Paper 16-2019

Principles of Automation

Robert Ellsworth, Ellsworth Stewart Consulting Inc.

ABSTRACT

This paper discusses the basic principles to automate base SAS programs. The paper describes how automation is achieved by using no manual code changes, programmatically populating output, scheduling, process run control, validating inputs, validating results, email notification, restartability, and using standard code.

INTRODUCTION

This paper came about from a major project at a bank in Canada. The problem we faced. Report analysts did not have time to run the existing reports and develop new reports. They had over 200 reports to be run either daily, weekly, or monthly. The reports took between 30 minutes and 4 hours of analyst time to run. There was a large demand for changes to existing reports and new reports.

The possible solutions add analysts, decline requests for new reports and enhancements, reduce existing reports, decrease time needed to produce reports through automation.

They chose to eliminate unused and redundant reports, but most of the time saved was created by this project was through report automation.

PRINCIPLES OF AUTOMATION

We identified the guidelines to automate a report. They are list below.

- No manual changes of code between runs
- Programmatically populate output
- Schedule on recurring basis
- Use process run control
- Validate inputs including parameters
- Programmatic result validation
- Email notification of success/failure/warning
- Restartable
- Utilize standard codes and macros

NO MANUAL CODE CHANGES

The program must be coded so no code changes are required between runs. The common issues were date comparison in query, run dates on report, filenames, user id and password.

USING DATES IN SAS PROGRAM

To avoid hard coding dates in a SAS program, you can calculate the date into a macro variable and use the variable in the code.

Program:

```
%let dt = %sysfunc(intnx(month,%sysfunc(today()),-1,end));
data _null_;
  call symput('yymm',put(year(&dt),4.)||put(month(&dt),z2.));
  call symput('period',"'"||put(&dt,yymmdd10.)||"'");
  call symput('rptdt',trim(put(year(&dt),4.)|| " "
   ||put(month(&dt),z2.) || " " || left(put(&dt,$monname.))));
run;
%put dt=&dt yymm=&yymm period=&period rptdt=&rptdt;
```

Result:

```
dt=20392 yymm=201510 period='2015-10-31' rptdt=2015 10 October
```

Figure 1 Program to set date macro variables

DATES AND OTHER PARAMETERS IN JCL

When running processes on the host some data file names include dates and other run dependent logic. We created a parameter file on the host each morning (12:01 am). This parameter file can be used in your JCL to avoid having to make JCL changes for each run.

```
//ELLSWR2A JOB (xxxx),'DATECARD',MSGCLASS=X,CLASS=F,
// PRTY=8,NOTIFY=&SYSUID
//TEMP JCLLIB ORDER=('TDGU.ACF2.BYPASS.RBPU.PARMLIB')
// INCLUDE MEMBER=DATEPARM
//SASRUN1 EXEC SAS,TIME=NOLIMIT,WORK='3000,3000',SORT=4,
// PARM='SYSPARM=&SDAY &PDAY'
//WORK DD UNIT=DISK,SPACE=(CYL,(4000,4000),RLSE)
//DGAMACRO DD DSN=RBPU.ELLSWR2.SASDATA.&CMTHYYYY,DISP=OLD
//SYSIN DD *
```

Figure 2 JCL to include date parameters

```
// SET TODAY=151101
// SET SDAY=151031
// SET PDAY=151030
// SET SDATE=151029
// SET PDATE=151028
// SET CMTHEND=151130
// SET SMTHEND=151031
// SET PMTHEND=150930
// SET CWKEND=151107
// SET SWKEND=151031
// SET PWKEND=151024
// SET PMTH=1510
// SET SUNDAY=151101
// SET MONDAY=151102
// SET P6MTHEND=201509
// SET P1MTHEND=29SEP15
```

```
// SET P2MTH=1509
// SET CMTH=1511
// SET CMTHYYYY=201511
```

Figure 3 Example JCL parameters

USER ID AND PASSWORD

In order to access other systems and databases from SAS, you will often need a user ID and password. Rather than code this into your program, you can use a secure file in your home directory.

