PharmaSUG 2024 – Paper PO-158

If its not broke, don’t fix it; existing code and the programmers' dilemma

Jay Iyengar, Data Systems Consultants LLC

ABSTRACT

In SAS shops and organizational environments, SAS® programmers have the responsibility of working with existing processes and SAS code which projects depend on to produce periodic output and results and meet deadlines. Some programming teams still cling to the old adage; if it not broke, don’t fix it. They’ve come to depend on code which runs clean, and is reliable. However, besides processing with no errors and warnings. there other criteria to judge the quality of a SAS program. Programming guidelines dictate that code should be well-documented, readable, and efficient, and conform to best practices. This paper challenges the conventional wisdom that code which works shouldn’t be modified.

INTRODUCTION

Programming work can be divided into two different project tasks; developing code from scratch, and modifying and updating existing code. This is the case regardless of the type of employment, whether full-time, contract, or consulting. It is usually the case when a programmer is brought in on a contract basis that they’re needed to update and modify existing code. A full-time employee may have left the company, and they need a temporary replacement to fill the void. So the contractor takes the role of a troubleshooter/firefighter who’s needed to keep the ship afloat until they have more time to go through the recuiting process, and hire a full-time developer. In this paper, I suggest procedures a programmer should follow and strategies to take for working with existing code. I also make specific recommendations for how to go about working with legacy SAS® code.

THE PROCESS OF REVISING CODE

How does a programmer go about renovating or updating existing code. Its important and necessary that a programmer understand the SAS process which they’re going to modify. Understanding code another programmer has developed rests on several aspects of the code; clarity, readability documentation of the code. Hopefully, the code you’re working with has been well-formatted using white space, indentation and other elements of style. If not, the task of understanding the code becomes more challenging.

Good documentation includes the purpose of the program, the author of the code, and the date it was written. This usually is contained within a program header at the start of the program. A well documented program will contain comments throughout the program which describe each step or key steps in the code. This makes it easier to understand what is happening in each step, as well as connect one step to the next in understanding the data flow throughout the program. Other key documented information includes the source SAS data sets or external files which are referenced and used to perform data extraction, and SAS output or output SAS data sets which are produced from the program.

FIND THE DEVELOPER WHO WROTE THE CODE

Hopefully the code is well-documented so the process of understanding the code will be straight forward. However, most of the time, a SAS consultant is brought in on a project to perform tasks which no one on staff is able to complete. Its entirely possible that the code was written by a former employee, who's now left the company. Its ideal to find the original author of the code, if possible. If the author of the code is still with the organization, then a best practice would be to put that programmer in charge of maintaining the code, since they're the most familiar with it. In this case, it wouldn't be necessary to bring in an external consultant for the project.

Find the Programmer who wrote the code!

Since the company is bringing in a consultant, its unlikely that the original author of the code is still employed by the company. On some projects, project managers might try to track down and contact former employees who authored the code. The former employee might not be willing to assist and discuss the project. In my own experience, I've worked for companies whom I've been contacted by after I've left the company to discuss projects I worked on while I was with them.

Nevertheless, as a SAS consultant, the search for the author of the code shouldn't involve going to great lengths, and tracking down professors at Stanford University as Vince Vaughn and Owen Wilson did in the movie 'The Internship'.

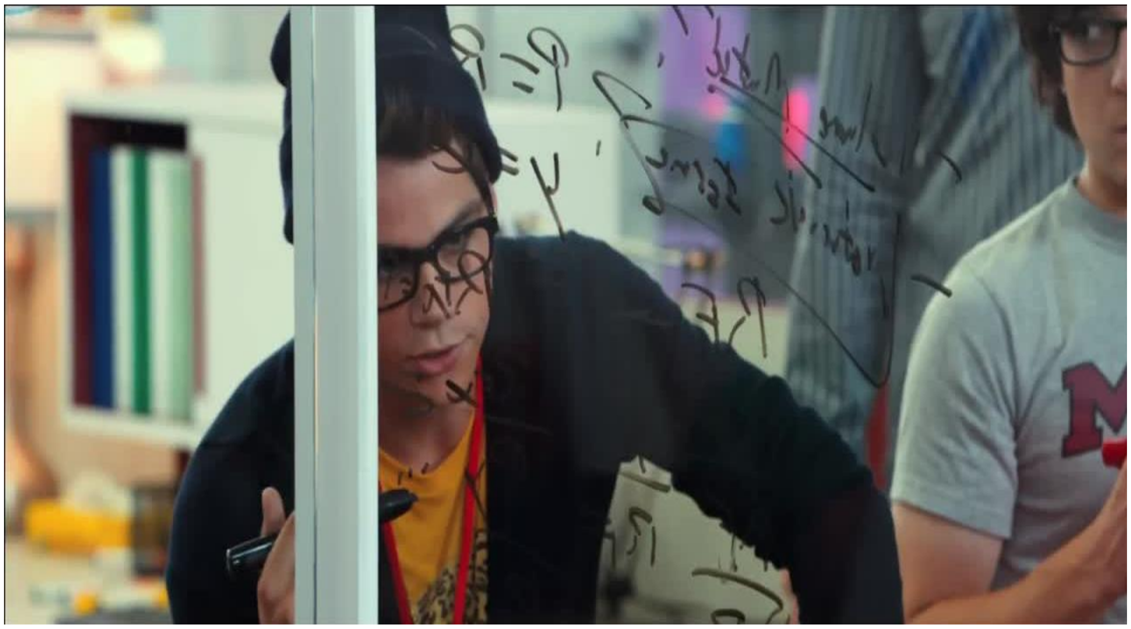
```
/**********************************/
/* Name: GET_CASE_MASTER.SAS      */
/* Date: 08/12/22                 */
/* Authors: JI                    */
/*                                */
/*--------------------------------*/
/* Maintained By: JI              */
/*--------------------------------*/
/* Purpose:                       */
/*                                */
/* Downloads Case Master file from */
/* AWS bucket into a text file    */
/*                                */
/* Imports text file and converts */
/* to permanent SAS data set      */
/*                                */
/* Excludes records from          */
/* SAS data set.                  */
/*                                */
/*--------------------------------*/
/* Input Files: AWS Text File     */
/*--------------------------------*/
/* Output Files: Case_Master.txt  */
/*               Case_Master      */
/**********************************/
```

