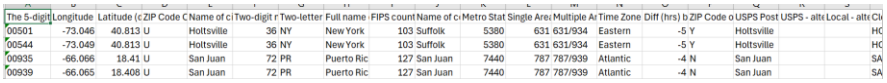
Exporting with headers as variable names:

```
proc export data = sashelp.xipcode(where=(ZIP_CLASS eq "U")) outfile = 'S:\Test\Code.csv' dbms = csv
replace;
run;
```

ZIP	X	Y	ZIP_CLASS	CITY	STATE	STATECOD	STATENAM	COUNTY	COUNTYN	MSA	AREACOD	AREACOD	TIMEZONE	GMTOFFSI	DST	PONAME	ALIAS_CIT	ALIAS_CIT
00501	-73.046	40.813	U	Holtsville		36	NY	New York	103	Suffolk	5380	631	631/934	Eastern	-5	Y	Holtsville		
00544	-73.049	40.813	U	Holtsville		36	NY	New York	103	Suffolk	5380	631	631/934	Eastern	-5	Y	Holtsville		
00935	-66.066	18.41	U	San Juan		72	PR	Puerto Ric	127	San Juan	7440	787	787/939	Atlantic	-4	N	San Juan		

Exporting with headers as variable labels:

```
proc export data = sashelp.xipcode(where=(ZIP_CLASS eq "U")) outfile = 'S:\Test\Code.csv' dbms = csv
label replace;
run;
```



The 5-digit	Longitude	Latitude (c	ZIP Code C	Name of ci	Two-digit r	Two-letter	Full name	FIPS count	Name of c	Metro Stat	Single Are	Multiple Ar	Time Zone	Diff (hrs) b	ZIP Code o	USPS Post	USPS - alte	Local - alte	Cl
00501	-73.046	40.813	U	Holtsville	36	NY	New York	103	Suffolk	5380	631	631/934	Eastern	-5	Y	Holtsville			H
00544	-73.049	40.813	U	Holtsville	36	NY	New York	103	Suffolk	5380	631	631/934	Eastern	-5	Y	Holtsville			H
00935	-66.066	18.41	U	San Juan	72	PR	Puerto Ric	127	San Juan	7440	787	787/939	Atlantic	-4	N	San Juan			SA
00939	-66.065	18.408	U	San Juan	72	PR	Puerto Ric	127	San Juan	7440	787	787/939	Atlantic	-4	N	San Juan			SA

5. **vvalue function:** When handling data, you can conveniently retrieve the display formats by using the **vvalue** function. This approach eliminates the need to manually search for the format of each variable, streamlining the process and ensuring you have direct access to the necessary display formats.

```
data dm;
  set dcrf.dm;
  _STUDYID = "&STUDYID.";
  _SUBJID = strip(SUBJECT);
  _SEX = strip(vvalue(GENDER));
run;
```

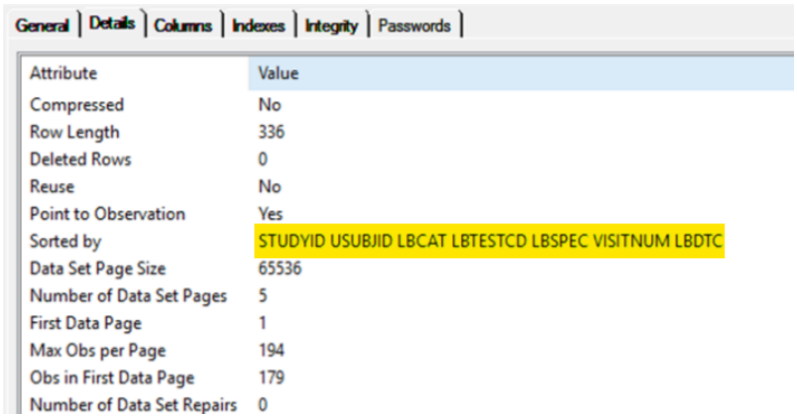
SUBJECT	GENDER
01-001	1
01-002	1
02-005	2

_SUBJID	_SEX
01-001	F
01-002	F
02-005	M

6. **Pre-sorted dataset:** If a SAS data set might already be sorted but hasn't been processed with a PROC SORT, you can use the PRESORTED option in PROC SORT. This option checks if the data set is already sorted by the variables specified in the BY statement and skips sorting if it is. Since this involves an additional check, use it only when you suspect the data set might be sorted. If the data set has already been sorted using PROC SORT, a note is written to the log: 'Input data set is already sorted, no sorting done'.

```
proc sort data = EXAMPLE presorted;
  by AGE;
run;
```

7. **Viewing Sort order:** Once a dataset is sorted using the key variables according to the standards, you can view the sort order in the properties (details) of the SAS dataset. This feature makes it easier for reviewers to verify the sort order. By checking the properties of the SAS dataset, as illustrated in the accompanying image, you can easily find and confirm the sort order.



8. **LIST option in PROC REPORT:** When working with PROC REPORT in SAS, specifying DEFINE statements for each variable in a dataset with many variables can be quite tedious and time-consuming. Each DEFINE statement is used to customize the appearance and behaviour of the report columns. However, there's a more efficient way to handle this using the LIST option. The LIST option in PROC REPORT allows you to automatically generate all the necessary DEFINE statements without having to write them manually. When you use the LIST option, PROC REPORT prints all the statements required to produce the output directly into the Log window. You can then copy these statements from the Log window to the Editor window and customize them as needed. This feature saves you a lot of effort and ensures that you don't miss any variables.

```
proc report data = ADSL LIST;
run;
```

```
38404
38405  proc report data = adsl LIST;
38406  run;

PROC REPORT DATA=WORK.ADSL LS=102 PS=56  SPLIT="/" CENTER ;
COLUMN  STUDYID SUBJID SITEID SCRNID BRTHDAT AGE AGEU AGEGR1 AGEGR1N SEX RACE RACEOTH;

DEFINE  STUDYID / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Study Identifier" ;
DEFINE  SUBJID / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT
"Subject Identifier for the Study" ;
DEFINE  SITEID / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Study Site Identifier" ;
DEFINE  SCRNID / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Screening Identifier" ;
DEFINE  BRTHDAT / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Date of Birth" ;
DEFINE  AGE / SUM FORMAT= BEST9. WIDTH=9     SPACING=2   RIGHT "Age" ;
DEFINE  AGEU / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Age Units" ;
DEFINE  AGEGR1 / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Pooled Age Group y" ;
DEFINE  AGEGR1N / SUM FORMAT= BEST9. WIDTH=9     SPACING=2   RIGHT "Pooled Age Group y (N)" ;
DEFINE  SEX / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Sex" ;
DEFINE  RACE / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Race" ;
DEFINE  RACEOTH / DISPLAY FORMAT= $200. WIDTH=200   SPACING=2   LEFT "Race Other" ;
RUN;

NOTE: There were 1168 observations read from the data set WORK.ADSL.
NOTE: PROCEDURE REPORT used (Total process time):
      real time           0.48 seconds
      cpu time            0.46 seconds
```