The SAS program change_pswd.sas creates a logon file on Unix and Windows which contains macro variables that allow you to access these other systems and databases.

Caution must be taken when using the macro variables for user ID and password. If SYMBOLGEN is set, the user id and password will be displayed in the log.

```
%include'~/logon.sas';
Libname edwdb2 datasrc=prd1 schema=edw user=&user password=&password;
```

Figure 4 Example of using password file

```
%let user = xxxx;
%let password = xxxx;
%let username = xxxx;
Filename logon "c:\user\Documents\My SAS Files\9.3\logon.sas";
Data _null_;
  file logon;
  put '%let user = ' "&user;";
  put '%let password = ' "&password;";
  put '%let username = ' "&username;";
run;
options comamid=tcp netencryptalgorithm=sasproprietary;
%let Td3=scsnmap03 7551;
Filename rlink"c:\user\Documents\My SAS Files\9.3\unix_script.scr";
Signon Td3;
%syslput user=&user;
%syslput password=&password;
%syslput username=&username;
rsubmit;
  filename logon "/home/logon.sas";
  data _null_;
    file logon;
    put '%let user = ' "&user;";
    put '%let password = ' "&password;";
    put '%let username = ' "&username;";
  run;
  x"chmod600 /home/logon.sas";
endrsubmit;
```

Figure 5 Program to create password file

AUTO LOG ON

To automate a process that runs on PC SAS but needs to access UNIX based file or database you can change the logon script to use the macro variable for user and password. Caution must be used when changing the script if echo is set, the user id and password will be displayed in the log.

```
%include "c:\user\Documents\My SAS Files\9.3\logon.sas";
Options comamid=tcp netencryptalgorithm=saspropriety;
%let Td3=scsnmap03 7551;
Filename rlink "c:\user\Documents\My SAS Files\9.3\unix_script.scr";
Signon Td3;
```

Figure 6 Program to auto logon

```
/*   input 'Userid?'; change input to send the user id macro variable*/
   type "&user" LF;
   waitfor 'Password', 30 seconds : nolog;
/*   input nodisplay 'Password?'; send the password macro variable */
   type "&password" LF;
```

Figure 7 Auto logon rlink script changes

OPTIONS TO PROGRAMMATICALLY POPULATE OUTPUT

The goal is to eliminate cut and paste activity for producing reports. SAS needs to place the data in the report or somewhere the report can retrieve it.

The Options:

- ODS to xls or xml with style sheets
- CSV file with excel macro
- Write to a database like SQL Server or DB2 with connection in excel
- SAS Addin for Microsoft Office
- Proc export
- Write directly to excel with DDE

ODS TO XLS OR XML

Using ODS SAS can generate an Excel formatted or xml formatted file that can be opened in excel. Use a style sheet to format the report.

Pros:

- SAS can be run on any platform, not just Windows

Cons:

- Limited formatting
- No longer have full power of excel in producing the report

CVS FILE WITH EXCEL MACRO

SAS program writes the result set to a CSV file. File is opened in excel and a format macro is run.

Pros:

- SAS can be run on any platform
- Full power of excel is available to build report

Cons:

- Two-step process
- VB script skills required

SQL SERVER, DB2, …

SAS program writes result set to a database table. The table is then opened in excel via a database connection.

Pros:

- SAS can be run on any platform that has access to database
- Full power of excel is available to build report

Cons:

- Two-step process
- Restructure report to work with database connection

SAS ADDIN FOR MICROSOFT OFFICE

SAS program processes a result set into a SAS dataset that is accessible to excel. The table is then opened by Excel via a SAS Addin connection.

Pros:

- SAS can be run on any platform where the dataset is visible to excel
- Full power of excel is available to build report

Cons:

- Two-step process
- Restructure report to work with SAS Addin

PROC EXPORT

SAS program writes data directly to excel workbook using proc export.