Figure 1. Program Header documenting Author, Date, and Purpose of Program

In Figure 1, is a standard program header which documents key information of the program. The name of the developer who authored the code, and maintained the code is provided in the header as initials. The program header includes a description and purpose of the program, along with name of the program, and the date it was written. These are all important elements which should be required with every program which is written. Other useful information includes the source files, or input data sets used, and the output files which are produced from the code.

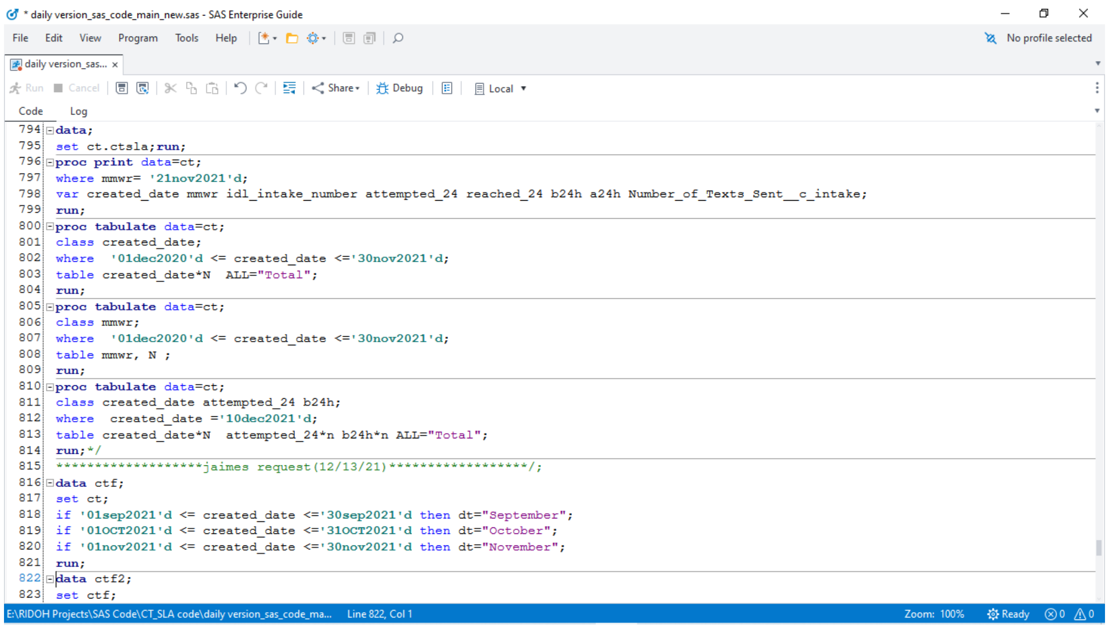
FORMAT THE CODE

The code which you are handed might be written in a style which is unreadable or illegible. Some programmers don't incorporate elemtents of style in their program which make the code easy to follow. These aspects include indentation, white space (adding lines between steps of the program), capitalization and use of case.. Instead DATA STEPS might be stacked on top of SAS PROCS in ensuing steps, making it hard to distinguish step boundaries. Multiple SAS statements might be packed together on one line of code, rather than giving each statement its own separate line in the program.



“*Format the Code*”

In Figure 2 below is a screenshot of an Enterprise Guide session, with a SAS program opened in the editor. Steps in the program have been stacked directly on top of each other which makes it challenging to discern the flow of the program. Besides line spacing, the code was written without indentation to aid the reader in deciphering it. Its not uncommon to encounter existing code written in this style on projects which a contract SAS programmer is assigned to.


* daily version_sas_code_main_new.sas - SAS Enterprise Guide
File Edit View Program Tools Help
No profile selected
daily version_sas...
Run Cancel Share Debug Local
Code Log

