Paper 207-2011

Plate of Spaghetti Anyone? Techniques for Learning Existing SAS® Programs

Tricia Aanderud, Independent Consultant, Raleigh, NC, USA

ABSTRACT

When starting a new job or project, a programmer may receive existing programs that need modifications. Often the code's original intent has morphed over the years and many other programmers have made minor changes to it. There may be little documentation to explain the code, the job setup, or even how to run it.

INTRODUCTION

To understand existing programs, you need to learn the environment or structure, learn the code, and learn the data. This paper walks you through the steps for collecting and evaluating the needed information. While aimed at larger jobs, this process can be applied to smaller ones as well. Once you have collected and understood this information, it will be easier to make modifications and even improvements to the program.

LEARN THE STRUCTURE

Prepare yourself to learn the code by first learning about its supporting environment. Use a spreadsheet application to organize your information. The first sheet contains the basics about the program; the second sheet contains information about the environment, and the third sheet contains information about the code. After being completed, this spreadsheet will be your reference guide to the code.

GATHER THE BASIC INFORMATION

You should collect some basic information about the code, such as program name, purpose, and location. The Overview tab is a good location to keep any other general notes that may be valuable later, such as people who may have helpful input along with their contact information. Figure 1 has an example Overview tab for a fictitious customer service job.

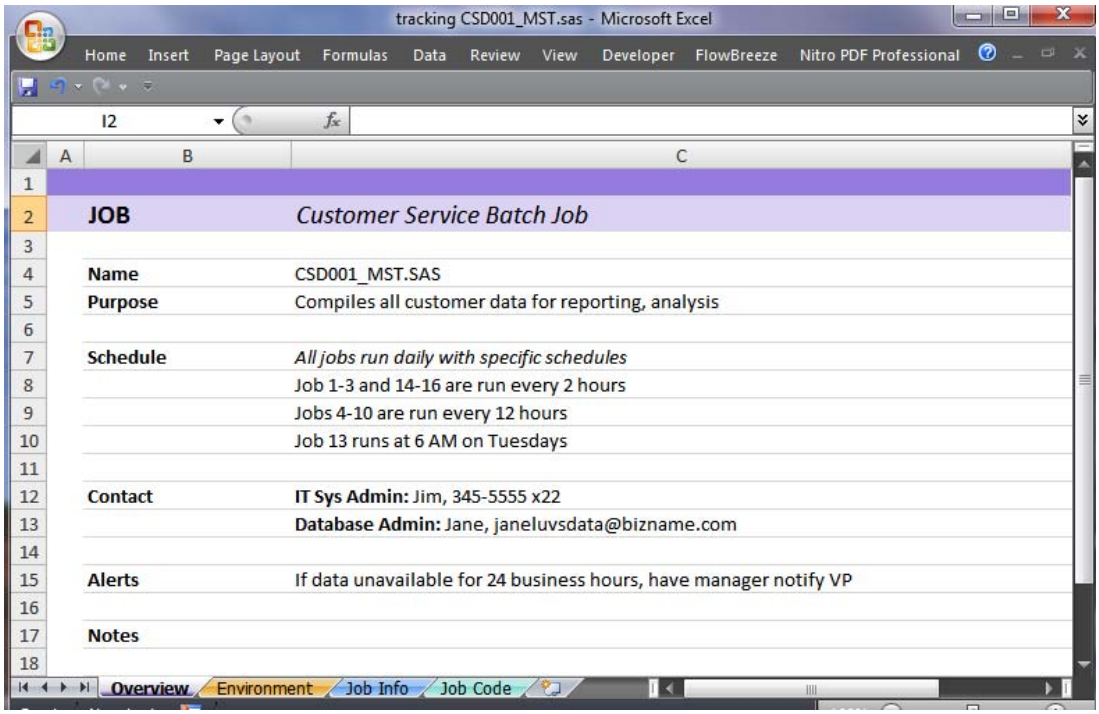
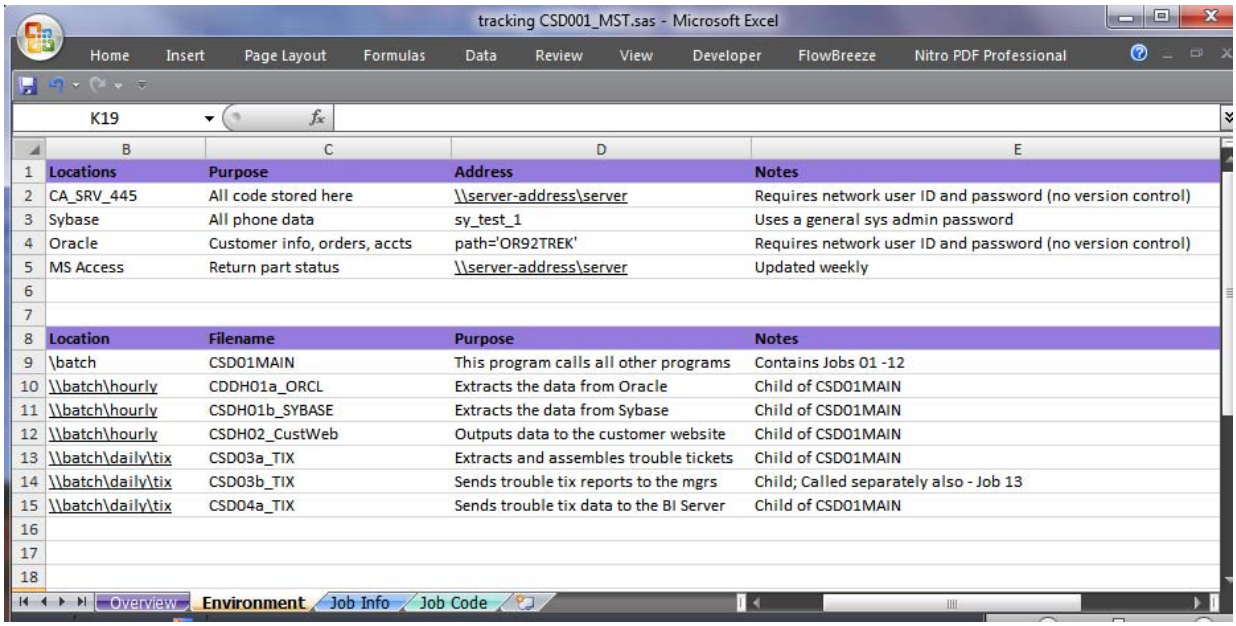


