Maintaining Inherited SAS Code

Clarence Wm. Jackson, CSQA

Abstract

SAS has been around awhile and has worked so well that programs are still running at companies while the person who wrote the program code (the code) has left the company. When the code requires attention because of changes, ABENDS, or other reasons, then someone not familiar with the code must be able to quickly make the changes. Thoughtful people will usually want to understand the code before making changes, but getting an understanding requires some time for reviewing the code. Applying a process to evaluate the code can reduce the time needed to understand the code before making changes. This paper introduces a checklist process that when applied, reduces the time needed to understand unfamiliar and undocumented code. The checklist can further be used as documentation of the code for later reference.

Introduction

A program is written to provide an IT solution to a business problem or specific requirement. When the program was originally written, there was a specific purpose intended in support of the business. Changes in a program then should address changes in the solution needed to solve the changed business problem.

Chances are that every SAS programmer will encounter a situation where a change to SAS code is needed to programs not written by the programmer. In those cases, the person needing to make changes will need to at least review the code prior to making changes. In most cases, the SAS code to be modified has been in use for many run cycles without problems, and the person that wrote it may be long gone from the organization. In many cases, the code may not be documented.

For experienced SAS programmers, finding out what the code is really doing may take only a reading of the code and making a few notes. Reading the code is only part of the process. An understanding of what is going on in the code is needed before making changes. A change made and not understood will be a problem later.

A process of maintaining inherited code is presented here by use of a checklist approach to documenting the program to identify where in the code changes are needed. Simply put, this paper will define a way to get to know SAS code written by others, and now must be maintained by you. You would need to know the code before making modifications, addressing run errors, and other issues that come up in changing environments. Making changes without a few key pieces of info can break the code or cause unintended results.

Maintaining Inherited SAS Code

Clarence Wm. Jackson, CSQA

Maintaining SAS code is a process that can be controlled with a checklist. The process will define what information is included on the checklist such as the location of the code module, the purpose and other details that assist in documenting the program.

Maintenance, Enhancements, and Changes

For the purpose of this paper, the terms “maintenance, changes, and enhancements” are used interchangeably, but are indeed different actions.

To “maintain” is keep in working order as it currently exist. “Maintenance” then is the process of keeping something working as is. Maintenance items include things such as backing up code and data, updating because of operational or environmental issues. Maintaining does not require changes to source code logic or processing of data.

“Enhancements” are modifications that improve the solution. Enhancements usually requires changes to source code logic such as selection or extraction criteria, input sources, output and report requirements, functionality, and other changes that modify the IT solution.

“Changes” are any material modifications to source code. Changes can be either maintenance or enhancements.

The process of changing SAS code is the same as that used to change any programming software code. The COBOL, ALC, JCL, and other ‘legacy’ programming languages have benefited from having to exist in strict production environments with regards to maintenance and change control.

The Process of Maintaining SAS Code

The common steps when faced with making changes to an unfamiliar program are to:

1. Print a listing of the code, or get the last run log
2. Read and make notes on the listing
3. Note where changes will be made in the code.
4. Make the changes
5. Test the code for syntax errors and correct if needed
6. Test the code against data and adjust as needed
7. Return the code to scheduled run mode
8. Monitor at least two runs of the program

Maintaining Inherited SAS Code

Clarence Wm. Jackson, CSQA

While this is a common approach, the effort needed to make the changes is often duplicated when the code needs to be modified again, and the steps need to be repeated. The information about the program that is gained can be forgotten if not documented. To compensate for documentation, sometimes the marked up listing is saved for later reference. This is OK for small 1-2 page listings, but for larger listings and multiple programs, this can be inefficient.

One way to document the information learned is to develop a standard document that can provide key info about the program to speed the process. The process for making changes to SAS programs (or any other) includes knowing:

1. The purpose of the program.
2. The location of the program in the IT environment.
    a. In mainframe JCL, the source code location can be a PDS or in-stream as noted in the "SYSIN DD" within the EXEC SAS step
    b. In distributed links, icons, and batch files, the source code location is the "*.sas" filename
3. The input data/files, including record layouts or data elements used in the program.
    a. This varies by site, but can be identified by INFILE, FILENAME, and FILE references within the SAS code.
4. The program logic and internal processing within DATA and PROC steps that manipulate data.
5. The output reports, files and data.