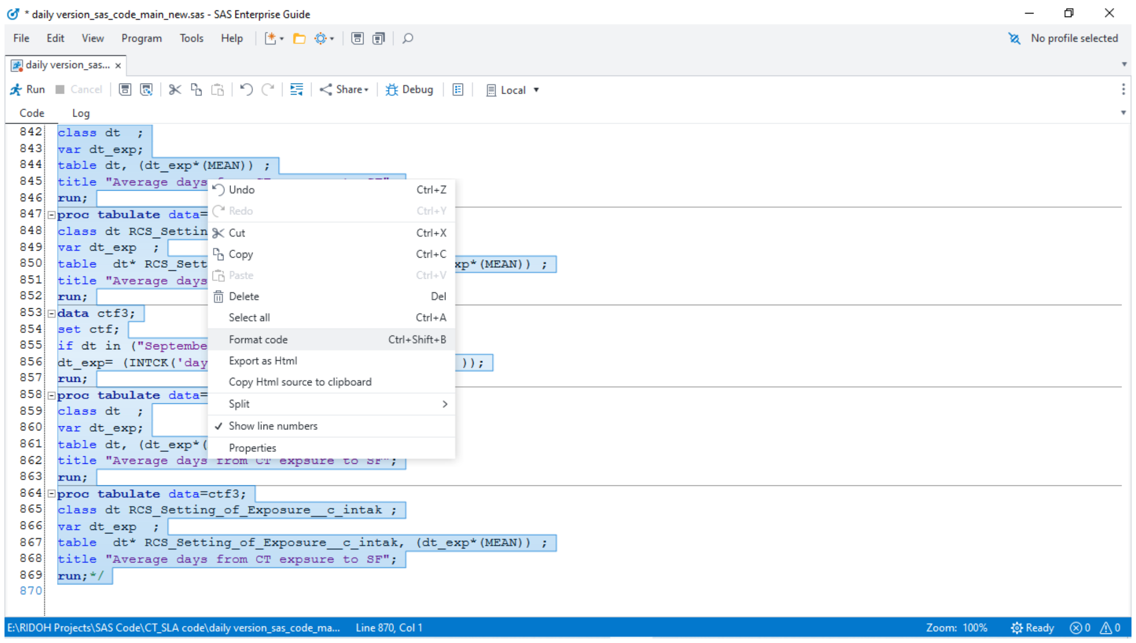
```
data;
set ct.ctsla;run;
proc print data=ct;
where mmwr= '21nov2021'd;
var created_date mmwr idl_intake_number attempted_24 reached_24 b24h a24h Number_of_Texts_Sent__c_intake;
run;
proc tabulate data=ct;
class created_date;
where  '01dec2020'd <= created_date <='30nov2021'd;
table created_date*N  ALL="Total";
run;
proc tabulate data=ct;
class mmwr;
where  '01dec2020'd <= created_date <='30nov2021'd;
table mmwr, N ;
run;
proc tabulate data=ct;
class created_date attempted_24 b24h;
where  created_date ='10dec2021'd;
table created_date*N  attempted_24*n b24h*n ALL="Total";
run;*/
********************jaimes request(12/13/21)******************/;
data ctf;
set ct;
if '01sep2021'd <= created_date <='30sep2021'd then dt="September";
if '01OCT2021'd <= created_date <='31OCT2021'd then dt="October";
if '01nov2021'd <= created_date <='30nov2021'd then dt="November";
run;
data ctf2;
set ctf;
```

E:\RIDOH Projects\SAS Code\CT_SLA code\daily version_sas_code_ma... Line 822, Col 1 Zoom: 100% Ready 0 0

Figure 2. Screenshot of SAS program in Enterprise Guide session

One of the features and capabilities of SAS Enterprise Guide is the ability to automatically format code according to specific criteria. The ‘FORMAT CODE’ feature allows you the ability to insert line spaces in between lines of code or between steps, and indentation for specific statement and keywords.

Figure 3 displays the same code example in Figure 2 with the addition of the drop-down menu showing the ‘Format Code’ option selected. To use the ‘FORMAT CODE’ feature, right-click in the program editor, and select ‘FORMAT CODE’ from the drop-down menu. This will automatically convert your program. There’s also a ‘FORMAT CODE’ button on the toolbar which you can use.



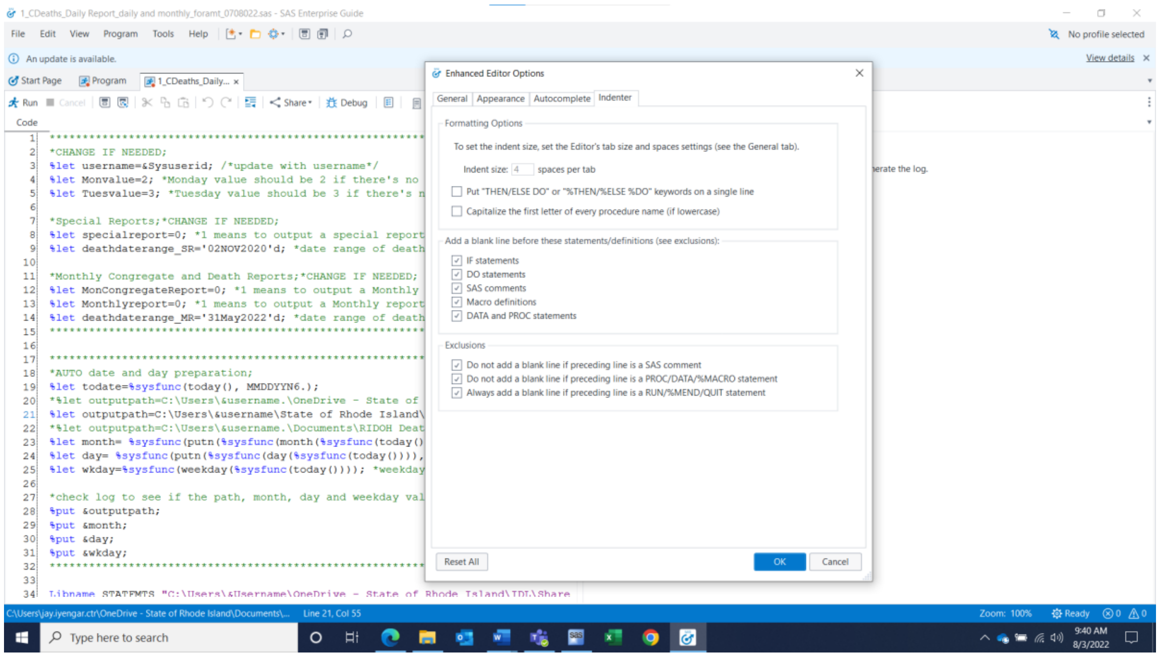
```
class dt  ;
var dt_exp;
table dt, (dt_exp*(MEAN)) ;
title "Average days
run;
proc tabulate data=
class dt RCS_Settin
var dt_exp  ;
table  dt* RCS_Sett                              xp*(MEAN)) ;
title "Average days
run;
data ctf3;
set ctf;
if dt in ("Septembe
dt_exp= (INTCK('day                              ));
run;
proc tabulate data=
class dt  ;
var dt_exp;
table dt, (dt_exp*(
title "Average days from CT exposure to SF";
run;
proc tabulate data=ctf3;
class dt RCS_Setting_of_Exposure__c_intak ;
var dt_exp  ;
table  dt* RCS_Setting_of_Exposure__c_intak, (dt_exp*(MEAN)) ;
title "Average days from CT exposure to SF";
run;*/
```

Figure 3. SAS program in Enterprise Guide session using ‘Format Code’ feature.