Figure 1. Job Organization - Overview Tab

DOCUMENT THE ENVIRONMENT

For the environment, you need an idea of the job scope, where it runs and where all its pieces are located. This sheet contains detailed information about file locations, supporting file locations, and any information about what is needed to run the job. Eventually you will have this information memorized, but in the beginning a reference helps. Figure 2 shows one way to organize this information. The way you layout this information may vary considerably depending on the job and its scope.



	B	C	D	E
1	**Locations**	**Purpose**	**Address**	**Notes**
2	CA_SRV_445	All code stored here	\\server-address\server	Requires network user ID and password (no version control)
3	Sybase	All phone data	sy_test_1	Uses a general sys admin password
4	Oracle	Customer info, orders, accts	path='OR92TREK'	Requires network user ID and password (no version control)
5	MS Access	Return part status	\\server-address\server	Updated weekly
6				
7				
8	**Location**	**Filename**	**Purpose**	**Notes**
9	\batch	CSD01MAIN	This program calls all other programs	Contains Jobs 01 -12
10	\\batch\hourly	CDDH01a_ORCL	Extracts the data from Oracle	Child of CSD01MAIN
11	\\batch\hourly	CSDH01b_SYBASE	Extracts the data from Sybase	Child of CSD01MAIN
12	\\batch\hourly	CSDH02_CustWeb	Outputs data to the customer website	Child of CSD01MAIN
13	\\batch\daily\tix	CSD03a_TIX	Extracts and assembles trouble tickets	Child of CSD01MAIN
14	\\batch\daily\tix	CSD03b_TIX	Sends trouble tix reports to the mgrs	Child; Called separately also - Job 13
15	\\batch\daily\tix	CSD04a_TIX	Sends trouble tix data to the BI Server	Child of CSD01MAIN
16				
17				
18				

