Paper 094-2012

Managing Analytic Processes with Paired SAS ® Utility Macros

Zhongwen Huang, Outcomes & Analytics, Walgreen Co., Deerfield, IL

John Hou, Outcomes & Analytics, Walgreen Co., Deerfield, IL

ABSTRACT:

A pair of macros was developed to enhance user analytic process management in interactive SAS sessions. %_mStart(), the preprocessing macro, creates a relative path system independent from hard-coded absolute paths and is executable without program modifications when copied to a different location. In addition, %_mStart() can back up SAS source code automatically while the program is submitted for executing. The backed-up SAS program will have date and time stamp added to the original file name, serving the purpose of version control which is not directly available in a typical analytical SAS environment. Following the end of execution of the program, a post-processing macro %_mEnd() detects error and warning messages in a SAS log window and displays them in a pop-up window.

Keyword: Macro; Relative Path; File Backup; Version Control; Error Detection.

INTRODUCTION:

SAS offers a full spectrum of products and solutions that supports business analytics and academic research. Many SAS business users have in-depth familiarity with SAS modules related to their fields of specialties. However, they rarely have exposure to the backend system environment options that can be used to improve their analytic quality and efficiency. Increasing users' awareness of tools to improve quality and efficiency is especially important in our fast paced business and analytic environments.

In our experience, many users neglect to backup their work regularly and are unaware of version control processes. This leads to lost work that needs to be redone or excessive time spent finding a particular version of a working SAS program when needed.

In addition, SAS users typically read the log window every time when submitting SAS code which is appropriate in application development stage. It becomes less effective and inefficient after the application is developed and deployed. Large volumes of process information are written to the log window, searching for error and warning messages requires careful attention. A process to automatically monitor SAS sessions and check for warning and error messages in the log window is another process that could improve quality and efficiency.

We developed a pair of SAS macros to address the needs for version control and error detection.

1. PRE-PROCESSING MACRO %_MSTART() - BUILD A RELATIVE PATH SYSTEM WHILE AUTOMATICALLY ARCHIVE THE ACTIVE PROGRAM

Rather than hard coding an absolute path, this macro enables easy transfer of programs to new locations using a relative path derived from available SAS system parameters SAS_EXECFILEPATH and SAS_EXECFILENAME.

As an example, a project under "c:\project1" has two sub-directories: 'c:\project1\data' and 'c:\project1\code'. To access the data, we include the following statement in our SAS program: 'Libname my_data "c:\project1\data\";'. All programs that were debugged and validated will run well without any errors when the project directories were not changed. However, issues arise if projects are migrated to a new location, typically a different directory (e.g., "c:\client\projectA"). Code modifications have to be applied to make the programs work and the original project directory name for the data has to be manually changed to 'Libname my_data "c:\client\projectA\data\";'.

When many similar statements exist, the work involved can become lengthy, tedious and prone to errors especially when the project contains several SAS program files and the adjustment are needed in multiple places. By incorporating macro %_mStart() and building a relative path, the project can be copied to a new directory and still be executed smoothly without modifications.

Following is the source code for macro %_mStart():

```
* ------------------------------------------------------------------------------;
* Name01 : _mStart()                                                            ;
* Purpose: 1. Generate macro variables for current directory and file           ;
*          2. Back up source SAS file by adding date and time stamp             ;
*          %_mStart() should be called every time before we start our program   ;
* ------------------------------------------------------------------------------;
%macro _mStart(backup=Y);
   options noxwait;
   %global program_pathname _program_name program_path;
   * Clear the log window when we start the program;
   dm 'log;clear';
   dm 'clear output';

   data _null_;
       _datetime= translate(translate(put(datetime(), E8601DT.), '','-'), '',':');
    call symputX('_DateTime',translate(compress(_datetime),'-','T'),'G');
   run;

   data _null_;
       xpath="%sysget(SAS_EXECFILEPATH)";
    xname="%sysget(SAS_EXECFILENAME)";
       call symput('program_pathname', strip(xpath));
call symput('_program_name', substr(strip(xname),1, length(xname)-
index(strip(reverse(xname)),'.')));
call symput('program_path', substr(xpath, 1, length(strip(xpath))-
index(reverse(strip(xpath)),'\')));
   run;

   %if &backup=Y %then %do;
       %let fref=&program_path.\SasFileBackUp;
       %if %sysfunc(fileexist("&fref"))ne 1 %then %do;
           data _null2;
                command='md '||'"'||"&fref"||'"';
             call symput('qfref', '"'||"&fref"||'"');
                call system(command);
         run;
      %end;
       %let CopyCommand =%bquote(')copy "&program_pathname"
"&program_path.\SasFileBackUp\&_program_name._&_DateTime..sas"%bquote(');
      x %unquote(&CopyCommand);
      options xwait;
  %end;

       %global StartTime;
       %let StartTime = %sysfunc(time());
%mend _mStart;
```

The %_mStart() SAS macro first obtains a SAS program name and path information for an active SAS session, then generates global macro variables with the values of the current SAS program directory and filename. These macro variables create a relative path independent of the absolute path. The relative path improves efficiency and convenience of code sharing and minimizes efforts in code adjustments when migrating code. . The relative path is established by statements in a program like the following:

```
% _mStart();
Libname my_data "&program_path.\Data\";
```

The %_mStart() macro also includes function for backing up SAS source files when a session is initiated. This can replace unreliable and inefficient manual file backups and prevent work losing.