With the ‘FORMAT CODE’ feature, SAS Enterprise Guide will automatically apply these elements to your code to enhance its readability. Using this feature is preferable to going through the code line by line and manually adding line spaces between steps, and indenting code inside of a DATA STEP or PROC.

Enterprise Guide 8.3 gives you the option of specifying where within your program blank lines should be inserted, and in relation to what specific SAS statements and constructs.

If you go under the ‘TOOLS’ menu, then select ‘OPTIONS’, SAS will open up a dialog box with a menu panel on the left side. From here, select ‘SAS PROGRAMS’. Then click on the ‘EDITOR OPTIONS’ box at the top. This will open up the ‘ENHANCED EDITOR OPTIONS’ box with 4 new tabs. If you select the ‘INDENTER’ tab, you’e screen should appear similar to Figure 4.



```
*************************************************************
*CHANGE IF NEEDED;
%let username=&Sysuserid; /*update with username*/
%let Monvalue=2; *Monday value should be 2 if there's no
%let Tuesvalue=3; *Tuesday value should be 3 if there's n

*Special Reports;*CHANGE IF NEEDED;
%let specialreport=0; *1 means to output a special report
%let deathdaterange_SR='02NOV2020'd; *date range of death

*Monthly Congregate and Death Reports;*CHANGE IF NEEDED;
%let MonCongregateReport=0; *1 means to output a Monthly
%let Monthlyreport=0; *1 means to output a Monthly report
%let deathdaterange_MR='31May2022'd; *date range of death
*************************************************************


*************************************************************
*AUTO date and day preparation;
%let todate=%sysfunc(today(), MMDDYYN6.);
*%let outputpath=C:\Users\&username.\OneDrive - State of
%let outputpath=C:\Users\&username\State of Rhode Island\
*%let outputpath=C:\Users\&username.\Documents\RIDOH Deat
%let month= %sysfunc(putn(%sysfunc(month(%sysfunc(today()
%let day= %sysfunc(putn(%sysfunc(day(%sysfunc(today()))),
%let wkday=%sysfunc(weekday(%sysfunc(today()))); *weekday

*check log to see if the path, month, day and weekday val
%put &outputpath;
%put &month;
%put &day;
%put &wkday;
*************************************************************

Libname STATFMTS "C:\Users\&Username\OneDrive - State of Rhode Island\IDI\Share
```

Enhanced Editor Options

General | Appearance | Autocomplete | Indenter

Formatting Options

To set the indent size, set the Editor's tab size and spaces settings (see the General tab).

Indent size: 4 spaces per tab

- [ ] Put "THEN/ELSE DO" or "%THEN/%ELSE %DO" keywords on a single line
- [ ] Capitalize the first letter of every procedure name (if lowercase)

Add a blank line before these statements/definitions (see exclusions):

- [✓] IF statements
- [✓] DO statements
- [✓] SAS comments
- [✓] Macro definitions
- [✓] DATA and PROC statements

Exclusions

- [✓] Do not add a blank line if preceding line is a SAS comment
- [✓] Do not add a blank line if preceding line is a PROC/DATA/%MACRO statement
- [✓] Always add a blank line if preceding line is a RUN/%MEND/QUIT statement

C:\Users\jay.iyengar.ctr\OneDrive - State of Rhode Island\Documents\...   Line 21, Col 55

Figure 4. SAS EG session with 'ENHANCED EDITOR OPTIONS' and 'INDENTER' tab.

In Figure 4 above is a screenshot of the Enterprise Guide session showing the 'INDENTER' tab and 'ENHANCED EDITOR OPTIONS'. The 'INDENTER' feature allows you to specify that conditional logic constructs, do loops, and macro conditional logic all should appear on a single line of code. In addition, you are able to specify tab definitions in number of spaces for indenting purposes.

TEST RUN THE CODE

One of the first tasks which a SAS consultant needs to perform is to test run the code to enure it works, and if necessary to debug the code of errors, warning, or notes such as uninitialized variables. Testing code is an essential step in the development phase of software development. Its mandatory to test code prior to putting it into production.

Executing and running code can be a time consuming process. The run time of your code can be long depending on the size and scale of data you're working with. Other factors affecting processing time include the length of the code (number of lines of code), and the computing resources available at your site, including memory, storage space, and internet bandwidth.

One good rule of thumb is to take a sample of the data you're using, and perform a test run on the sample. Another good practice is to test run the code without any observations. You can set the number of observations which are processed using the OBS= and FIRSTOBS= options. OBS= and FIRSTOBS= can be specified as global options or alternately as data set options.

Find The Bug!

