Paper 79-2010

SAS® Macro Application Using Microsoft Excel As Both Control File and Process Documentation

Mike Tangedal, Minneapolis MN

Abstract

Any SAS production process executed repeatedly can be converted using SAS Macro language to reduce redundancy. If this process is run using macro parameters which do not vary significantly each time then use of a control file may be warranted. Utilizing the Excel spreadsheet format as a control file provides ease of use in maintenance, documentation, and transferability. A time stamp of some sort need be added to each control file entry and another can be generated to note when the SAS program was last run. Upon completion, the log from the SAS job can also be stored in a tab within this control file worksheet as a monitoring device. Further quality assurance can be obtained through implementing basic tests on source files read with each iteration. Results from these tests can also be stored on a separate tab within the control file worksheet. The control files themselves can be organized in a relational database structure. The SAS macro language tips provided are derived mostly from world-class SAS programmer Art Carpenter.

Introduction

SAS© programs often need to run repeatedly – many on a timed routine such as monthly. Some SAS programs not run every month still require repeated iterations before completion. The SAS macro language assists greatly with this task allowing the user to change only the pertinent parameters before the SAS job is run again. With increased code sophistication, the parameters to the SAS macro may stem from a variety of sources, including graphical or dynamic interfaces. However, upon subsequent runs of the base SAS program, these parameters to the SAS macro are overwritten and discarded as the process continues on.

"Control File" or "Control Data Set" is a term in the SAS vernacular popularized by Art Carpenter in many of his books and papers. This is a meta-data concept referring to the storage of parameters to a SAS macro stored in an external file or data set. This practice is warranted whenever benefit can be derived from storage of these parameters in a repeated practice or the number of parameters or macro calls is so large as to warrant documentation outside of the SAS process. For example, for a process which is to be run monthly requiring a different source file each time, here might be the list of parameters submitted to a SAS macro over time.

Year	Month	Input File	QA? (Y/N)	Report? (Y/N)
2010	5	/datawarehouse/source/May2010	Y	Y
2010	6	/datawarehouse/source/June2010	N	Y
2010	7	/datawarehouse/source/July2010	Y	Y

The metadata above can obviously be stored in all manner of formats, including a SAS data set, text file, or Microsoft© Excel spreadsheet. The main advantage of utilizing a spreadsheet file is that this format is now the standardized method of storing accessible columnar information. Another large advantage is this format allows for easy interaction outside of the realm of the programmer. All manner of documentation can be added to the list above which can used to fully explain the process to all interested parties. This documentation is another great advantage. A control file stored as an Excel spreadsheet can function as both storage of metadata for the SAS program as well as documentation and verification of the entire process. All parties involved in this process can access this spreadsheet file to assess the current state of the project.

Spreadsheet files can of course do more than store simple metadata. Through the use of Proc Import, DDE or Proc Export, information within Excel spreadsheets is easily converted to variables within data sets and back again. Both the SAS log as well as reports/results of each run of the monthly program can be saved as either separate spreadsheet files or tabs within the same worksheet. .

Control File Conversion into Macro Variables

The first step to getting control file information stored in an Excel spreadsheet into SAS macro variables used as parameters in a monthly production process is to convert the tabs in the worksheet file to SAS data step variables. Thankfully this process has become much more simple in recent versions of SAS using Proc Import. For example, to convert the noted list to a SAS dataset would involve code such as the following:

```
PROC IMPORT OUT=MonthlyControl
       DATAFILE= "/datawarehouse/controlfile/control1.xls"
       DBMS=Excel REPLACE;
       GETNAMES=YES;
       SHEET=”sheet1”;
RUN;
```

This is of course a generic example. Utilization of this process within a structured data warehouse would warrant both a more formal naming convention as well as substitution of this code for a “Read Excel Tab” macro. This overall process would likely serve to read all metadata information stored in all pertinent spreadsheet files as the cost of such as process in both term of memory requirements and run time would be minimal.