Whenever a program is to be changed, the essential items needed are:

1. What changes are needed and where in the program to make them?
    a. Changes in input data such as:
        i. New record/data element definition changes to current data sources.
        ii. Input data from new data sources.
    b. Changes in data element usage and handling.
    c. Changes in output data.
    d. Reports changes, new reports.
    e. Eliminating functions and features no longer needed.
    f. Adding new functions and features.
2. What is the impact on the program to be changed?
3. Difficulty of making the change?
4. Time need to implement the change?
5. Is the change permanent or temporary?
    a. Temporary changes should be addressed by creating another program or module.

Maintaining Inherited SAS Code

Clarence Wm. Jackson, CSQA

Use A Checklist To Document Changes

One way to document the code is to use a checklist. Checklists are used as a quality control tool to ensure that items essential to a process is covered. Checklists are dynamic within the organization, and should include items specific to a process. In other words, checklists should be created that meets the needs of the people using them.

An example use of the checklist is applied to the program below in figure 1. The changes requested are to add processing for handling overpayments. Overpayments will have a balance of less than zero. As the program is currently designed, overpayments will not be properly accounted for, and will show up in the 'BAD' group as noted in the output report from PROC FREQ (figure 2).

```
* CHEKDATA reviews input and ensures that the
records are properly coded with the correct status
for the balance due.;

DATA CHEKDATA;
  INFILE "//DATA/ACCT/PAYMENTS.TXT";
  INPUT  STATUS $ 1.
         BALANCE 5.2;
  IF BALANCE = 0 THEN
         BAL='0 $ DUE';
    ELSE BAL='$$ DUE';
RUN;

PROC FORMAT;
         VALUE $ST
                  'F'='PAID'
                  'D'='DUE'
         OTHER='BAD';
RUN;

PROC FREQ;
         FORMAT STATUS $ST.;
         TABLE STATUS*BAL;
RUN;
```

Figure 1

The FREQ Procedure
Table of status by BAL

status	BAL	
	Frequency|	
	Percent |	
	Row Pct |	
Col Pct |	$$ due	0 $ due | Total
DUE	1	0 | 1
PAID	1	1 | 0
	33.33	0.00 |
BAD	1	1 | 0
Total	2	1 3
	66.66	33.33

Figure 2

A quick reading of the code shows that the DATA step sets the BAL flag based on the balance due, and the PROC FORMAT step creates formats for the status from the cards read in. These will be the areas for changes.

The first change will be in the DATA step. Logic in the form of an additional ELSE IF statement is added to account for balances being less than zero, indicating an overpayment.

Also, the status will need to be recoded for records with overpayments. Current business rules require that the status be either 'F' or 'D'. An additional status of 'O' has been added to the business rules. The program will be required to recode the status based on the balance in this case.

The second change will be in the PROC FORMAT step. An addition VALUE is added to $ST to format the new 'O' status.

After noting the areas of changes before making the changes, a checklist is used to document the proposed changes. Beginning with the DATA step, the current logic and changes are documented. The documentation continues for all areas that are of concern regarding the program.

The headings on the checklist are:

- Program/module name
- Item
- Description
- Current
- Changed To

The checklist fields that are used are any items that is of importance to maintaining the program. The list does not need to match the example, but should be adjusted for organization. Consultants may want different info about the programs than internal staff. For the example, info noted in the example:

1. Program/module location
2. SAS version
3. Program purpose and business rules
4. Input source(s) and important date elements
5. Output Report(s)
6. Output data/files
7. General processing logic and organization
8. Number of DATA steps
9. MACRO coding used?
10. Run conditions
11. Selection criteria
12. User FORMATs/INFORMATs
13. Error handling procedures