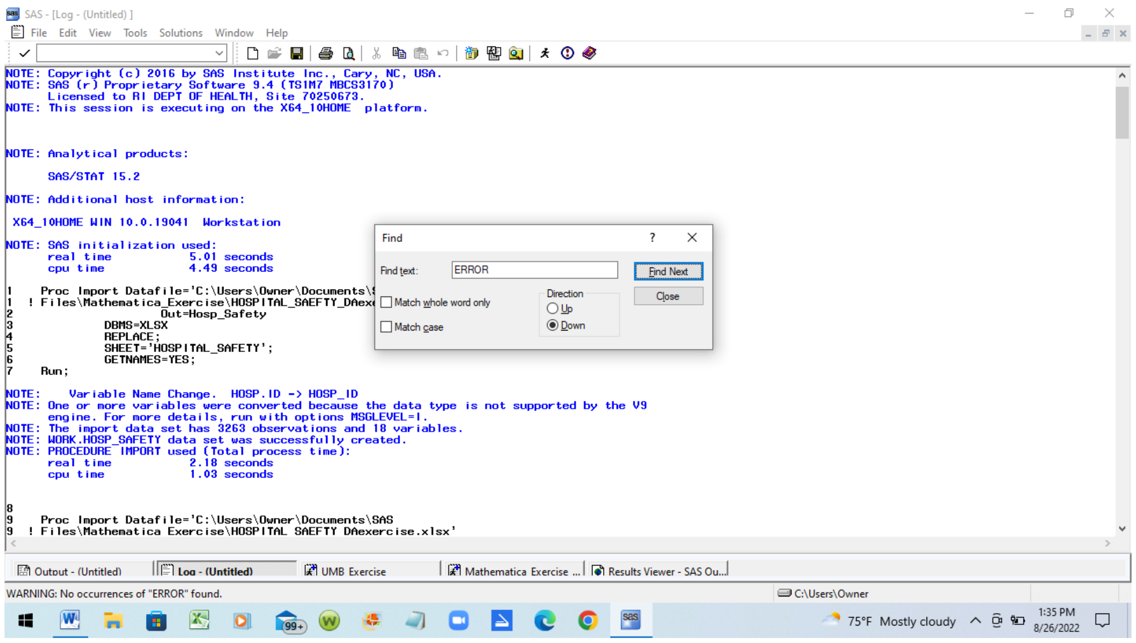
When test running lengthy code, I recommend using global options and specifying them at the beginning of your program. This makes more sense than specifying the options every time you read in a data set.

After you test run your code, the first thing you should do is check your SAS log. Its necessary to examine your SAS log to ensure your run of the code didn't generate any errors, warnings or uninitialized variables. Its possible that your log may contain warnings, and even some errors which may not indicate there was a problem with the results produced from your code. Nevertheless, a clean log is a desirable goal because it gives others the peace of mind that your code is problem-free.

If there are errors in your code, you need to figure out why the errors and/or warnings were generated. It may be the case that a statement in your code was missing a semi-colon. It also may be the case that you have unbalanced parentheses or quotation marks. Some other typical situations where errors occur happen because variable names got misspelled, or variables were specified in your code which aren't found on a SAS data set. Nevertheless, you need to find the errors and resolve them.

If you code is relatively short, you may be able to scroll through the SAS log to search for errors and warnings. This is a technique which I've used before to examine the log, but I don't recommend it exclusive of other methods.

A best practive technique is to use the FIND function to search for the words ‘ERROR’, and ‘WARNING’ in your code. In Figure 5 is a screenshot of a SAS session and the SAS log, with the search text box used to find the word ‘ERROR’.



```
NOTE: Copyright (c) 2016 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.4 (TS1M7 MBCS3170)
      Licensed to RI DEPT OF HEALTH, Site 70250673.
NOTE: This session is executing on the X64_10HOME  platform.



NOTE: Analytical products:

      SAS/STAT 15.2

NOTE: Additional host information:

 X64_10HOME WIN 10.0.19041  Workstation

NOTE: SAS initialization used:
      real time           5.01 seconds
      cpu time            4.49 seconds

1     Proc Import Datafile='C:\Users\Owner\Documents\S
1   ! Files\Mathematica_Exercise\HOSPITAL_SAEFTY_DAexe
2                    Out=Hosp_Safety
3              DBMS=XLSX
4              REPLACE;
5              SHEET='HOSPITAL_SAFETY';
6              GETNAMES=YES;
7     Run;

NOTE:    Variable Name Change.  HOSP.ID -> HOSP_ID
NOTE: One or more variables were converted because the data type is not supported by the V9
      engine. For more details, run with options MSGLEVEL=I.
NOTE: The import data set has 3263 observations and 18 variables.
NOTE: WORK.HOSP_SAFETY data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           2.18 seconds
      cpu time            1.03 seconds


8
9     Proc Import Datafile='C:\Users\Owner\Documents\SAS
9   ! Files\Mathematica Exercise\HOSPITAL SAEFTY DAexercise.xlsx'
```

Find

Find text: ERROR

☐ Match whole word only

☐ Match case

Direction: Up [✓ Down]

Find Next | Close

WARNING: No occurrences of "ERROR" found.

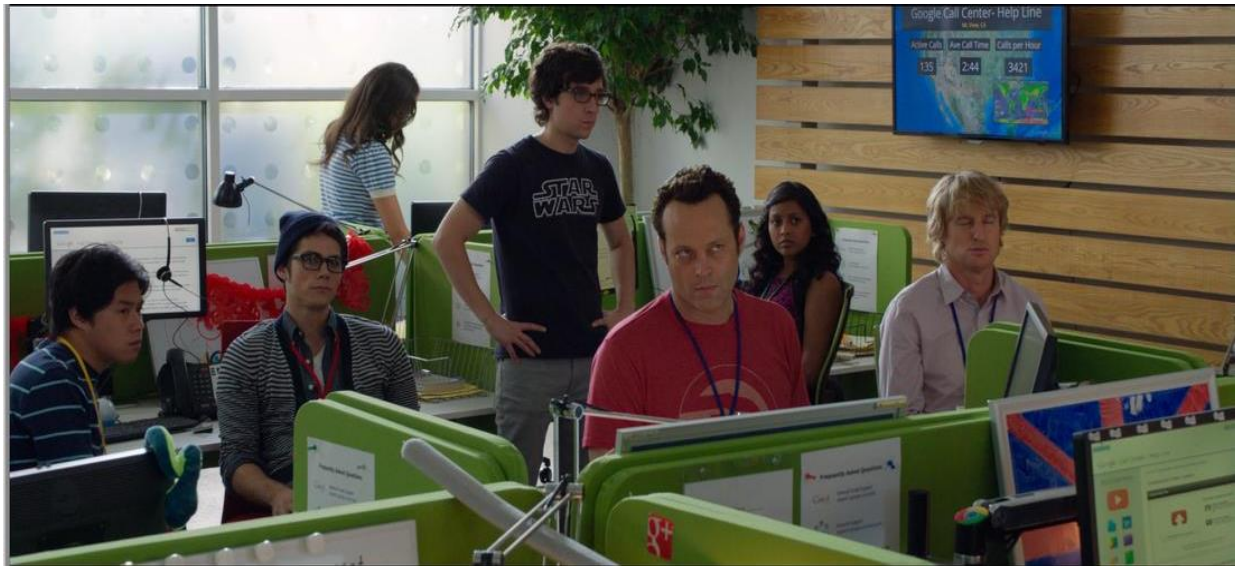
Figure 5. SAS LOG with a search box to find errors.