Figure 2. Job Organization - Environment tab

DIAGRAM THE ENVIRONMENT

If there are many elements, having a picture helps you better understand the relationships. You can categorize the environment by data, code, and outputs. The following diagram is a simplified example of how you can layout the information. Keep in mind; this does not have to be a formal drawing. Even a simple hand drawing that you keep near your monitor may be the only quick reference you need.

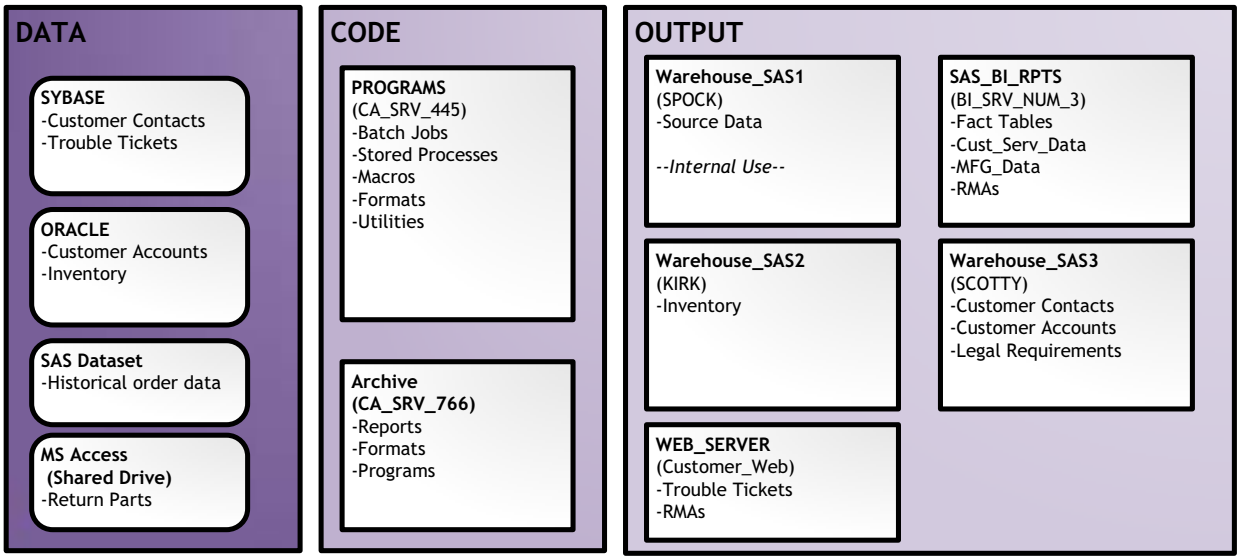


Figure 3. Example of Job Diagram

LEARN THE CODE

After you have the environment, you can start reviewing the code structure to understand how it works. You need to go through the code initially with the goal of gathering some key information about the job itself. Thereafter you will be learning the code flow and structure.

Gathering Key Information

The first time through the code look for information about the job setup and other clues about how the code works. If the code has had many hands helping it along, there is most likely a variety of approaches. During this phase, search for the key information, as suggested in the following table.

Information	Purpose	Suggested Search Words
What libraries are used	Where data is access and stored	*libname*
Sources	How data is accessed or stored	*filename*
Options	Finding a hidden option can save you hours of troubleshooting later	*options, ods*
Macros library	Is there a central macro storage area or some macros used in your job that you cannot modify	*sasautos, mstored*
Included code	It's helpful to know if the code takes a detour.	*%include*
Macro code	Repetitive code created in the job; some programmers depend heavily on macros	*%macro, %mend*
Formats	Is there a central storage area and are you able to make changes	*fmtsearch; proc format; value*

Using SAS to Help

If you treat the code as raw data, you can use SAS to import the code into a dataset. Then, you can parse each line to find keywords related to the code flow. Use ODS to write the output to an XLS file that you can copy into your spreadsheet. The code to import and classify the SAS program is in Appendix A. In Figure 4, you see how the code looks after it is categorized and how color the helps guide your eyes. Now, you can then take advantage of spreadsheet features, such as auto filter and search to learn the code.