Pros:

- Full power of excel is available to build report

Cons:

- SAS process must run on PC SAS
- Restructure report to work with exported sheet

SAS WRITES TO EXCEL VIA DDE

SAS program writes data directly to cells in excel workbook using DDE protocol.

Pros:

- Full power of excel is available to build report
- No changes to existing excel workbooks required

Cons:

- SAS process must run on PC SAS

DDE: THE SELECT OPTION

To minimize changes to the existing reports and work with analysts' skillsets, DDE was selected.

Example of Using DDE

```
Options noxwait noxsync;
x """O:\Programs\fee_waiver\Weekly Fee Waiver temp.xlsx""";
data_null_;
  x=sleep(10);
run;
filename data dde"excel|sheet1!r1c1:r10c10"lrecl=32000;
data _null_;
  file data notab;
  set cust;
  if _n_ = 1 then put"customer name"'09'x"customer address";
  put name '09'xaddr1;
run;
filename cmds dde'excel|system';
data _null_;
  file cmds;
  put"[SAVE.as(""O:\Reports\fee_waiver\Weekly Fee Waiver.xlsm"")]";
  put'[QUIT()]';
run;
```

Figure 8 Program to output to Excel via DDE

DATA CLASSIFICATION OF OUTPUT

At the bank all output to excel must be Data Classified. If you create your output using a template excel sheet and the template has been classified, your output will retain that classification. You must insure that the classification of the template reflects the correct classification of the output.

If you are creating an excel workbook without a template, you *must* classify the output.

SCHEDULED ON A RECURRING BASIS

Programs should be scheduled on a recurring basis. These are macro's were created to schedule programs.

- sch_unix.sasfor jobs to run on Unix.
- sch_host.sasfor jobs to run on Host.
- sch_windows.sasfor jobs to run on Windows.

SCH_UNIX MACRO

Schedules a SAS program to run on the Unix platform at a specific time and frequency.

Syntax: %sch_unix(os,pgm,execdir,jobname,dt,repeat);

- **os**–From where to copy the SAS program. W –Windows environment, U –Unix environment, and N –do not copy the program.
- pgm–Program name including path.
- **execdir**–directory on Unix from where the program is to be executed. This is where the log and lstfiles will be created.
- **jobname**–Used for the jobname in bsub, also used for names of the command file and other control files
- **dt**–Optional parameter that specifies the date and time of execution. Format is mm:dd:hh:mm. If left lank the SAS program will execute immediately.
- **repeat**–How frequently the SAS program is to be executed. The parameter is optional. If blank the program is not repeated. The valid values are H –Hourly, D-Daily, W –Weekly, M -Monthly

```
%include'/td/edw/general/general1/edwcore/dga_macro/sch_unix.sas';
%letdt = %sysfunc(month(%sysfunc(today()))):%sysfunc(day(%sysfunc(today())));
%sch_unix(w,O:\Programs\macros\getrates.sas,
          /td/imss/home/ellswr2/general/ellswr2/getrates,
          getrates,&dt:17:30,d);
```

Figure 9 Example call to Unix schedule macro

In the above example, we are running the program getrates.sas from a directory on Windows. The execution directory is in general file system on Unix to avoid space issues in the home directory. The jobname is getrates. Today's month and day are calculated so I don't have to enter them, but it still runs at 5:30pm. The job will execute everyday going forward.

SCH_HOST MACRO

Schedules a SAS program on the Host to run on the Host at a specific time and frequency. The SAS program is launched on the host by running some jclon the host via ftp that jclcopies the SAS program to the internal reader.