You can access the FIND function by going to the EDIT menu and scrolling down to ‘FIND’. Alternately, you can pull up the FIND search box by hitting CTRL-F on your keyboard. In the example in Figure 5, SAS is unable to find any occurences of the word, ‘ERROR’ in the log, as evidenced by the message at the bottom, ‘WARNING: No occurrence of ERROR found’.

I recommend searching for three words using the FIND function ‘ERROR’, ‘WARNING’, and ‘UNINITALIZED’. If SAS finds no occurences of any of these words in the log, you can be reasonably sure that your program is problem-free. However, a thorough approach also involves checking and examining SAS output as well as the log. This is necessary to validate the results produced by your code. Checking SAS ouptut helps confirm that any analysis which was produced by your code was performed correctly. Reviewing SAS output also validates the assumptions made in performing the analysis of your data. Saving a copy of your SAS log wIll be covered in the next section.

SAVE YOUR LOG

Once the testing phase of your code has been completed, it’s a good idea to save a copy of your SAS log from the test run. One reason you do this is so that others who might be uncertain of the status of the task or project. For instance, your manager might be wondering whether to trust the data or the SAS output which you provide to them. A copy of the the SAS log demonstrates that your program is valid, and gives confidence that the data produced from your program is valid as well.



Save your log, so that your work can be reviewed later.

The log provides some important information besides code validation information concerning errors, warnings, and the like. The log provides a date-time stamp which provides the date and time the code was executed.   Saving a copy of the log then allows one to refer back to the code to confirm when it was executed.

The SAS log also provides run-time and processing information for each step in your code. For each step in your code, the log provides CPU time and REAL time. These are important metrics to be used to improve processing efficiency and performance tuning of your code.

Processing information is useful during the testing phase of your SAS program. On a project, higher-level managers and internal clients may complain that its taking a long time to produce a deliverable. A copy of the SAS log with timing information will provide an explanation why that's the case.

There are multiple ways to save a copy of the log. The manual way is to use the FILE menu, and choose 'SAVE AS'. However, there are also automated ways to save the log. Using an automated log-saving method prevents you from having to save the log manually every time you run your code. SAS users tend to find the manual method of saving the log to be tedious and cumbersome.

You can route the SAS log to an external file using PROC PRINTTO. PROC PRINTTO can also be used to save SAS output to an external file with an .lst extension. PROC PRINTTO must be placed at the top of your program or before any SAS code which you want routed to external files when executed. The code below is an example of PROC PRINTTO.

```
Proc Printto log="C:\Projects\SAS Files\SAS Logs\ETL_Process1.log";
run;
```

One of the disadvantages of using PROC PRINTTO to save the log is the color-coded log disappears from the LOG window. You no longer have the ability to browse and search the log in the LOG window. The external file containing the log is a text file in black and white. While PROC PRINTTO is an automated approach to log-saving, it is not the best choice when you need to debug your code.

An alternate technique to saving the log is to use display manager or DM commands. These display manager commands work much the same way they do when used from the command line or prompt. When using DM commands within your code, you can chain multiple commands back-to-back in a string. Below is an example of the DM command which saves the log to an external file.

```
    DM ‘Log;File “C:\Projects\SAS Files\SAS Logs\ETL_Process1.log”;
```

With the DM command, you can place it at the end of your SAS program, and your color-coded log in the log window will still be available for browsing and searching.

I recommend using the DM command for log-saving, since the other methods are too tedious, or remove access to the log window. If you submit your SAS program in a batch session, the log and output are automatically routed and stored in external files. However, this is much like the outcome you get using PROC PRINTTO.

MODIFY THE CODE

As a contractor or independent consultant, working with existing SAS code you’re handed on a project, at some point there will probably either be a need or a desire to modify the code. Modifications to the code can be simple and straight forward. Maybe you’ll need to update the path where data is stored in a LIBNAME statement. Then again, maybe you’ll need to update the source data set you’re using in the program.

The changes to the code may also be more extensive and complex. You’re manager might request that you streamline the code. Alternately, maybe you have an idea to restructure the code, although its not part of the specified request.Yet again, It might make sense to convert the code to a macro program and automate it, because it needs to be executed repeatedly.

In some environments I’ve been in, I’ve come across an attitude that’s part of the internal culture of the company that existing SAS code and processes shouldn’t be modified. In these situations, the code is production code and is executed on a regular basis. Thus, the department is depending on the code to produce a deliverable to a client, internal or external. The code works, so there’s no reason to change it. In short, ‘If Its not broke, why fix it.’

The existing code you’re working with might be ‘okay’, meaning it runs free of errors and warnings, or is otherwise clean. This doesn’t mean the code runs well. Maybe it works, but not very efficiently. Its entirely possible the code isn’t readable or formatted well. Just as Vince Vaughn said in ‘The Internship’; ‘Okay isn’t great! Okay isn’t fantastic!’.