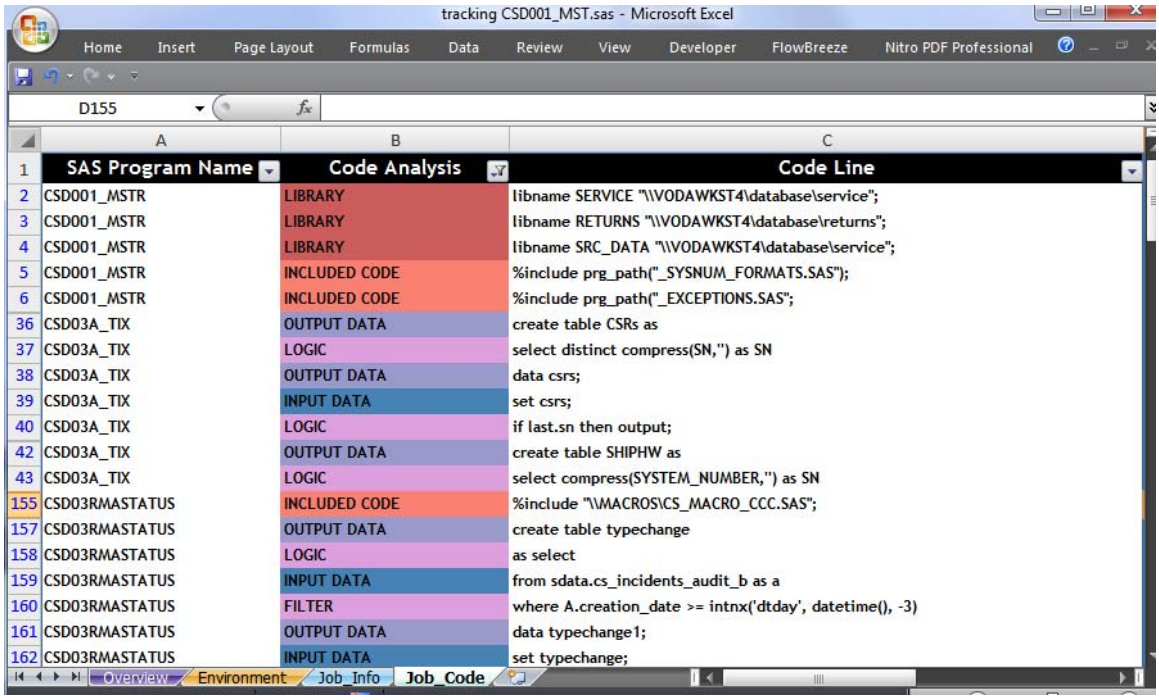


Figure 4. Code after reading into spreadsheet application

You can also use the data to build reports about your code. Figure 5 shows a sample report from the dataset that details the Files, Libraries, and other information you need in your first walkthrough.