9. **Indentation in RTF:** Typically, we use the notation **^{Super }^_^_** for the first level of indentation and **^{Super }^_^_^_^_** for the second level of indentation. However, this method can be cumbersome and less efficient.

```
Type: n (%)
  Category: n (%)
    Subcategory1: n (%)
      Subcategory2: n (%)
-------------------------------------------------------------

Any relevant protocol deviation

Any relevant eligibility deviation

  Randomized or treated with study drug under > 1 subject
identifier
  Migraine history issue
    Experienced migraines for < 1 year prior to screening
    ≥ 9 moderate or severe migraines per month in the 3
months prior to screening
  Cardiovascular disease risk factor
    Ischemic coronary artery disease
    Other significant underlying cardiovascular disease
```

Instead, we can simplify the process by using the RTF (Rich Text Format) commands. For the first level of indentation, we use **(^R/RTF'\li220')**, and for the second level of indentation, we use **(^R/RTF'\li440')**. These commands automatically set the indentation levels, making the formatting process more straightforward and less time-consuming. By using these RTF commands, we can achieve the desired indentation levels more efficiently, as shown in the example output below.
Note: ^ is *odsescapechar*

```
Type: n (%)
  Category: n (%)
    Subcategory1: n (%)
      Subcategory2: n (%)
-------------------------------------------------------------

Any relevant protocol deviation

Any relevant eligibility deviation

  Randomized or treated with study drug under > 1 subject
  identifier
  Migraine history issue
    Experienced migraines for < 1 year prior to screening
    ≥ 9 moderate or severe migraines per month in the 3
    months prior to screening
  Cardiovascular disease risk factor
    Ischemic coronary artery disease
    Other significant underlying cardiovascular disease
```

10. ***PROC REPORT* - Splitting and Repeating Variables:** When dealing with a large number of variables in a single report, it can be challenging to maintain readability. To address this, we can split the variables across multiple pages while retaining a few key variables on each page for identification purposes. This ensures that the report remains readable and that important identifiers are always visible.

    In ***PROC REPORT***, you can achieve this by using the ***PAGE*** and ***ID*** options. Here's how you can do it:

a) ***PAGE Option:*** This option allows you to split the report at a specific variable. When the report reaches this variable, it will start a new page.
b) ***ID Option:*** This option ensures that certain key variables are repeated on each page, making it easier to identify and track the data across multiple pages such as Make and Model.

```
ods rtf file='S:\Test\cars.rtf';
options nodate nonumber;
title1 "(*ESC*)S={fontstyle=roman}Car Details";
proc report data = sashelp.cars;
    column make model type origin drivetrain msrp invoice enginesize;
    define make / display id;
    define model / display id;
    define type / display;
    define origin / display;
    define drivetrain / display page;
    define msrp / display;
    define invoice / display;
    define enginesize / display;
run;
ods rtf close;
```

Car Details

Make	Model	Type	Origin
Acura	MDX	SUV	Asia
Acura	RSX Type S 2dr	Sedan	Asia
Acura	TSX 4dr	Sedan	Asia
Acura	TL 4dr	Sedan	Asia
Acura	3.5 RL 4dr	Sedan	Asia
Acura	3.5 RL w/Navigation 4dr	Sedan	Asia
Acura	NSX coupe 2dr manual S	Sports	Asia
Audi	A4 1.8T 4dr	Sedan	Europe
Audi	A41.8T convertible 2dr	Sedan	Europe
Audi	A4 3.0 4dr	Sedan	Europe
Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe
Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe
Audi	A6 3.0 4dr	Sedan	Europe
Audi	A6 3.0 Quattro 4dr	Sedan	Europe
Audi	A4 3.0 convertible 2dr	Sedan	Europe
Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe
Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe
Audi	A6 4.2 Quattro 4dr	Sedan	Europe
Audi	A8 L Quattro 4dr	Sedan	Europe
Audi	S4 Quattro 4dr	Sedan	Europe
Audi	RS 6 4dr	Sports	Europe
Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe
Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe
Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe
Audi	A6 3.0 Avant Quattro	Wagon	Europe
Audi	S4 Avant Quattro	Wagon	Europe
BMW	X3 3.0i	SUV	Europe
BMW	X5 4.4i	SUV	Europe
BMW	325i 4dr	Sedan	Europe
BMW	325Ci 2dr	Sedan	Europe
BMW	325Ci convertible 2dr	Sedan	Europe
BMW	325xi 4dr	Sedan	Europe
BMW	330i 4dr	Sedan	Europe
BMW	330Ci 2dr	Sedan	Europe
BMW	330xi 4dr	Sedan	Europe
BMW	525i 4dr	Sedan	Europe
BMW	330Ci convertible 2dr	Sedan	Europe