After making the changes to the code as noted in figure 3, the program is now ready for testing and execution. A review of the output report (figure 4) displays the correct value of 'overpaid' instead of 'bad', which is the correct result.

```
* CHEKDATA reviews input and ensures that the
records are properly coded with the correct status
for the balance due.;

DATA CHEKDATA;
  INFILE  "//DATA/ACCT/PAYMENTS.TXT";
  INPUT  STATUS $ 1.
         BALANCE 5.2;
  IF BALANCE = 0 THEN
         BAL='0 $ DUE';
   ELSE IF BALANCE  LT 0
         THEN DO;
                BAL='OVER PAID';
                STATUS='O';
         END;
  ELSE BAL='$$ DUE';
CARDS;
F 0.00
D 3.00
F -3.00
;

PROC FORMAT;
         VALUE $ST
                'F'='PAID'
                'D'='DUE'
                'O'='OVER PAID'
         OTHER='BAD';
RUN;

PROC FREQ;
         FORMAT STATUS $ST.;
         TABLE STATUS*BAL;
RUN;
```

Figure 3

The FREQ Procedure
Table of status by BAL

status	BAL		
Frequency |			
Percent |			
Row Pct |			
Col Pct |	$$ due	0 $ due	Total
DUE	1	0	1
PAID	1	1	0
OVERPAID	1	1	0
Total	2	1	3
	66.66	33.33	

Figure 4

An example checklist that documents the changes made to the code is included at the end of this paper.

The Value of the Checklist

The value of using a checklist isn't initially realized, as it could appear to add to the process of making changes. However, the benefits are many. Documentation now exists regarding info about the program for later reference. If a need for the program to be changed later, we have a ready reference of what the program does, major items within the program, and what was last updated. So the next time the code needs updating, the checklist can be used as a reference, since it has the info needed that is usually gained from reading and analyzing the code. Areas for

changes are understood sooner, and changes can be made and tracked. The amount of time needed to make changes is reduced, as well as providing estimates of the work needed improved. The more programs, the better the value, as the checklist can be used as an inventory of the programs in use.

Conclusion

The need to make changes to programs is always present in a vibrate organization. Changes in business rules, laws, regulations, data, and other sources for change require fast and accurate updates to programs. The checklist should be modified for the shop in which it is used, with items of importance to the programmers within the organization in which it is used. Hopefully, the use of checklists to document programs will assist in the effort to maintain inherited SAS code, as well as provide documentation for new code.

Maintaining Inherited SAS Code

Clarence Wm. Jackson, CSQA

Example Change Review Checklist for 'CHEKDATA'

#	Item	Description	Current	Changed To
1.	Program/module location	LAN	//programs/sasjobs/acct/chekdata.sas	
2.	SAS version	Version 8.1		
3.	Program purpose and business rules	Checks the balances and status of accounts in accordance with current business rules.	If the balance is 0 then the status should be 'F' for fully paid. If the balance is greater than 0 then the status should be 'D' for balance due.	Added rule to check for over payments if the balance is less than 0 then the status should be 'O' for over paid.
4.	Input source(s) and important date elelements	CARDS from data entry process contains 2 fields	//DATA/ACCT/PAYMENTS.TXT" Status $1., Balance 5.2	
5.	Output Report(s)	PROC FREQ at the end of the program. Frequency and counts of records by status and balance	TABLE STATUS*BAL;	
6.	Output data/files	None created		
7.	General processing logic and organization	Read CARDS. Compare business rules for balances to data. Print frequency of status to balance due report.	If Balance = 0 then BAL='0 $ due'; else BAL='$$ due';	Added "else if Balance LT 0 then do; BAL='Over Paid'; STATUS='O'; END" before last 'else' statement.
8.	Number of DATA steps	1 at the start of the program	DATA CHEKDATA	
9.	MACRO coding used?	Not used		
10	Run conditions	Update cards with current balances and status		
11	Selection criteria	All records processed		
12	User FORMATs/INFORMATs	$ST used for the status code	'F'='PAID', 'D'='DUE', OTHER='BAD'	Added 'O'='OVER PAID'
13	Error handling procedures	1. Check CARDS for input 2. Rerun job		

Maintaining Inherited SAS Code

Clarence Wm. Jackson, CSQA

Contact Info

Clarence Wm. Jackson, CSQA
Manager, Business Risk and Quality Assurance
City of Dallas, Communication and Information Services
1500 Marilla St, 4DS
Clarence.Jackson@DallasCityHall.com
CJac@compuserve.com (home)

This paper will be posted to my personal web site at http://ourworld.compuserve.com/homepages/CJac/list.htm after the conclusion of the SCSUG 2005 conference.