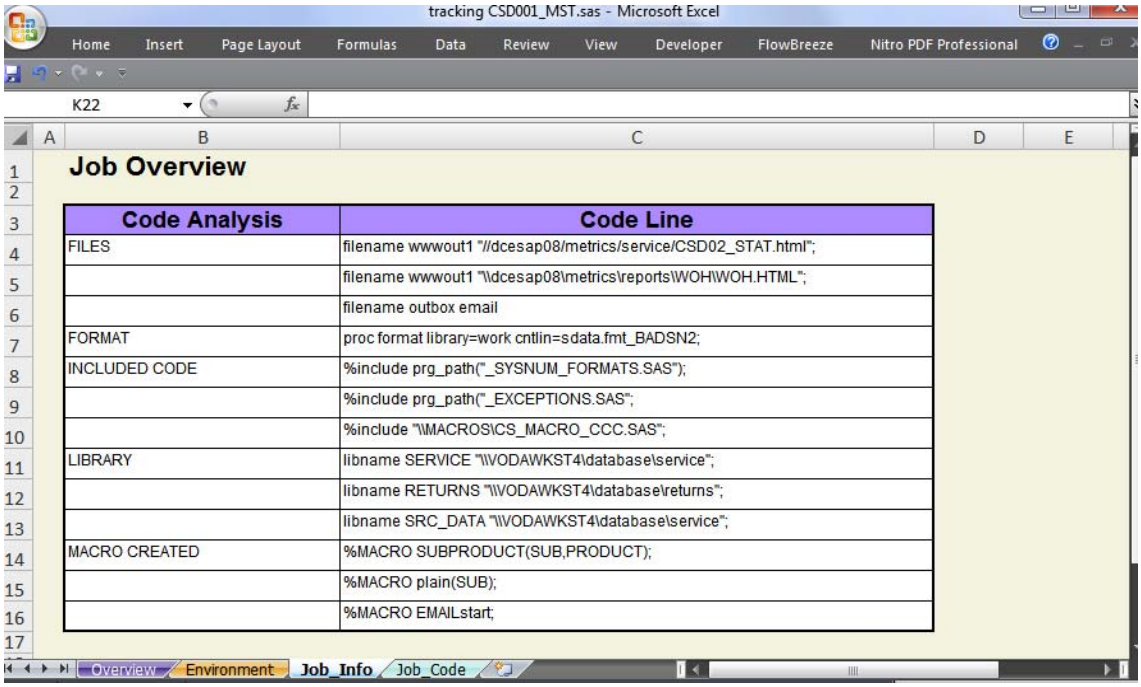


Figure 5. Example of Job Overview Report

Tips for Walking the Code

It's easy to get distracted at this point as the code is probably not written the way you think it should be. Also if the code has survived for several years, it may not follow a logical flow or have a consistent style.

- If you use a spreadsheet for the code analysis, add a column to the right where you can place your notes or your own explanation of what the code is doing
- Start from the end and work your way back through the code. The advantage of starting at the end is that you are less distracted by the coding techniques because you are focused on determining how the output was created.
- Use a separate sheet to make notes of changes you want to implement. You need to prevent yourself from rewriting the code before you have learned it. You can also highlight the code in the spreadsheet so you can return to it later.

Common Items Checklist

As you read the code flow, use the following checklist as a guideline to search for useful information:

- Can you identify a programming style? Did the programmer prefer data steps or PROC SQL?
- Are there comments to explain what is happening and why? Are they accurate?
- Is a lot of macro coding used or maybe none at all?
- What datasets are used as source and which are created as a result? What datasets are used repeatedly? Which datasets are kept? Can you determine why some are saved over others?
- What variables do you see most often?
- Can you determine any unique variables (look for the BY statement in PROC SORT)?
- Can you determine the requirements based on the logic (IF/THEN/ELSE statements)?

LEARN THE DATA

After you have determined the code flow, learning the data is the next step. Using the code as a reference, it's useful to determine which variables are used most often. If you review the BY statements and Join statements you determine the unique variables. Otherwise, scanning the code you can begin to divide the variables into three groups, as follows:

- Primary - used often in the program
- Secondary - used less often but referenced
- Tertiary – rarely or never used

Taking a Data Deep Dive

When you need help understanding the datasets size and variables, use the SAS dictionary views. The following example shows a way to use a dictionary to determine the dataset size, variables, and change dates. This code uses the SASUSER library and some common datasets to help you understand how it works. In Figure 6, the code result is shown. The CARS dataset has 428 observations and a total of 15 variables: 5 character and 10 numeric.

```
proc sql;
select libname, memname, crdate, modate, nlobs, nvar, num_character, num_numeric
    from sashelp.vtable
    where libname = UPCASE('SASHELP');
quit;
```