Once the metadata is available as SAS data set information, it is converted to SAS macro variables through the widely-used practice of creating pseudo-macro variable arrays. Prior to creating such arrays, selection criteria are applied to the metadata data set to retain the one record of note for the current production run. This is demonstrated in the following SAS macro.

```
%MACRO LATEST(METADATA=,MON=,YR=);
       /* METADATA - data set name containing metadata */
       /* MON - Source file month.  Default is latest value from metadata */
       /* YR  - Source file year.  Default is latest value from metadata */
       %IF &MON= AND &YR= %THEN %DO;  /* get latest information from metadata */
              PROC SORT DATA = &METADATA;
                     BY year month;
              RUN;
              DATA LATEST;  /* data set containing latest metadata info */
                     SET &METADATA END=LAST;  /* flag last record */
                     BY year month;
                     IF LAST;
              RUN;
              %END;  /* &MON and &YR missing */
       %ELSE %DO;  /* month and year selected */
              PROC SORT DATA = METADATA NODUPKEY OUT=LATEST;
                /* select the one record from metadata info */
                     BY year month;
                     WHERE year=&YR AND month=&MON;
              RUN;
              %END;  /* &MON and &YR missing */
       DATA _NULL_; SET LATEST;  /* read latest metadata data set */
              CALL SYMPUTX (‘MONTH’, month); /*symputx converts to text */
              CALL SYMPUTX (‘YEAR, year); /*symputx converts to text */
              CALL SYMPUT (‘INSET’,’Input File’n); /* name of input file */
       RUN;
%MEND LATEST;
```

The macro above is overly simplified in that no default is set for the ‘METADATA’ parameter and no check is made to ensure validity of the year and month parameters if entered, but the concept is the same. The proper record is selected from the metadata data set and the variables in this record are converted to SAS macro variables.

Run Main SAS Program using Metadata Information

Often the metadata retrieved is sufficient to complete the production process. For example, if the name of the production SAS macro code was named ‘monthly’, completion of this process might be as simple as submission of three standardized programs.

```
%METADATA  /* metadata is the name of a compiled SAS macro that reads all standard
metadata information stored in the data warehouse and creates various data sets with a
standardized naming construct */
%LATEST(METADATA=MonthlyControl)
%MONTHLY(MON=&MONTH,YR=&YEAR,INSET=&INSET)
```

Note that the ‘MON’,’YEAR’, and ‘INSET’) macro variables created in the macro %LATEST are passed again into the standard program ‘MONTHLY’. This was done for illustrative purposes only. In a production environment, the SAS macro ‘GLOBAL’ statement would be used within the ‘LATEST’ macro to set these values the one time. These parameters would then no longer be required as part of the ‘MONTHLY’ macro.

This illustrates another important point in the use of control file information and metadata. Once deliverables of all production parameters in the monthly job are moved from the standard process (in the example above the ‘MONTHLY’ macro) to the metadata recovery process (in the example above the ‘LATEST’ macro), then the ‘MONTHLY’ program is not modified month to month and the ‘LATEST’ program should contain GLOBAL statements

to ensure once the proper metadata is retrieved, it is available throughout the end of the process. In the case of the monthly update, the GLOBAL statement would appear as follows within the 'LATEST' macro.

```
%GLOBAL MONTH YEAR INSET;
```

Note Results of Production Program Execution

Storing control file information external to the production code requires increased diligence in that the process completes without any interaction into the main SAS code. To thwart any complacency that may occur, the log from the production SAS code should be written to a tab in the same worksheet file storing the control file information. Although various means exist to store results of the SAS log to an external file, the most direct way is through the use of the 'printto' procedure. Many options and settings exist as to selecting what exact information is written to the SAS log, but I will leave that up to the discretion of the reader as all SAS log items have different meaning in various context.

```
%MACRO PRINTLOG;
       PROC PRINTTO NEW
       LABEL="Process: Monthly Control   Year: &YEAR   Month: &MONTH   Input File: &INSET"
       LOG="/datawarehouse/MonthlyControl/log&YEAR.&MONTH..xls";
       RUN;
%MEND PRINTLOG;
```

Note that the label for this file contains the record information from the selected control file record. This is of critical importance in tying the input parameter information to the results of the execution of that code. A more sophisticated time stamp on this file can be created but this will suffice for this simple example. Also, code could be included in this simple macro to scan the results for something critical such as an error code. The level of complexity of this step needs to match the need for verification from all those having write access to the control data file. Providing the means to drive a process through a control file requires an increased emphasis on ensuring results are completed within expectations.