Okay Isn’t Great! Okay Isn’t Fantastic!

There could be a whole list of changes to the code which could improve it. The concern that modifying existing prodution processes will cause programs to fail is largely based on fear. An astute programmer knows that any changes made to production code need to be sufficiently tested.

Before any modifications are made, a copy of the production code should be made. The copy of the production code can then be modified and tested. While testing is being performed on the new process, the original production code is kept in place. This way the routine process which the team depends on won’t be upset.

In Figure 6 below is a sample of code which is used to download and import a text file from AWS in the cloud. The code uses PROC S3 to download the file, and then PROC IMPORT to convert it to a SAS data set.

```
FILENAME MEDCOND "C:\SAS\SAS Temporary Files\ medical_cond_tables.txt";

Proc S3 Config="C:\Users\OneDrive-DHHS\DataLayer\Cred\tks3.conf";
      Get "/cdc-corefiles-datalayer-dataanalytics/medcond_tables.txt"
          "C:\SAS\SAS Temporary Files\medcond_tables.txt";
Run;

Proc Import Datafile=MEDCOND
      Out=Medical_Conditions
      Dbms=DLM
      REPLACE;
      Delimiter=TAB;
      Getnames=YES;
      Datarow=2;
Run;

data Medical_Conditions2;
 set Medical_Conditions;
 if Person_ID="IJKLMNOP54231534" then delete;
run;

data Medical_Conditions;
 set Medical_Conditions2;
run;

proc datasets library=work;
 delete Medical_Conditions2;
run;

proc freq data = Medical_Conditions;
 tables file_update_date / nocum nopercent;
run;
```

Figure 16. SAS Code to download and Import Text File.

This code is followed by two data steps which remove records from the data set, and rename it, and PROC DATASETS and PROC FREQ to delete a temporary data set and generate frequencies respectively. The code contains 6 steps in total not counting the FILENAME statement. The two DATA STEPS seem unnecessary given that the only action really being performed is to delete records using an IF-THEN-DELETE statement. The second DATA STEP is only used to rename the data set back to its original name.

In Figure 7 is another sample of code. The code in Figure 17 is the same example which is presented in Figure 6. However, The code in Figure 6 has been modified. The code in Figure 7 contains only 4 steps, 2 steps fewer than the 6 steps contained in Figure 16.

In Figure 7, the two DATA STEPS have been replaced with a single PROC SQL step. PROC SQL permits you to delete or exclude records from a SAS data set without creating a new SAS data set, using the DELETE FROM ‘Table’ statement.

In this example, since we avoid creating another new SAS data set, then the PROC DATASETS step used to delete the data set in Figure 6 is no longer needed.

```
FILENAME MEDCOND  "C:\SAS\SAS Temporary Files\medical_cond_tables.txt";

Proc S3 Config="C:\Users\OneDrive-DHHS\DataLayer\Cred\tks3.conf";
   Get "/cdc-corefiles-datalayer-dataanalytics/medical_cond_tables.txt"
       "C:\SAS\SAS Temporary Files\medical_cond_tables.txt";
Run;

Proc Import Datafile MEDCOND
      Out=Medical_Conditions
      Dbms=DLM
      REPLACE;
      Delimiter=TAB;
      Getnames=YES;
      Datarow=2;
Run;

Proc Sql;
      Delete from Medical_Conditions
      Where Person_ID="IJKLMNOP54231534";
Quit;

Proc Freq data = Medical_Conditions;
      Tables file_update_date / nocum nopercent;
Run;
```

Figure 7. Modified SAS Code to download and Import Text File.

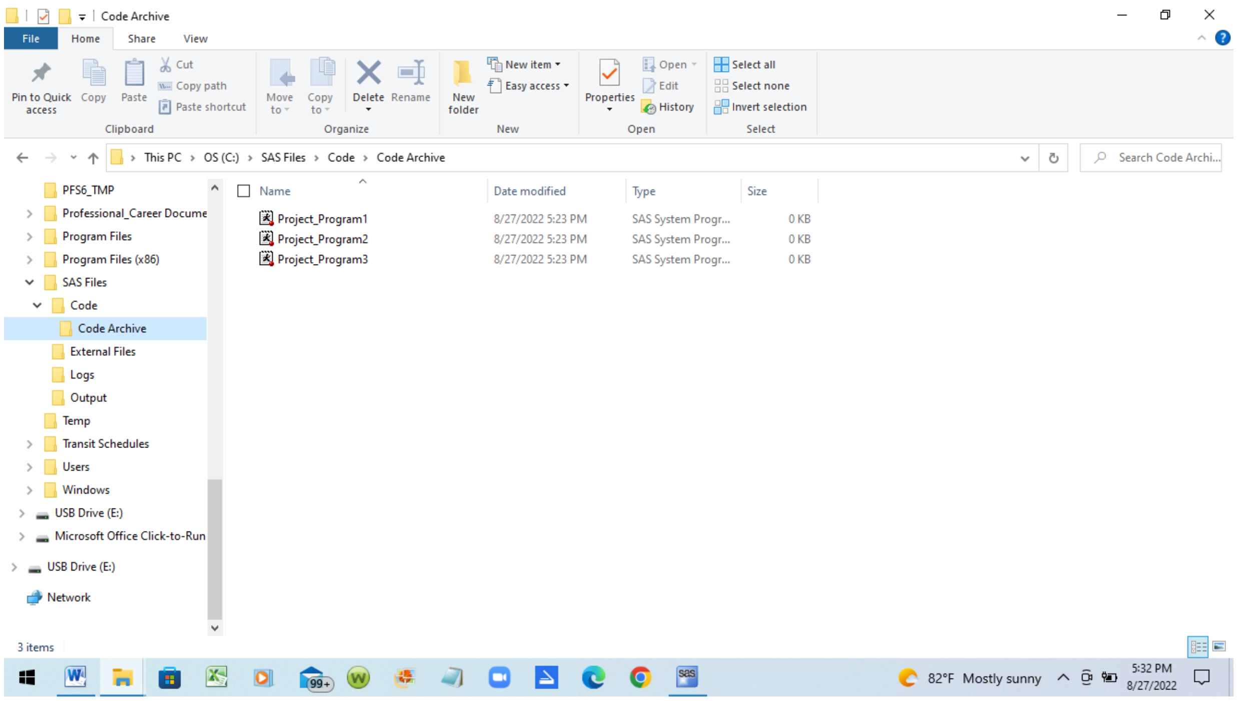
The new code in Figure 7 is more streamlined than the code in Figure 6. With fewer steps, there are fewer passes through the data, which entails less processing or reading and writing of data. Thus, the new code is a more efficient process which consumes less computing resources, such as CPU, and Input/Output.