Car Details

Make	Model	DriveTrain	MSRP	Invoice	Engine Size (L)
Acura	MDX	All	$36,945	$33,337	3.5
Acura	RSX Type S 2dr	Front	$23,820	$21,761	2
Acura	TSX 4dr	Front	$26,990	$24,647	2.4
Acura	TL 4dr	Front	$33,195	$30,299	3.2
Acura	3.5 RL 4dr	Front	$43,755	$39,014	3.5
Acura	3.5 RL w/Navigation 4dr	Front	$46,100	$41,100	3.5
Acura	NSX coupe 2dr manual S	Rear	$89,765	$79,978	3.2
Audi	A4 1.8T 4dr	Front	$25,940	$23,508	1.8
Audi	A41.8T convertible 2dr	Front	$35,940	$32,506	1.8
Audi	A4 3.0 4dr	Front	$31,840	$28,846	3
Audi	A4 3.0 Quattro 4dr manual	All	$33,430	$30,366	3
Audi	A4 3.0 Quattro 4dr auto	All	$34,480	$31,388	3
Audi	A6 3.0 4dr	Front	$36,640	$33,129	3
Audi	A6 3.0 Quattro 4dr	All	$39,640	$35,992	3
Audi	A4 3.0 convertible 2dr	Front	$42,490	$38,325	3
Audi	A4 3.0 Quattro convertible 2dr	All	$44,240	$40,075	3
Audi	A6 2.7 Turbo Quattro 4dr	All	$42,840	$38,840	2.7
Audi	A6 4.2 Quattro 4dr	All	$49,690	$44,936	4.2
Audi	A8 L Quattro 4dr	All	$69,190	$64,740	4.2
Audi	S4 Quattro 4dr	All	$48,040	$43,556	4.2
Audi	RS 6 4dr	Front	$84,600	$76,417	4.2
Audi	TT 1.8 convertible 2dr (coupe)	Front	$35,940	$32,512	1.8
Audi	TT 1.8 Quattro 2dr (convertible)	All	$37,390	$33,891	1.8
Audi	TT 3.2 coupe 2dr (convertible)	All	$40,590	$36,739	3.2
Audi	A6 3.0 Avant Quattro	All	$40,840	$37,060	3
Audi	S4 Avant Quattro	All	$49,090	$44,446	4.2
BMW	X3 3.0i	All	$37,000	$33,873	3
BMW	X5 4.4i	All	$52,195	$47,720	4.4
BMW	325i 4dr	Rear	$28,495	$26,155	2.5
BMW	325Ci 2dr	Rear	$30,795	$28,245	2.5
BMW	325Ci convertible 2dr	Rear	$37,995	$34,800	2.5
BMW	325xi 4dr	All	$30,245	$27,745	2.5
BMW	330i 4dr	Rear	$35,495	$32,525	3
BMW	330Ci 2dr	Rear	$36,995	$33,890	3
BMW	330xi 4dr	All	$37,245	$34,115	3
BMW	525i 4dr	Rear	$39,995	$36,620	2.5

STRATEGIES FOR QUALITY CONTROL

1. **Problem:** Double characters or graphic characters often emerge during the import of external files, particularly from free-text entries, complicating their identification and management.

   Tips: However, using the compress function with the 'g' modifier makes this process simpler by enabling effective detection of double characters or graphic characters, which is especially useful when dealing with external files.

   For example: The g modifier in the COMPRESS function adds graphic characters to the list of characters to be removed in the variable Graph as shown in the example.

```
data CHECK;
    set DEVIATION;
    graph = compress(DSTERM," ", "g");
    len = length(DSTERM);
    klen = klength(DSTERM);
run;
```

	dsterm	Graph	len	klen
1	In the 16th century, an age of great marine and terrestrial exploration, Ferdinand Magellan led the first expedition to sail around the world. As a young Portuguese noble, he served the king of Portugal, but he became involved in the quagmire of political intrigue at court and lost the king's favor.	'	3	1
2	｡｡just having double chars	｡｡	6	2
3	青い (あおい	青いあおい	15	5

2. **Problem:** Data analysts and programmers often face challenges in managing and understanding the structure of datasets, particularly when dealing with large volumes of data or frequently changing datasets. Manually inspecting each dataset for metadata such as variable names, types, lengths, and labels is time-consuming and prone to errors. For example: Finding the minimum and maximum dates across all datasets in a library is often demanding and time-consuming process. The addition of new datasets during live studies adds further complexity and increases the chance of errors.

   Tips: In SAS, *dictionary.columns* is a key resource for retrieving metadata about dataset columns. It offers detailed insights, including variable names, types, lengths, and labels. By doing so, users can create dynamic reports that reflect the current state of datasets without hardcoding variable information. This approach will facilitate data validation, enhance documentation efforts, and improve overall data governance by providing a comprehensive view of dataset attributes. For example: This functionality supports the user to calculate the minimum and maximum dates or any other data validation by confirming the existence and attributes of specific columns i.e., formats, length.

3. **Problem:** In SAS, users often need to check for the existence of macro variables to ensure that their code runs smoothly without errors related to undefined variables. However, the absence of a straightforward method to verify the existence of a macro variable can lead to complications, including runtime errors and unexpected results in data processing or reporting.

   Tips: %SYMEXIST function is effectively used to determine whether specific macro variables are defined in the current SAS session. This will enable users to conditionally execute code segments based on the presence or absence of these variables, enhancing code robustness and improving error handling.

   By employing %SYMEXIST, users can streamline their workflows, reduce debugging time, and ensure that dependent processes are only executed when the necessary macro variables are available. This leads to more reliable and maintainable SAS programs.

   For example: If the macro is not compiled and you encounter an error such as “apparent symbolic reference not resolved,” you can use the following code to avoid this issue through conditional execution. The code checks whether the macro variable STARTDATE already exists before proceeding with the execution.

```
%if %symexist(STARTDATE) %then %do;
      %put &STARTDATE.;
%end;
```

4. **Problem:** Effective management of macro variables is crucial for efficient programming and data analysis in SAS. As projects progress, it becomes necessary to remove or reset these macro variables to avoid conflicts and keep the code organized. However, without a systematic method, users may find it difficult to identify and eliminate unused or outdated macro variables, which can lead to errors and confusion in their SAS programs.

   Tips: ***%SYMDEL*** function is used to delete specified macro variables from the current SAS session. This will provide users with a systematic method to manage their macro variable environment, ensuring that only relevant variables remain defined. By implementing %SYMDEL, users can enhance code clarity, reduce the risk of unintended variable reuse, and improve overall program maintainability.

   For example: The below statement will delete the STARTDATE macro variable.

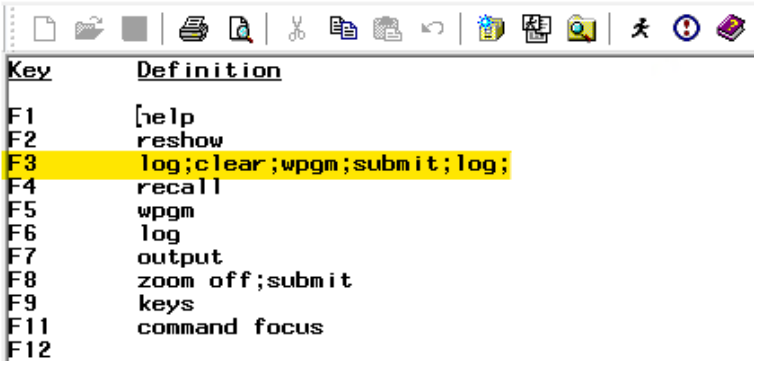
```
%symdel STARTDATE;
```

TIME SAVING TIPS

1. **To clear the log for each run:** To ensure efficient log management and troubleshooting, configuring SAS using tools menu to clear the existing logs and display the log window. This process will help in maintaining a clean and organized log environment, making it easier to identify and resolve any issues that arise during the execution of SAS programs. By regularly clearing the logs, you can prevent the accumulation of unnecessary data, which can slow down the system and complicate the debugging process. Displaying the log window

allows for real-time monitoring of program execution, providing immediate feedback on any errors or warnings that may occur. This proactive approach to log management is essential for maintaining optimal performance and ensuring the accuracy and reliability of your SAS analyses.