With the above macro, historical file versions can be easily retrieved and restored. One can trace back the early versions based on the date and time stamp that are post suffix of the file names. For example, a SAS program with name 'Program.sas' will be backed up with the name 'Program_20110830-164348.sas' if the program was submitted at 16:43:48PM on Aug 30, 2011. This backed up file will be automatically saved under a subfolder named as SasFileBackUp, which is a level lower than the folder of the original SAS program residing folder.

%_mstart() backups SAS code by copying the original program or the most recent saved version, changes made to the program without saving will lost. File should be saved before submit.

We can also modify macro to back up the content in the editor window directly. Following code is a sample to implement this function.

```
%let SaveCommand =%bquote(')&_program_name._&_DateTime..sas%bquote(');
dm "wpgm; file &SaveCommand replace;" editor;
```

Macro %_mstart() offers several advantages. It allows SAS programs to be relocated more quickly and reduces the probability of errors during migration; organizes the backups so that users can easily find earlier version of their SAS programs and essentially eliminate the problem of "lost code."

2. POST-PROCESSING MACRO %_MEND() - AUTOMATICALLY DETECT ERRORS AND WARNINGS

SAS programs typically generate a large amount of log information. This macro automatically detects error and warning messages in the log window and thus prevents the need for users to manually scan each page of log information.

Following is the source code for macro %_mEnd():

```
 * ------------------------------------------------------------------------------;
* Name02 : _mEnd()                                                               ;
* Purpose: 1. Detect the error and warning messages                              ;
*          2. Calculate the total running time                                   ;
*          _mEnd() should be recalled every time when we end the program         ;
* Sample : _mEnd()                                                               ;
* ------------------------------------------------------------------------------;
%macro _mEnd() ;
    * Following code is used to scan all log information;
    * First create a file in work sheet to load all log information;
    filename _mlog catalog 'work._m.LogCheck.log';
    * Then save the log information to our file;
    dm   'log;file _mlog';
    ods  listing close ;
    %global _mErrorStatus;
    %let    _mErrorStatus = TRUE;
    data ErrorWarningSet;
         length line $ 120;
         infile _mlog end=end truncover ;
         input  line & ;
         if substr(line,1,5)      ='ERROR'
            or  (substr(line,1,7) ='WARNING' and (index(line,'expir') = 0
            and index(line,'The Base Product product') = 0)) then output;
         else
            n + 1;
         * if no error and warnings detected, error will not be displayed;
         if end and n=_n_ then call symput('_mErrorStatus',"FALSE");
    run;
```

```
    * Free file ;
    filename _mlog;
    * Delete temporary files;
    dm 'del work._m.LogCheck.log' results viewer;
    * Display error/warning messages in cases of being detected;
    %if &_mErrorStatus eq TRUE %then %do;
        ods html file="Error.htm";
            proc report data = ErrorWarningSet nowd noheader;
                 column  n line;
                 define  n            /  "Line #  "  left display;
                 define  line         /  "Messages"  left display;
                 compute line;
                   if substr(line,1,5) = 'ERROR'  then do;
                   call    define(           1    ,    "style","style=[FOREGROUND=RED
BACKGROUND=WHITE]");
                   call         define(_col_,         "style","style=[FOREGROUND=RED
BACKGROUND=WHITE]");
                   end; else do;
                   call    define(          1    ,    "style","style=[FOREGROUND=BLUE
BACKGROUND=WHITE]");
                   call        define(_col_,        "style","style=[FOREGROUND=BLUE
BACKGROUND=WHITE]");
                   end;
                endcomp;
            run;
        ods html close;
    %end;
    * Delete error message files;
    proc sql noprint; drop table ErrorWarningSet; quit;
    ods listing;
    * Calculate the total cost time;
    %let EndTime = %sysfunc(time());
    %let SpentTimeInSecond = %sysevalf(&EndTime-&StartTime);
    %let SpentTimeInMinute = %sysevalf(&SpentTimeInSecond/60);
    data _null_;
         call symput('SpentTimeInSecond',strip(put(&SpentTimeInSecond,8.2)));
         call symput('SpentTimeInMinute',strip(put(&SpentTimeInMinute,8.2)));
    run;
    %put  Total  running  time  is  &SpentTimeInSecond  seconds  or  &SpentTimeInMinute
minutes;
%mend _mEnd;
```

The %_mEnd() macro first captures and saves the session log information into a temporary file and then scans the file line by line for 'ERROR' and 'WARNING' keywords. If error or warning messages are detected, they are outputted to an html file and displayed as a pop up window; if there is no error or warning message then no html file is displayed. By using the %_mEnd() macro, users can eliminate the time-consuming task of manually reading every page of the log window. It also alerts users who may neglect to review warning messages.