Library Name	Member Name	Date Created	Date Modified	Number of Logical Observations	Number of Variables	Number of Character Variables	Number of Numeric Variables
SASHELP	CARS	12NOV08:22:51:38	12NOV08:22:51:38	428	15	5	10
SASHELP	PRICEDATA	12NOV08:22:34:38	12NOV08:22:34:38	1020	28	3	25
SASHELP	SNACKS	12NOV08:22:34:38	12NOV08:22:34:38	35770	6	1	5
SASHELP	TOURISM	26MAR10:02:37:37	26MAR10:02:37:37	29	8	0	8

Figure 6. Data Dictionary Example

Down to the Data Details

After you have listed the main datasets and determined the primary variables, you are ready to learn what the data contains. The basic SAS procedures can help with this task. PROC MEANS can help you discover the data population and PROC FREQ can you help you understand the common values in variables. The following provides examples using data from the SASHELP library.

Analysis with Proc Means

Using the _numeric_ variable reference, you can review all the numeric dataset variables at once with PROC MEANS. You can quickly determine any missing values.

In this example, Cylinders has 2 records missing. This could affect your results, so it is worth further investigation.

Code Example

```
proc means data=sashelp.cars
           n nmiss;
var _numeric_;
run;
```

Variable	Label	N	N Miss
MSRP		428	0
Invoice		428	0
EngineSize	Engine Size (L)	428	0
Cylinders		426	2
Horsepower		428	0
MPG_City	MPG (City)	428	0
MPG_Highway	MPG (Highway)	428	0
Weight	Weight (LBS)	428	0
Wheelbase	Wheelbase (IN)	428	0
Length	Length (IN)	428	0

Analysis with Proc Freq

Using the same dataset, only this time use PROC FREQ and the _character_ variable reference to see all character variables at once.

In this example, you can see the values for Origin and DriveTrain. You learn that the Origin is evenly spread but not all continents are represented. Also DriveTrain has front wheel drive the majority of the models. When reviewing the code, you can understand what the filters and logic are doing with these variables.

Code Example

```
proc freq data=sasuser.cars;
table _character_ /missing;
run;
```

Origin	Frequency	Pe	Cumulative	Cumulative Percent
Asia	158			36.92
Europe	123			65.65
USA	147	34.35	428	100.00

Are the other continents relevant?

DriveTrain	Frequency	Percent	Cumulative Frequency	Cumulative Percent
All	92	21.50	92	21.50
Front	226	52.80	318	74.30
Rear	110	25.70	428	100.00

READ THE LOGS

A final step would be to review the job logs. You may read the code without realizing that it is riddled with errors and warnings that are preventing it from processing correctly. If you are changing the code, you may find additional maintenance is needed before you can make any improvements. There are several past papers that contain techniques and code to help you auto check log files.

CONCLUSION

Most likely, you are being asked to learn the code so you can make changes or correct any issues. The quicker you are able to understand the environment, code, and datasets, the quicker you will be able to get to the real work.

REFERENCES

- Cody, R. 2007. Learning SAS by Example: A Programmer's Guide. Cary, NC. SAS Press.
- Cody, R. Paper 57-27. Data Cleaning 101. SUGI 27. Available at http://www2.sas.com/proceedings/sugi27/p057-27.pdf.
- Dilorio, F and Abolafia, J. Paper 237-29. Dictionary Tables and Views: Essential Tools for Serious Applications, SUGI 29. Available at http://www2.sas.com/proceedings/sugi29/237-29.pdf.
- Droogendyk, H and Fecht, M. Paper 106-31. SAS® to Publishable Excel … Seamlessly – Using ODS, XML, and Other Tricks, SUGI 31. Available at http://www2.sas.com/proceedings/sugi31/106-31.pdf.
- Hadden, L. Paper 142-31. Advanced PROC REPORT: Traffic Lighting - Controlling Cell Attributes With Your Data, SUGI 31. Available at http://www2.sas.com/proceedings/sugi31/142-31.pdf.
- Mitchell, R. Paper 63-27. Fast and Easy Ways to Annoy a Statistician. SUGI 27. Available at http://www2.sas.com/proceedings/sugi27/p256-27.pdf.
- Stojanovic, M. Paper CC-037. SAS ® Log Summarizer – Finding What's Most Important in the SAS ® Logs. SESUG 2008. Available at http://analytics.ncsu.edu/sesug/2008/CC-037.pdf.
- Winn, Jr., T. Paper 258-29. Guidelines for Coding of SAS© Programs. SUGI 29. Available at http://www2.sas.com/proceedings/sugi29/258-29.pdf.