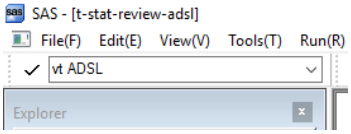
Tips: We can configure the F3 key to **log;clear;wpgm;submit;log;** via **Tools > Options > Keys**. This setup will clear the SAS log before each program execution and open the log window. This setup will help us ensure that everything runs smoothly before checking the output.



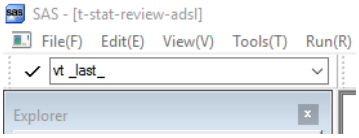
2. **Viewing dataset:** Users require a quick method to open datasets to avoid the time-consuming process of scrolling and searching for specific libraries within the SAS environment.

   Tips:

   - Utilize the command **vt dataset_name** to open datasets directly in the SAS environment. This command allows users to view the contents of the specified dataset, facilitating efficient data inspection.



   - Utilize the command ***vt _last_*** to open the last run dataset and view the contents directly in the SAS environment.



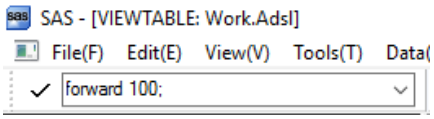
   By implementing a direct command or shortcut to access datasets, users can enhance their workflow, ensuring they can promptly verify data integrity and make necessary adjustments without unnecessary delays. This approach will significantly improve productivity and accuracy in data handling tasks.

3. **Record Navigation:** In a large SAS dataset, pinpointing a specific record for debugging can be daunting because of the sheer volume of data. Users require an efficient method to locate and isolate particular records to troubleshoot and resolve data issues effectively.

   Tips: The forward and backward options in SAS are essential for navigating to a particular record exactly where they are in the SAS dataset even when they contain vast amounts of data. These options help users locate specific records efficiently. They are also useful for re-executing recent commands without retyping them and revisiting older commands. By entering a number, users can jump to a specific position or move relative to their current position to find another record.

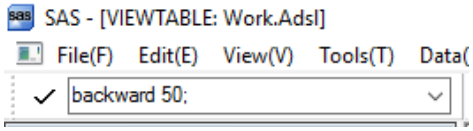
   Example: Imagine you have a dataset with 10,000 records, and you are currently viewing record 5,000. If you want to move forward by 100 records, you will use the forward option in command bar with the number 100. Conversely, if you want to move back by 50 records, you will use the backward option in command bar with the number 50.

   - Forward Command: forward 100 (moves to record 5,100)



- Backward Command: backward 50 (moves to record 4,950)

This functionality streamlines the process of navigating through large datasets, making it easier to locate and analyse specific records.



4. **To close all the datasets in SAS window:** To close all datasets opened in the SAS window for data investigation, you must manually navigate through the SAS interface and close each dataset individually. This process is time-consuming and requires significant manual effort.

   Tips: You can quickly close all opened datasets in the SAS window using the following macro code:

```
%macro Closefile;
 %do i=1 %to N;
    dm "next viewtable; end;";
 %end;
%mend Closefile;
%Closefile;
```

   This macro iterates to N through the open datasets and closes each one, saving you time and reducing manual effort. Just run this code in your SAS environment, and it will handle the rest for you.

5. **File open issue during SAS code execution:** When running a SAS program, if an Excel file is open, it may cause an error and allow the program to continue executing subsequent steps. This can lead to incomplete or incorrect data processing. Additionally, the program may need to be rerun after the initial execution is complete, resulting in inefficiencies and potential data integrity issues.

   Tips: To prevent runtime errors and ensure accurate data processing, it is crucial to check if the Excel file is open before running the SAS program. If the file is open, the program should raise a warning and halt execution. This approach helps avoid potential issues during runtime. The following SAS code can be used to identify if the file is open, issue a warning, and stop execution to prevent any problems:

```
%let rc = %sysfunc(filename(myfile, &DIRPATH.Programs\00 Config\TitlesFootnotes.csv));
%let fid = %sysfunc(fopen(&myfile));
        %if &fid = 0 %then %do;
                %put ATTENTION: The file TitlesFootnotes is open by another application.;
                %return;
        %end;
        %else %do;
                %let rc = %sysfunc(fclose(&fid));
         %end;
%let rc = %sysfunc(filename(myfile));
```

6. **Keep and Show options in Command bar:** Remembering the entire variable name to search in a dataset can be quite tedious and challenging. This issue often arises when dealing with large datasets containing numerous variables, making it difficult to recall each variable's exact name. This can lead to inefficiencies and errors in data analysis, as users may struggle to locate and utilize the necessary variables.

   Tips: If you're unsure about which variable names to display or retain in SAS datasets, you can type show or keep option in command bar and press Enter. This command will list all the variables in the dataset, allowing you to manually select the ones you want to keep or show in the SAS window as below.

KEY BOARD SHORTCUTS

Editing and Navigation

1. F3: Executes the selected code or the entire program if no code is selected.
2. F8: Submits the code to be run, like clicking the “Submit” button.
3. Ctrl + Tab: Cycles through open windows, allowing quick navigation between the editor, log, and output windows.
4. Ctrl + S: Saves the current program or data step you are working on.
5. Ctrl + N: Opens a new program editor window.
6. Comment / Uncomment Code:
    - Ctrl + /: Comments out the selected code.
    - Ctrl + Shift + /: Uncomments the selected code.
7. Indent / Outdent Code:
    a. Tab: Indents the selected code or lines.
    b. Shift + Tab: Outdents the selected code or lines.
8. Find and Replace:
    a. Ctrl + F: Opens the Find dialog to search for text in the code.
    b. Ctrl + H: Opens the Replace dialog to find and replace text in the code.
9. Go to Line:
    a. Ctrl + G: Opens a dialog where you can enter a line number to jump directly to that line.
10. Changing case:
    a. Ctrl + Shift + U: Converts the value to uppercase.
    b. Ctrl + Shift + L: Converts the value to lowercase.