Quality Assurance Results Can also be a Metadata Component

The bottom line is that usage of any control file structure instead of regular altering of macro parameters in a regularly run process will only be proven successful if the structure and maintenance of the control file meets every expectation of others involved in the process. Thankfully Microsoft Excel is so ubiquitous that the file structure should cause no concern. Once enough information has been added to the spreadsheet tab holding the parameters as well as explanatory comments and the format of the SAS log file is sufficient to overcome concern, adding quality assurance tests to the process adds little time and effort and can become a powerful analysis tool, especially in conjunction with historical metadata.

Building a QA component to a metadata data warehouse presumes a standard process by which files of the same structure are analyzed in a timed manner. In the case of the example noted in this paper, new files of a consistent structure are processed each month.

The key to enacting quality control as part of the overall process without adding significantly to the time and resources required is to build a series of macro variable arrays from a once-through reading of the source file. Without too much difficulty, this data step can be performed in conjunction with the initial creation of the SAS data step, but to show the full code of that process would be excessive. So for this example, I will show the process as if part of a 'data _null_' step. Also, for the sake of simplicity, imagine the input data set for this process has two categorical variables (category_a and category_b) and two numeric variables (number_a and number_b).

```
DATA SAMPLE;
       INPUT VARNAME $ TYPE $ MAX $MIN $ MISSING $;
       CARDS;
Category_a C N N Y
Category_b C N N N
Number_a   N Y Y Y
Number_b   N Y Y N
;
RUN;

%MACRO QA(INSET=,ANALYZE=SAMPLE,MAX=N,MIN=N,MISSING=Y);
       /* INSET  - name of data set to be analyzed */
       /* ANALYZE - name of data set containing analysis required */
       /* MAX - Note the maximum value seen within source file */
```

```
        /* MIN - Note the minimum value seen within source file */
        /* MISSING - Note the number of missing values */
        DATA _NULL_; SET &ANALYZE;
               I + 1;  II=LEFT(PUT,I,2.));  /* count of variables */
               CALL SYMPUT('VARNAME'||II,VARNAME);  /* macro array variable */
               CALL SYMPUT('VARTYPE'||II,VARTYPE);  /* macro array variable */
               CALL SYMPUT('MAX'||II,MAX);  /* macro array variable */
               CALL SYMPUT('MIN'||II,MIN);  /* macro array variable */
               CALL SYMPUT('MISSING'||II,MISSING);  /* macro array variable */
               CALL SYMPUT('TOTAL',II);  /* number of variables */
               RUN;
        DATA QA  /* in production code, this may be part of initial read */
               (KEEP=VARNAME VARTYPE MAX MIN MISSING);
               LENGTH VARNAME $32 TYPE $1;
               SET &INSET END=LAST;  /* flag last record */
               ARRAY MAX _TEMPORARY_;
               ARRAY MIN _TEMPORARY_;
               ARRAY MISSING _TEMPORARY_;
               IF _N_=1 THEN DO I = 1 TO &TOTAL;  /* initialize arrays */
                      %IF %UPCASE(&&MAX&I)=Y %THEN %DO; /* if max flag=Y */
                             MAX{I}=&&VARNAME&I; /* retain value */
                             %END;
                      %IF %UPCASE(&&MIN&I)=Y %THEN %DO; /* if min flag=Y */
                             MIN{I}=&&VARNAME&I; /* retain value */
                             %END;
                      MISSING{I} = 0; /* set default for missing flag to zero */
                      END;
               ELSE DO I = 1 TO &TOTAL;  /* process each var in source file */
                      %IF %UPCASE(&&MAX&I)=Y %THEN %DO; /* if max flag=Y */
                             MAX{I}=MAX(&&VARNAME&I,MAX{I});
                             %END;
                      %IF %UPCASE(&&MIN&I)=Y %THEN %DO; /* if min flag=Y */
                             MIN{I}=MIN(&&VARNAME&I,MIN{I});
                             %END;
                      %IF %UPCASE(&&MISSING&I)=Y %THEN %DO; /* missing flag */
                             %IF &&VARTYPE&I=N %THEN %DO; /* missing number */
                                    IF &&VARNAME&I=. THEN MISSING{I}+1;
                                    %END;
                             %ELSE %DO; /* missing text variable */
                                    IF &&VARNAME&I='' THEN MISSING{I}+1;
                                    %END;
                             %END;
                      %END; /* updating counters for temporary arrays */
               IF LAST THEN DO I = 1 TO &TOTAL; /* end of file processing */
                      VARNAME="&&VARNAME&I";  /* data set names to be exported */
                      VARTYPE="&&VARTYPE&I";
                      %IF %UPCASE(&&MAX&I)=Y %THEN %DO; /* if max flag=Y */
                             MAX=MAX{I};
                             %END;
                      %IF %UPCASE(&&MIN&I)=Y %THEN %DO; /* if min flag=Y */
                             MIN=MIN{I};
                             %END;
                      %IF %UPCASE(&&MISSING&I)=Y %THEN %DO; /* missing flag */
                             MISSING=MISSING{I};
                             %END;
                      OUTPUT;  /* create a record for each variable */
                      END;
               RUN;
%MEND QA;
```