ACKNOWLEDGMENTS

Thanks to Jon Barry, Robert Bishop, Angela Hall, Jennifer Johnston, Tom Mabie, and Dipesh Patel for discussion of thoughts, running sample code, proofreading the paper, and also just encouraging me to write this paper.

RECOMMENDED READING

These books are not specifically about SAS, but the ideas can be applied to your code and verification practices.

- Feathers, Michael. 2004. Working Effectively with Legacy Code. New Jersey. Prentice Hall.
- Martin, Robert. 2008. Clean Code, A Handbook of Agile Software Craftsmanship. New Jersey. Prentice Hall.
- McConnell, Steve. 2004. Code Complete: A Practical Handbook of Software Construction. Bellevue, WA. Microsoft Press.

CONTACT INFORMATION

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

Tricia Aanderud

tricia.aanderud@gmail.com

APPENDIX A. CODE SAMPLE

```
/*=========================================================================*/
/* Purpose: Imports and categorizes code                                   */
/* Inputs:  SAS files                                                      */
/* Outputs: SAS Dataset, Excel files                                       */
/* Tested with SAS 9.1.3 and 9.2                                           */
/*=========================================================================*/
/*Usage Notes:                                                             */
/*  This file is setup to run in a Windows-based directory structure      */
/*                                                                         */
/*   To use the code as is, do the following:                              */
/*   1 - Create this following directories from the root directory level:   */
/*          C:\temp\code                                                   */
/*          C:\temp\code\output                                            */
/*   2 - Copy your code to the c:\temp\code subdirectory                   */
/*                                                                         */
/*   Note: To use alternate paths, modify the filename and output_path vars */
/*=========================================================================*/

options error=1 nocenter;

/*Load all files in the directory into one dataset*/
filename inCode "c:\temp\code\*.sas";

/*Path for the EXCEL output                      */
%let output_path=c:\temp\code\output\;

/*Optional: Storage area for the code as a dataset */
libname sdata "&OUTPUT_PATH";

/*_________________________________________________________________________*/
/*Loads each file, captures filename, & creates a dataset                  */

data sdata.CODE (drop=newline);
attrib Type                format=$15.        label='Code Analysis'
       Number              format=8.          label='Line Number'
       code_name           format=$50.        label='SAS Program Name'
       code_line                              label='Code Line'
       filename    length=$256
;
  retain code_name;
```

```
infile inCode filename=filename eov=eov length=L;
input Code_Line $varying256. l;

/*===Capture the name of SAS program name if more than one program  */
if _n_ eq 1 or eov then do;
   code_name = scan(upcase(filename),-2,'\.');
   eov  = 0;
end;

/*Helps preserve line order during a SORT */
  NUMBER=_N_;

/*Compress line to remove all white space, tabs; set line to uppercase*/
  NewLine=upcase(strip(compress(code_line,"'009'x")));

/*Note: Assignment order for TYPE is purposeful                     */
/*Re arrange the order or make additions based on your situation    */

if index(newline, '/*')   gt 0
    or index(newline, '*/') gt 0
    or substr(newline,1,1) = '*'                  then TYPE='COMMENT';

  else if index(newline, '%INCLUDE') gt 0         then TYPE='INCLUDED CODE';

  else if index(newline, 'TITLE') gt 0
       or index(newline, 'FOOT') GT 0
       or index(newline, 'STYLE') GT 0            then TYPE='DISPLAY';