Code Execution and Debugging

1. F9: Sets or clears a breakpoint in your code, useful for debugging.
2. F10: Steps over the current line of code during debugging.
3. F11: Runs the code up to the cursor’s current position, useful for debugging specific sections.

Data Handling

1. F7: Opens the data table view for the currently selected dataset in the editor.
2. Ctrl + L: Brings up the log window to view errors, warnings, and messages.
3. Ctrl + O: Opens the output window to view results and output data.

Miscellaneous

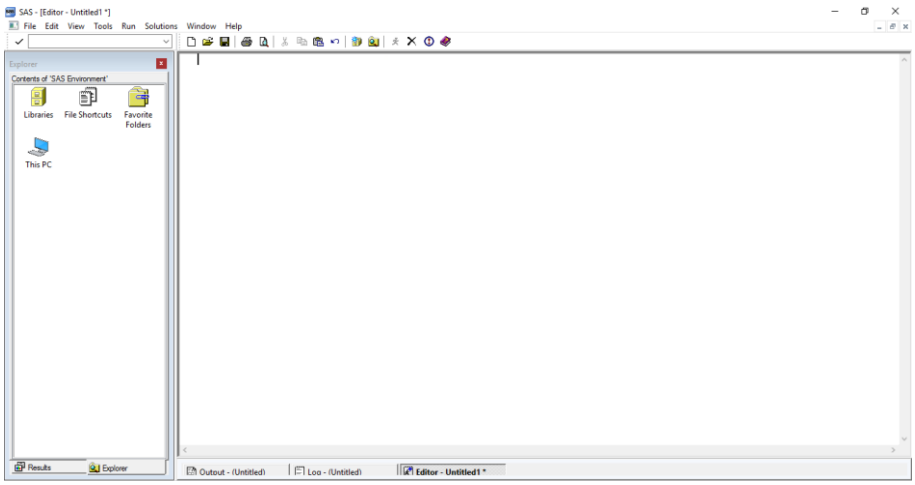
1. Show/Hide Code Window: Ctrl + Q: Minimizes or restores the code editor window.
2. Zoom In/Out: Ctrl + Mouse Wheel: Zooms in or out on the code editor for better readability.
3. Auto-complete Code: Ctrl + Space: Provides auto-completion suggestions for code, such as variable names and functions.

SAS KEYBOARD MACROS

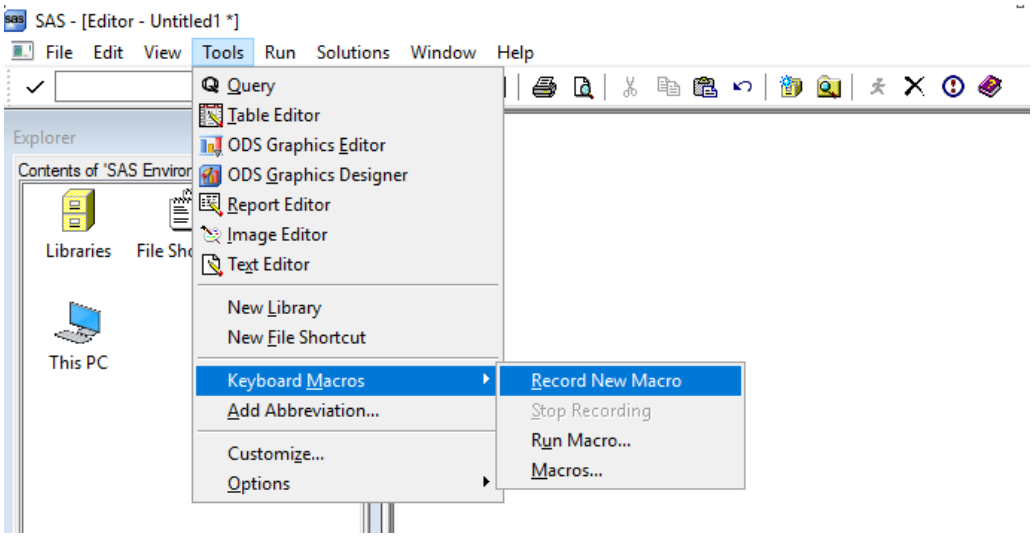
If you often find yourself typing the same code repeatedly, it can become a time-consuming task. You might also find it challenging to recall the precise syntax for specific code snippets you need to create. To streamline this process, you can create a keyboard macro or abbreviation to store the frequently used code. Once set up, you can effortlessly insert this repetitive code using a simple key combination or a short text string. This not only saves time but also reduces the likelihood of manual entry errors, ultimately enhancing your overall coding efficiency and allowing you to focus on more complex tasks.

Here are the steps to record your own keyboard macro in the SAS Enhanced Editor.