For the sake of simplicity, only the minimum value, maximum value, and number of missing values were tracked in this QA example. All manner of metrics can be calculated using this method. The key to success using such a method is to work closely with all member of a work group requiring quality assurance to alleviate any concern.

The results of the data set created in the macro above can be written to an excel spreadsheet as simply as follows:

```
PROC EXPORT DATA=QA
```

```
OUTFILE='datawarehouse/MonthlyControl/QA&YEAR.&MONTH..xls";
RUN;
```

Creation of much more sophisticated and formatted Excel spreadsheet results is available beyond the scope of Proc Export. Many papers and documentation has been presented on using DDE, XML, and the SAS Output Delivery System (ODS) to create all manner of spreadsheet reports. This topic is beyond the scope of this paper. However, should the presentation of the resulting spreadsheet information generated though the process described here require increased sophistication, please search through both the SAS papers and the online documentation found at www.sas.com to find answers.

Control Files Superstructure in Terms of a Relational Database

Once the notion of control files as the driving mechanism of a SAS production process utilizing macros is accepted, structure imparted to the relationship between the control files themselves is often warranted. This relationship should function in the context of a relational database. What I am talking about here is a relational database structure composed of metadata – a high minded concept indeed! Except that at the core level, the contents of the control files themselves as well as their structure is quite simple.

Imagine a simple listing of all input files used in a production process. Obviously this list would contain the names of each file along with some descriptive information. Another list would contain which input files are utilized with each production process. Related to this would be a master data dictionary. This would contain a list of all variables within all input files, including of course the name of each variable and to which data set it belongs. Also included on this list might be QA attributes such as whether a test for missing variables or unique values is important. Finally, review the first list noted in this paper and imagine the name of the production process is also listed. The relationship between these control files is clear and construes a rudimentary relationship database. In fact, the simpler this relational database structure is to convey to all those involved with the production process, the better the implementation of the control file structure. Again, the advantage of utilizing Excel as the medium for this structure is the ease in which elaborate documentation can be added to the structure with no loss in processing efficiency.

Conclusion

The use of methodologies described in this paper can increase quality assurance upon completion of the process and increase control and tracking ability at the inception of the process. The control file spreadsheet acts as both a history log of the process as well as documentation. The resulting QA spreadsheet created can subsequently be compared to previous versions of the QA report to create traffic light reports noting significant changes in measured metrics from month to month. This methodology allows SAS to work as efficiently as possible while allowing others not as familiar with SAS but familiar with Microsoft Excel spreadsheets to closely monitor the entire process.

Recommended Reading

Most of the essential SAS Macro wisdom I've learned has come from Art Carpenter. Many of the control set ideas stem from this publication.

Carpenter, Art. 2004. *Carpenter's Complete Guide to the SAS® Macro Language, Second Edition.* Cary; ND: SAS Institute Inc.

Contact Information

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

Mike Tangedal
3116 41st Ave S
Minneapolis MN 55406
Phone: 612-747-3797
E-mail: Minneapolis_mike@yahoo.com