Syntax: %sch_unix(jcl,jobname,dt,acct,repeat);

- **jcl**–Program name and dsnfor the SAS program on the Host.
- **jobname**–Used for the jobname in bsub, also used for names of the command file and other control files
- **dt**–optional parameter that specifies the date and time of execution. Format is mm:dd:hh:mm. If left blank the SAS program will execute immediately.
- **acct**–Account number the programmer would use on a jcljobcardon the Host.
- **repeat**–How frequently the SAS program is to be executed. The parameter is optional. If blank the program is not repeated. Valid values are H –Hourly, D-Daily, W –Weekly, M -Monthly

```
%include '/td/edw/general/general1/edwcore/dga_macro/sch_host.sas';
%let dt = %sysfunc(month(%sysfunc(today())+1)):%sysfunc(day(%sysfunc(today())+1));
%sch_host(RBPU.ELLSWR2.JCLLIB(DATECARD),datecards,&dt:00:01,2397,d);
```

Figure 10 Example call to the Host schedule macro

In the above example we are running the program RBPU.ELLSWR2.JCLLIB(DATECARD) on the host. The jobname is datecards. Tomorrow's month and day are calculated so I don't have to enter them, but it still runs at 12:01am. The job will execute everyday going forward.

SCH_WINDOWS MACRO

Schedules a SAS program to run on the Unix platform at a specific time and frequency.

Syntax: %sch_windows(os,pgm,execdir,jobname,dt,repeat);

- **os**–From where to copy the SAS program. W –Windows environment and N –do not copy the program.
- pgm–Program name including path.
- **execdir**–Directory on Unix from where the program is to be executed. This is where the log and lstfiles will be created.
- **jobname**–Used for the jobname in bsub, also used for names of the command file and other control files
- **dt**–Optional parameter that specifies the date and time of execution. Format is mm:dd:hh:mm. If left blank the SAS program will execute immediately.
- **repeat**–How frequently the SAS program is to be executed. The parameter is optional. If blank the program is not repeated. The valid values are H –Hourly, D-Daily, W –Weekly, M –Monthly

```
Libname dgamacro remote '/general1/edwcore/dga_macro/' server=td3;
Options intervalds=(BD=dgamacro.BankDayDS);
%let dte=%sysfunc(intnx(BD,%sysfunc(intnx(month,%sysfunc(today()),0,e)),3));
%let dt = %sysfunc(month(&dte)):%sysfunc(day(&dte));
%include "O:\Programs\macros\sch_windows.sas";
%sch_windows(w,O:\Programs\FinanceFeed\Load pil.sas,
             O:\Programs\FinanceFeed\Log,load_pil,&dt:06:35,);
```

Figure 11 Example call to the windows schedule macro

In the above example we are running the program load pil.sas from a directory on windows. The execution directory is on a network drive to avoid space issues in the home directory. The jobname is load_pil. The third business day is calculated so I don't have to enter them, and it runs at 6:35pm. The job will not repeat.

PROCESS RUN CONTROL

SAS macro language can be used to control how a SAS job runs. You can make it stop on error, skip steps that are already completed, notify of completion, and perform data validation. Basic run control can be done using a label and the goto statement.

```
%macro run_pgm();
  data rpt;
    infile xyz;
    input a b c;
  run;
  %if %sysfunc(today()) = &sysparm %then %goto pgmend;
  ....
  %pgmend:
%mend;
%run_pgm;
```

Figure 12 Example program using run control

USING ERROR ROUTINES FOR PROCESS RUN CONTROL

The error macros were built to help you check the progress of a program. A copy is kept on all platforms i.e. windows and Unix.