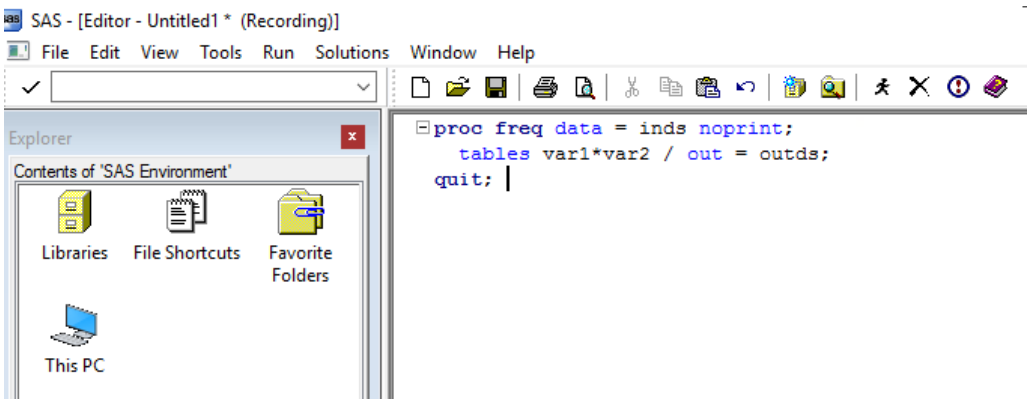
1. Go to **View** > **Enhanced Editor**



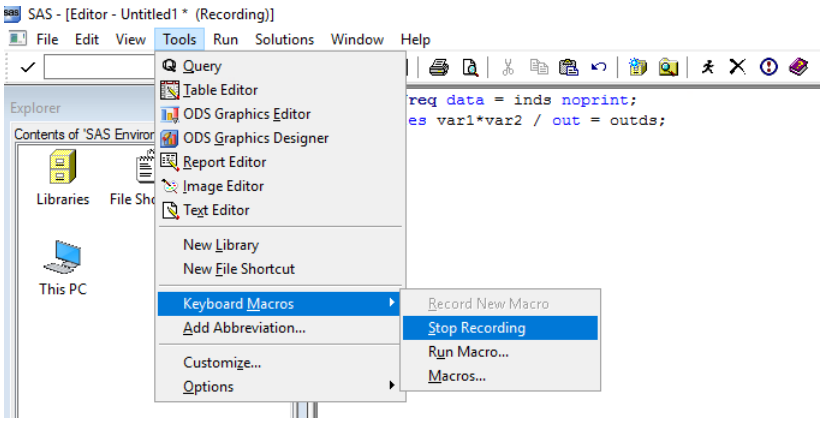
2. Select **Tools** > **Keyboard Macros** > **Record New Macro**



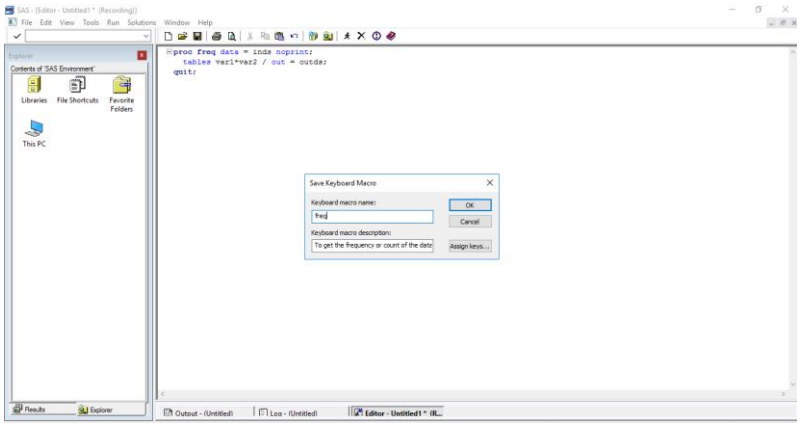
3. Enter the code you want to save in the Enhanced Editor to record as a macro. Make sure to type slowly, since any backspaces will be captured in the recording.



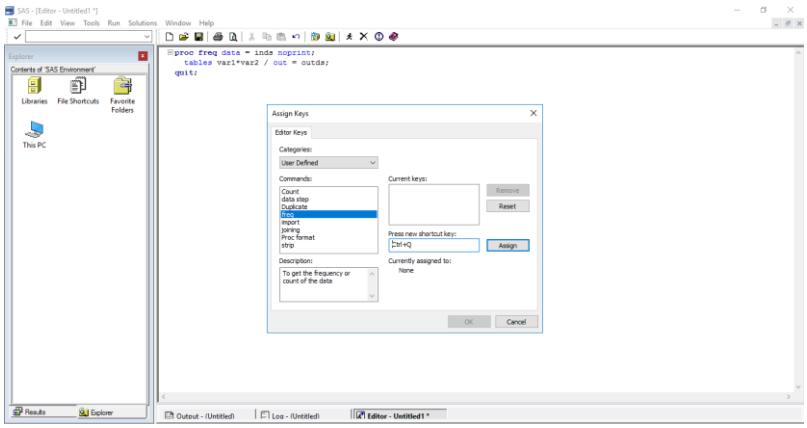
4. After entering the code in the Enhanced Editor, select **Tools** > **Keyboard Macros** > **Stop Recording** to finish the recording of a macro.



5. Enter the **Keyboard Macro Name** and **Keyboard Macro Description** in the dialog box that appears. Click **OK** to add the created macro to the list of existing macros or click **Assign keys** to assign a function key or key combination to the macro. Make sure not to use a combination that is already assigned to another macro.



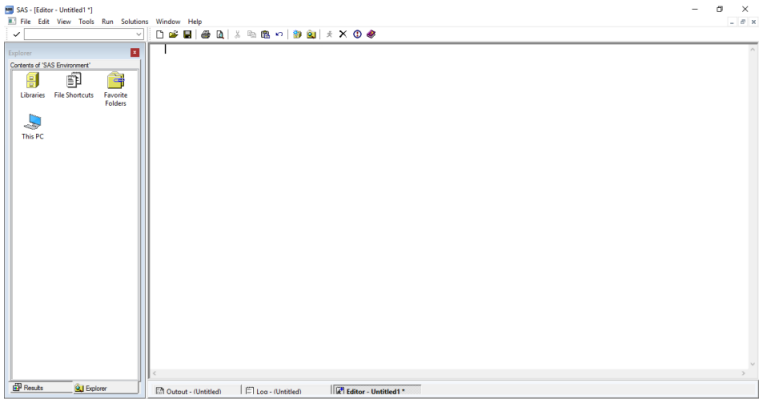
6. For key assignment, select "None" under **Press new shortcut key**. Then, press the desired key or key combination on the keyboard. Once the key is entered, click **Assign**, and then **OK** to close the dialog box.



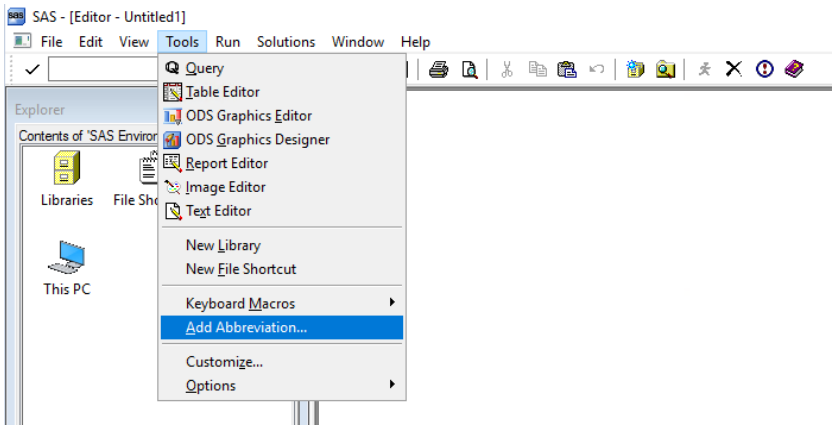
7. To use the created keyboard macro, simply press the assigned key or key combination in the Enhanced Editor.