An additional advantage of using the paired macros, %_mStart() and %_mEnd(), is that total elapsed time is generated which provides a good estimate of the run time needed for a specific job.

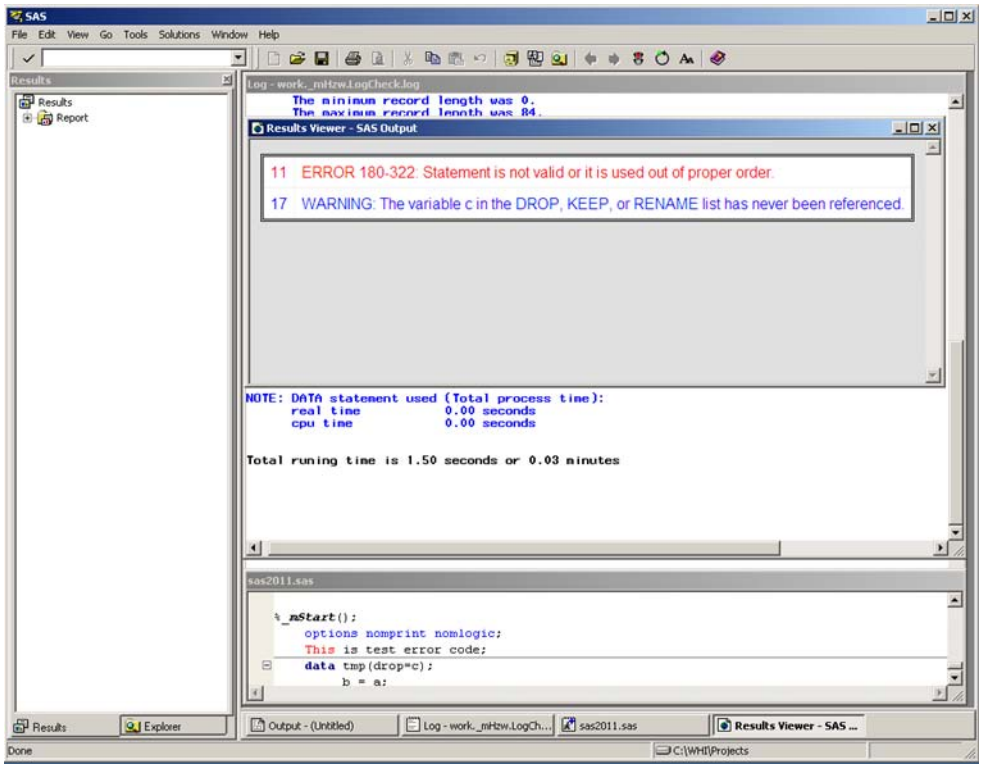
3. EXAMPLE

Following is a simple example to demonstrate the paired macro application:

```
%_mStart();
    options nomprint nomlogic;
    This is error code for test;
    data tmp(drop=c);
```

```
        b=a;
    run;
%_mEnd();
```

After submitting the code, the program is automatically backed up and a window appears, listing error and warning messages found in the log. The alert includes a reminder to check the code because of the listed errors or warnings.



Display 1. Sample Alert Received after Running the Macro

CONCLUSION

The pre-processing and post-processing paired macros offer a simple and convenient option for SAS users to manage their SAS sessions including version control, program and project archiving and code migrations. Once adapted, users can benefit from the macros in many ways such as automated version control, code debugging and process monitoring. Together these macros help analytic users can improve their quality and efficiency.

REFERENCES

SAS Institute Inc. SAS OnlineDoc@ 9.2, Cary NC. SAS Institute Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:
Zhongwen Huang,
Outcomes & Analytics, Walgreen Co.
1415 Lake Cook Road MS #L444, Deerfield, IL 60015
Phone: 847-964-6429 Fax: 847-374-2269
E-mail: Zhongwen.huang@walgreens.com
Or
John Hou, Ph.D.
Outcomes & Analytics, Walgreen Co.
1415 Lake Cook Road MS #L444, Deerfield, IL 60015
Phone: 847-964-8864 Fax: 847-374-2269
E-mail: John.Hou@walgreens.com