  else if index(newline, 'PROC FORMAT') gt 0
       or index(newline, 'VALUE') GT 0            then TYPE='FORMAT';

  else if indexw(newline, 'PROC') gt 0            then TYPE='PROCEDURE';

  else if indexw(newline, 'DATA')         gt 0
       or index(newline,  'CREATE TABLE') gt 0
       or indexw(newline, 'OUT=')         gt 0    then TYPE='OUTPUT DATA';

  else if indexw(newline, 'SET')    gt 0
       or indexw(newline, 'FROM')   gt 0
       or indexw(newline, 'JOIN')   gt 0
       or indexw(newline, 'MERGE')  gt 0
       or indexw(newline, 'UPDATE') gt 0
       or  index(newline, 'DATA=')  gt 0          then TYPE='INPUT DATA';

  else if index(newline, 'LIBNAME') gt 0          then TYPE='LIBRARY';

  else if index(newline, 'FILENAME') gt 0
       or indexw(newline, 'FILE=')     gt 0       then TYPE='FILES';

  else if index(newline, 'IF')       gt 0
       or index(newline, 'ELSE')     gt 0
       or index(newline, 'CASE')     gt 0
       or index(newline, 'WHEN')     gt 0
       or index(newline, 'SELECT')   gt 0
       or index(newline, 'DO')       gt 0         then TYPE='LOGIC';

  else if indexw(newline, 'WHERE') gt 0
       or indexw(newline, 'AND')   gt 0
       or indexw(newline, 'OR')    gt 0           then TYPE='FILTER';

  else if index(newline, '&') gt 0                then TYPE='MACRO VARIABLE';
  else if index(newline, '%MACRO') gt 0           then TYPE='MACRO CREATED';
```

```
    else if index(newline, '%') gt 0                then TYPE='MACRO USED';
/****** Any custom code can go here    */
                                                    else TYPE='OTHER';

/*Only output lines with data*/
 if length(newline) gt 1 then output;

run;

/*_____________________________________________________________________________*/
/*Color code the lines to assist with learning                                 */
/* You can change the hex codes for your preferred colors                      */
/*Reference for hex code colors: http://www.neopets.com/~triflot               */

%LET GRAY    = #D3D3D3;     %LET RED     = #CD5C5C;     %LET PINK  = #FFE4E1;
%LET ORANGE  = #FA8072;     %LET GREEN   = #808000;     %LET YELLOW  = #FFFFCC;
%LET LTBLUE  = #9999CC;     %LET DKBLUE  = #4682B4;     %LET PURPLE  = #DDA0DD;

proc format;
   value  $colorkeywords
   'COMMENT'        = &GRAY      'OTHER'   = &GRAY       'DISPLAY'  = &PINK
   'PROCEDURE'      = &GREEN     'MACRO'   = &YELLOW     'LIBRARY'  = &RED
   'INCLUDED CODE'  = &ORANGE    'LOGIC'   = &PURPLE     'FILTER'   = &PURPLE
   'OUTPUT DATA'    = &LTBLUE    'FORMAT' = &RED         'FILES'    = &RED
   'INPUT DATA'     = &DKBLUE
   ;
run;

/*_____________________________________________________________________________*/
/*Create Spreadsheets                                                          */

ods listing close;

ods html file="&OUTPUT_PATH.Job_Info.xls"  style=statistical;

title h=4 "Job Overview: Library";
proc report data=sdata.code nowd

       style(header)=[background=#AD8BFE font_size=3]
       style(report)=[background=white];

where type in ('LIBRARY', 'FILES', 'INCLUDED CODE')
       or type contains ('MACRO CREATED')
       or type contains ('FORMAT');

column type code_line;
define type/group;
define code_line/width=200;
run;

ods html close;

ods html file="&OUTPUT_PATH.Job_Code.xls" style=seaside;

title;
proc report data=sdata.code nowd
     style(header)=[background=black foreground=white font_size=3];

where type not in ('COMMENT' 'OTHER');   /*Remove this line to see comments */

column code_name type code_line;
define code_line/width=200;
```

```
compute type;
     call define (_COL_, 'style', 'STYLE=[background=$colorkeywords.]');
endcomp;

run;

ods html close;
ods listing;

/*_____________________________________________________________________________*/
/* END OF CODE ================================================================*/
```