Alternatively, we can define the keyboard abbreviations as follows.

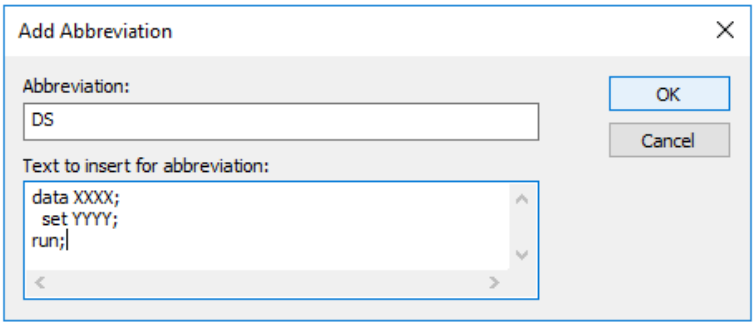
1. Go to **View** > **Enhanced Editor**



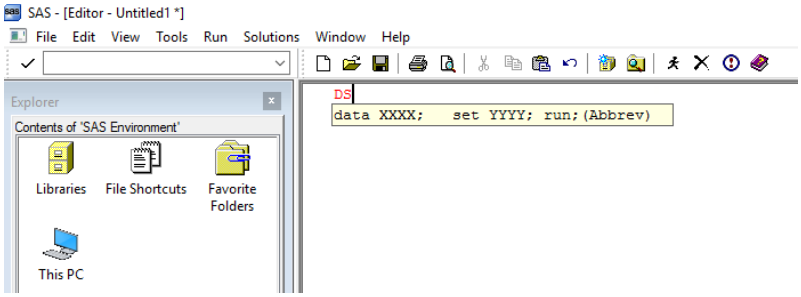
2. Select **Tools** > **Add Abbreviation**



3. Name the abbreviation and input the code into the **'Text to insert for abbreviation'** box. Once the code is entered, press **OK**.



4. In the Enhanced Editor, when you type the abbreviation which is set for the shortcut, like DS in this example, a window appears showing the text linked to that abbreviation. Please note that the abbreviation box is case sensitive, make sure to use the same case for the shortcut as when it is defined. If you press the ENTER key, the associated text will be inserted into the Enhanced Editor automatically. However, if you continue typing without pressing ENTER, the window will close, and the text won't be inserted.



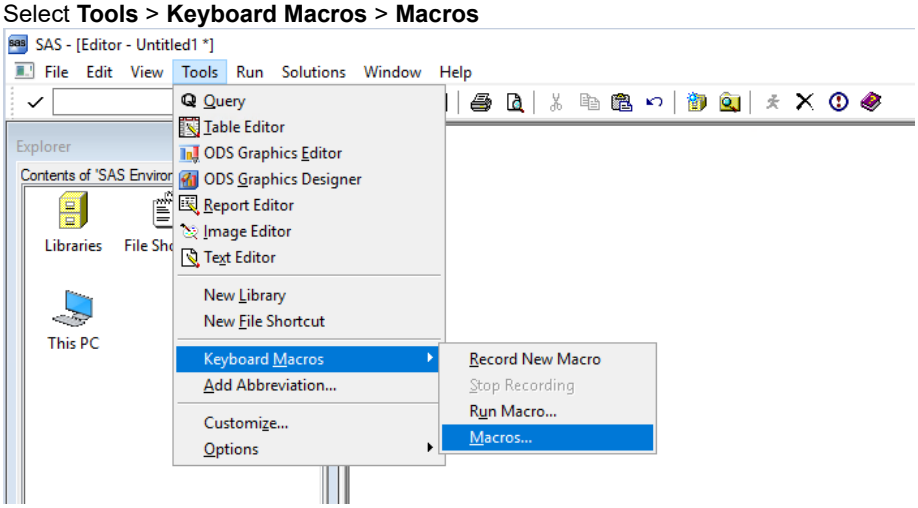
These abbreviations will only function within the local SAS Enterprise Guide installation where they were initially set up. Through keyboard macro files (.kmf), SAS users can share their predefined keyboard abbreviations with others.

KEYBOARD MACRO FILES (.kmf)

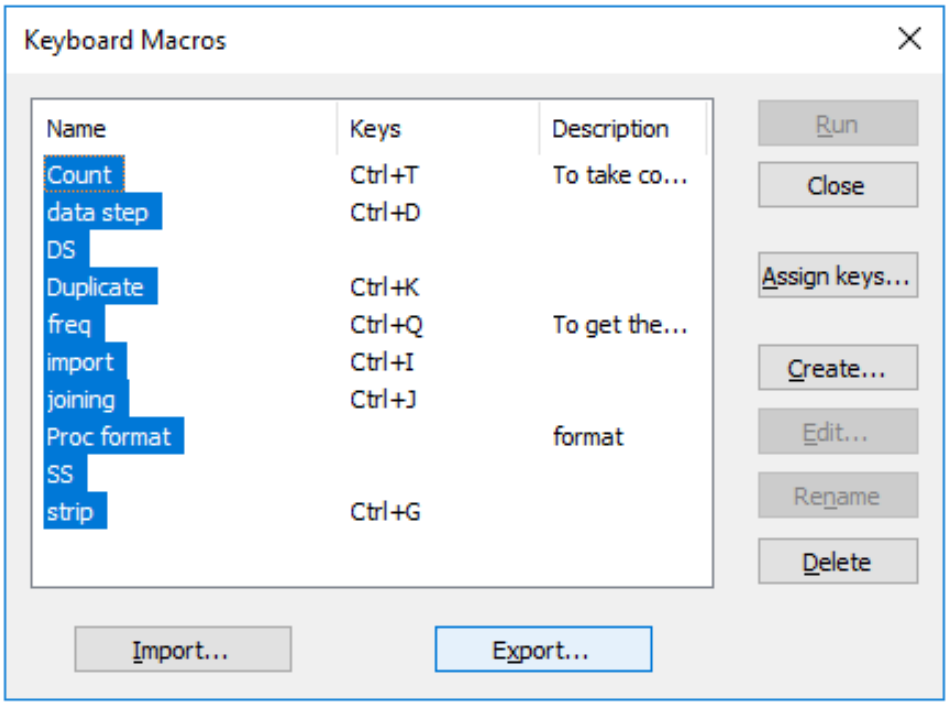
Users can export their keyboard macros/abbreviations into a .kmf file, which can be easily imported into another installation. This capability allows for the seamless sharing of custom shortcuts among different users or environments. Moreover, the .kmf file serves as a backup for these keyboard mappings, enabling users to save their settings and restore them as needed.