You use these macros after each data step or proc. They check the error codes, send an email if there is an error, and set an error flag. The flag is used stop processing on error.

```
%include'/td/edw/general/general1/edwcore/dga_macro/error_routines.sas';
%macro run_pgm();
  data rpt;
    infile xyz;
    input a b c;
  run;
  %chk_err(build report);
  %if &failed = 1 %then %goto pgmend;
  ....
  %pgmend:
%mend;
%run_pgm;
```

Figure 13 Example program using chk_err() macro

MACRO %CHK_ERR()

The chk_err macro checks sas error codes and sets a failed flag if there is an error, prints a message to the log, and sends an email to the programmer if there is an error.

```
%macro chk_err(msg);
  %put finished &msg;
  %if &syserr>0 or &sysrc>0 or &syscc>0 or &sysxrc^=0 %then %do;
    %put ERROR syserr=&syserr sysrc=&sysrc syscc=&syscc;
    %if &sysxrc^=0 %then
      %put ERROR sysxrc=&sysxrc sysxmsg=&sysxmsg;
    %let sysrc=0; %let syscc=0;
    %email(job failure,%quote(&syserrortext&msg));
    %let failed = 1;
  %end;
%mend;
```

Figure 14 Chk_Err() macro

MACRO %CHK_NOT_EMPTY()

The chk_not_empty macro verifies if a dataset has any records that fit the where clause, sets the failed flag and emails the programmer if the number of rows in the dataset with the where clause applied are 0.

```
%macro chk_not_empty(msg,ds,whr);
  proc sql;
    select count(*) into:cnt
      from &ds
      &whr;
    quit;
  %if &cnt= 0 %then %do;
    %put ERROR: &ds is empty &msg syserr=&syserr sysrc=&sysrc syscc=&syscc;
    %email(job failure,&dsempty &msg);
    %letfailed = 1;
  %end;
%mend;
```

Figure 15 Chk_not_empty() Macro

MACRO %CHK_NOT_EMPTY_UDB()

The chk_not_empty_udbmacro verifies if a db2 table has any records that fit the where clause, and sets the failed flag and emails the programmer if the number of rows in the db2 table with the where clause applied are 0.

```
%macro chk_not_empty_udb(msg,ds,whr);
  proc sql;
    connect to db2(datasrc=prd1 user=&user password=&password);
    select cnt into:cnt from connection to db2(
      select count(*) as cnt
        from &ds
        &whr);
  quit;
  %if &cnt=0 %then %do;
    %put ERROR: &ds is empty &msg syserr=&syserr sysrc=&sysrc syscc=&syscc;
    %let failed = 1;
    %email(job failure,&ds not empty &msg);
  %end;
%mend;
```

Figure 16 Chk_not_empty_udb() macro

MACRO %EMAIL()

The email macro sends an email to the programmer when executed.

```
%macro email(subject,msg);
  filename mail email to="&mailto" subject="&pgname&subject";
  data _null_;
    file mail;
    msg = symget('msg');
    put "Program: &pgname" /;
    put msg;
  run;
%mend;
```

Figure 17 Email() Macro

VALIDATE INPUTS INCLUDING PARAMETERS

Validating inputs is critical to ensure correct results. As we all know: garbage in, garbage out. The program needs to check if the input file is for the correct period. A date parameter is consistent with report period and run date.

```
DATA TRANS&rgn;
  INFILE PDATRANS MISSOVER;
  INPUT @6 file date7.;
  If filedate^=&fchkdt then do;
    Put "bkiflags&rgn not current " filedate "^= &fchkdt";
    Call symput('failed','1');
    stop;
  end;
run;
```

Figure 18 Check date on header

```
%if %sysfunc(fileexist(O:\ Reports\account scorecard\&rptdt)) = 1
   %then %goto pgmend;
```

Figure 19 Check report exists

```
%chk_not_empty_udb(data not available,edw.acct_holdng_balnce,
            %str(where efectv_dt= &period and ACCT_FAMILY_MN = 'LOC'));
%if &failed = 1 %then %goto pgmend;
```

Figure 20 Check if data available

```
%if %sysfunc(today()) = &passeddate %then %goto pgmend;
```

Figure 21 Check date parameter

```
%if %sysfunc(exist(datamart.loc&yymm)) = 0 %then %goto pgmend;
```