On another note, the new code contains fewer lines of code. With fewer lines of code, the process is easier to read and understand. Most importantly, the code is easier to maintain. Fewer lines of code mean less code which needs to be updated. By modifying and streamlining the code we have made improvements to it in several respects.

ARCHIVE THE CODE

Eventually code will need to be retired and stored in an archive. There could be a multitude of reasons why code needs to be stored in an archive. The code may be an old process which was replaced by a newer improved process, and the code was taken out of production. Yet again, maybe a new toolset or development language was used to perform the project or task. For instance, SAS code was replaced with one of the open-source languages, such as python. It is also possible that the project itself may have been completed or come to an end.

In archiving the code, it is a good idea to come up with some strategies which make sense. A basic archive entails creating a folder or subdirectory on the server or hard-disk. Specifically, there would be a project folder which is the root directory, and then subfolders created within it for SAS code, output, logs, and external files. The screenshot in Figure 8 shows this kind of basic configuration.



Name	Date modified	Type	Size
Project_Program1	8/27/2022 5:23 PM	SAS System Progr...	0 KB
Project_Program2	8/27/2022 5:23 PM	SAS System Progr...	0 KB
Project_Program3	8/27/2022 5:23 PM	SAS System Progr...	0 KB

Figure 8. SAS code archive directory in Windows Explorer

As Figure 8 shows, within the code subdirectory, the archive would have a separate folder titled ‘Code Archive’. The SAS programs within the archive folder would all be specific to the particular project the project folder in the root directory.

There are other more elaborate configurations for creating code archives. Instead of storing a code archive within a project folder, a code archive directory could be created as the main folder of the root directory. Then a set of project folders could be created within the code archive, detailing specific projects. Alternately, a code archive could be created by compressing SAS code files into a ZIP file.

Code archives don’t necessarily have to be created on a network server. Alternatively, archives can be created and stored on external media, such as external drives, CD-ROMs or even DVDs. Code generally doesn’t take up a considerable amount of space, but it might be worth exploring this option if the code archive gets too large, and takes up substantial storage space.

CONCLUSION

The industry need for SAS consultants and contract SAS developers is ever present as companies with open positions opt to select contract workers to fill their short-term needs. As companies grow and expand, and as employees leave companies, they need SAS computing professionals to come in with short notice to work with legacy SAS code for specific projects. In this paper, I have outlined specific strategies for using existing SAS code written by other programmers. From understanding code to modifying and streamlining code, I've tried to cover specific tasks and aspects of programming a contract programmer will need to perform.

REFERENCES

Letourneau, Kent and Rhoads, Amy (2002) "You CAN Save Your Log and View It, Too: An Improved Process for Automatically Saving the Contents of the Log and Output Windows", in Pharmasug 2002 Conference Proceedings, https://www.lexjansen.com/pharmasug/2002/proceed/Coders/cc10.pdf

Carpenter, Arthur L. (2012) "Doing More with the SAS® Display Manager: From Editor to ViewTable - Options and Tools You Should Know" in SAS Global Forum 2012 Conference Proceedings. http://support.sas.com/resources/papers/proceedings12/151-2012.pdf

IMDB. "The Internship – Photo Gallery". Accessed June 6, 2022. https://www.imdb.com/title/tt2234155/

Rotten Tomatoes. The Internship – Movie Review. Accessed June 6, 2022. https://www.rottentomatoes.com/m/the_internship_2013

Variety. "Film Review – The Internship". Accessed June 6, 2022. https://variety.com/2013/film/reviews/film-review-the-internship-1200491025/

Wikipedia. "*The Internship*". Accessed June 6, 2022. The Internship - Wikipedia

ACKNOWLEDGMENTS

The author would like to thank Matt Becker, PharmaSUG 2024 Operations Chair, Priscilla Gathoni, PharmaSUG 2024 Academic Chair, Srivathsa Ravikiran and Gopal Rajagoral, E-Posters Section Co-Chairs, and the PharmaSUG Executive Committee and Conference Team for accepting my abstract and paper and for organizing this great conference.

CONTACT INFORMATION

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

Jay Iyengar
Data Systems Consultants LLC
datasyscon@gmail.com
https://www.linkedin.com/in/datasysconsult/

Jay Iyengar is Director of Data Systems Consultants LLC. He is a SAS consultant, trainer, and SAS Certified Advanced Programmer. He’s been an invited speaker at several SAS user group conferences (WIILSU, WCSUG, SESUG) and has presented papers and training seminars at SAS Global Forum, Pharmaceutical SAS Users Group (PharmaSUG), and other regional and local SAS User Group conferences (MWSUG, NESUG, WUSS, MISUG). He was co-leader and organizer of the Chicago SAS Users Group (WCSUG) from 2015-19. He received his bachelor's degree from Syracuse University in Public Policy and Economics, and his master's degree from the American University.

TRADEMARK CITATION