Below are the steps to create the .kmf file from the keyboard macros/abbreviations.

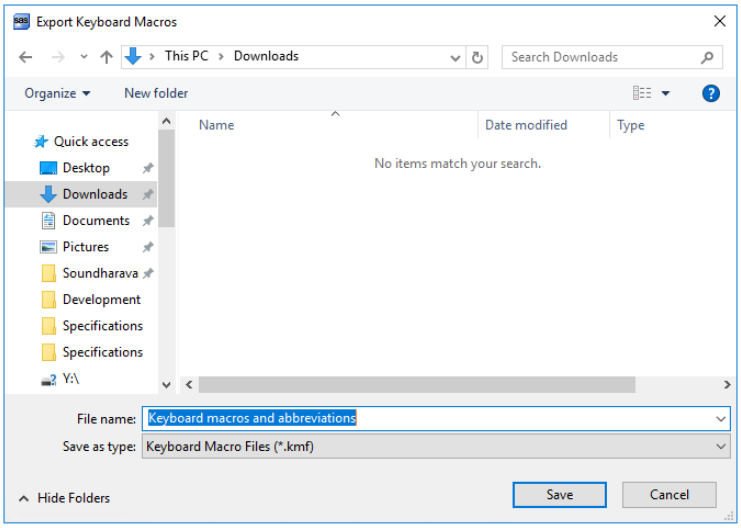
1. Select **Tools** > **Keyboard Macros** > **Macros**



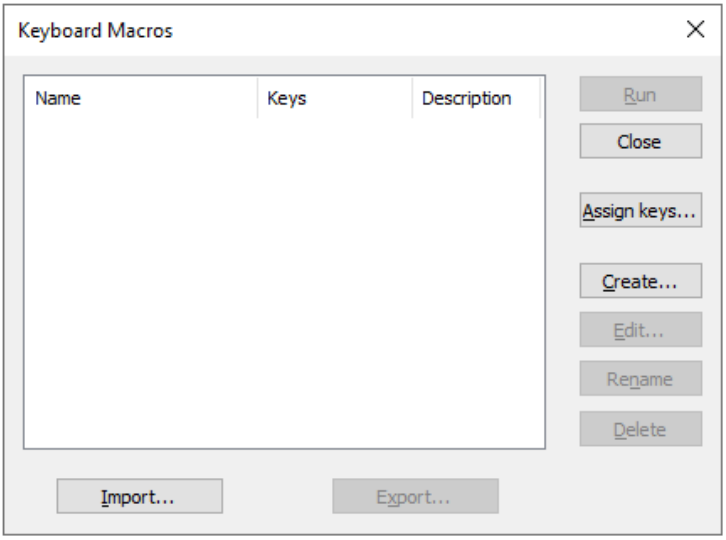
2. Once the keyboard macro window is open, all existing abbreviations and keyboard macros will be displayed. Users can create, edit, run, rename, and delete keyboard abbreviations and macros within this window. Additionally, multiple keyboard abbreviations can be selected at once for exporting the .kmf file.



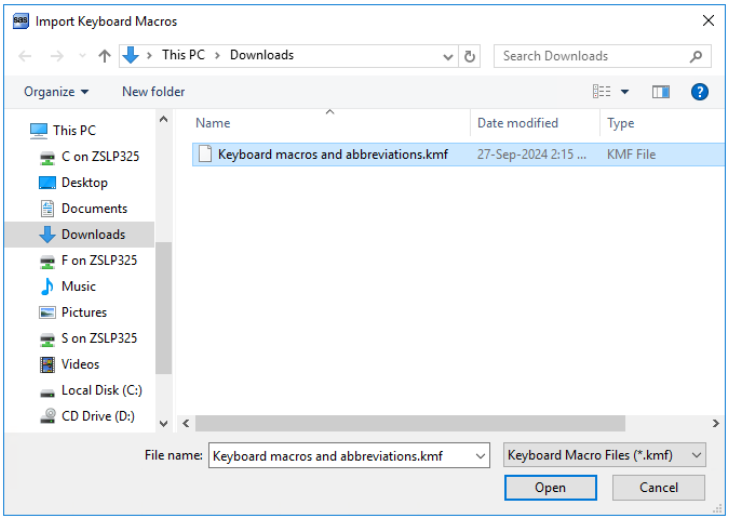
3. Choose all the abbreviations or keyboard macros that needed to export, then click the Export option in the keyboard macro window. This action will generate a .kmf file.



4. After the .kmf file is exported and shared, other users can select and import it into their SAS environment.



5. Please note that keyboard macros and abbreviations are imported exactly as they were defined and exported in the .kmf file. Once the .kmf file is imported through the keyboard macros window, users can utilize the keyboard abbreviations code by entering the corresponding abbreviation in the SAS Enhanced Editor. To use the imported keyboard macros, users must assign a shortcut key to them.



CONCLUSION

The tips and tricks we've discussed are just a glimpse of the extensive techniques available for SAS programming. By integrating these methods into your workflow, you can streamline processes, minimize manual effort, and achieve more accurate and timely results. Keeping up with the latest SAS features and enhancements is essential, as SAS continually evolves with new functionalities. This allows you to leverage the latest tools and techniques to boost your productivity and analysis quality.

REFERENCES

- Do More with Less: Programming Efficiently with Keyboard Macros (sas.com)

RECOMMENDED READING

- https://www.lexjansen.com/phuse/2012/cc/CC03.pdf
- SAS tips and tricks: Users-tell-all edition - SAS Users

CONTACT INFORMATION

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

Author Name: Deepak Ananthan
Company: Zifo
Address: 21A, Anna Salai, Littlemount, Saidapet, Chennai, Tamil Nadu 600015
City / Postcode: Chennai / 600015
Work Phone: +91 44 43114002
Email: Deepak.A@zifornd.com
Website: https://zifornd.com/