Figure 22 check that dataset exists

```
data _null_;
  if today() > &sysparm then abort abend 25;
run;
```

Figure 23 Check that sysparm is correct for host process

```
%if %sysfunc(exist(datamart.loc&yymm))=1 %then %goto pgmend;
```

Figure 24 Check that dataset doesn't exist

```
%if %sysfunc(fileexist(O:\Reports\account scorecard\&rptdt))=0
    %then %goto pgmend;
```

Figure 25 Check that file exists

```
DATA INFO;
  LENGTH INFONAME INFOVAL $60;
  DROP RC FID INFONUM I CLOSE;
  FID=FOPEN("PARMS");
  INFONUM=FOPTNUM(FID);
  DO I=1 TO INFONUM;
    INFONAME=FOPTNAME(FID,I);
    INFOVAL=FINFO(FID,INFONAME);
    OUTPUT;
    IF I = 8 THEN DO;
      CREATED=INPUT(INFOVAL,YYMMDD10.);
      IF CREATED LT TODAY() -50 THEN ABORT ABEND 25;
    END;
  END;
  CLOSE=FCLOSE(FID);
RUN;
```

Figure 26 Check creation date on Host

PROGRAMMATIC RESULT VALIDATION

Validate the results to be reported based on a verified source. Possible sources last month's report, other database, and other reports. Notify the programmer of any exceptions.

```
Data _null_;
  Retain warnings;
  Format warnings $32000.;
  Merge finance.crdt_&pyymm(keep=plan volume rename=(volume=old_volume))
        finance.crdt_&yymm;
  by plan;
  if volume > old_volume* 1.15 or volume < old_volume* .85 then do;
    msg = "Plan "|| trim(put(plan,$32.)) ||
          " volume change more than 15% "|| put(old_volume,comma15.) ||
```

```
            " to "|| put(volume,comma15.);
      Call symput('warning','1');
  warnings = trim(warnings) || " "|| msg;
  call symput('warnings',trim(warnings));
  end;
  run;
  %if &warning = 1%then%email('warning',%quote(&warnings));
```

Figure 27 Result validation compare this month with last month

EMAIL NOTIFICATION OF SUCCESS/FAILURE/WARNING

Automated programs need to let the programmer know when they are completed, whether successful or not.

%email is a macro that allows the program to notify the programmer. You pass it a subject and a message and it sends the programmer an email. You should call the email macro on successful completion, the error macros will send an email on error, and a warning email should be sent as part of result validation.

```
  %macro run_pgm();
  ....
  %email(job completed, job completed);
  %pgmend:
  %mend;
```

Figure 28 Example email at end of process

RESTARTABLE

How to make a program restartable:

- Save transient datasets in a permanent directory
- Don't rerun dataset generation if not in error
- Don't update dataset in place such that a rerun would not generate the desired result

```
  %if %sysfunc(exist(datamart.loc&yymm)) = 0 %then %do;
    Proc sql;
      Create table datamart.loc&yymm as
        Select * from connection to db2(
          Select a.acct_id, a.acct_no,
                 .....
           Order by a.acct_id);
  %end;
  %chk_err(getting account information);
  %if &failed = 1 %then %goto pgmend;
```

Figure 29 Example program to conditionally create dataset

UTILIZE STANDARD CODES AND MACROS

Using standard code and macros ensures that program can be transferred between analysts without coding changes.

Macros enforce consistent methods. Often changes to environment can be made in the macros without programing changes.

Consistent methodology reduces training time on new reports.

CONCLUSION

By automating our reports, scheduling the execution, and validating both input and output we were able to:

- Cut the time spent of analysts running BAU reports in half.
- Increased the analyst's job satisfaction because they were able to spend more time on development.
- Increase business user satisfaction by get the reports done on time and with fewer errors.

CONTACT INFORMATION

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

Robert Ellsworth
Ellsworth Stewart Consulting Inc.
416-414-1172
rob@escorp.ca