

# Principles of Automation



# The Problem



- ▶ Over 200 reports to be run either daily, weekly, or monthly
- ▶ Reports take between 30 minutes and 4 hours of analyst time to run
- ▶ Changes to existing reports and new reports are being requested
- ▶ Analysts don't have time to run the existing reports and develop new reports

# Possible Solutions



- ▶ Add analysts
- ▶ Decline requests for new reports and enhancements
- ▶ Reduce existing reports
- ▶ **Decrease time needed to produce reports through automation**

# Principles of Automation



- ▶ 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.

Common issues:

- ▶ 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.

## ▶ Example:

```
%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
```

# Dates and Other Parameters in JCL



When running processes on the host some data files include dates and other run dependent logic. There is a parameter file created 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.

## ► Example:

```
//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 *
```

# JCL 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
```



# 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.

## ► Example:

```
%include '~/logon.sas';  
libname edw db2 datasrc=prdl schema=edw user=&user password=&password;
```

# Change\_pswd.sas



```
%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 "chmod 600 /home/logon.sas";
```

# 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.

## ► Example:

```
filename logon "c:\user\Documents\My SAS Files\9.3\logon.sas";
options comamid = tcp netencryptalgorithm=sasproprietary;
%let Td3=scsnmap03 7551;
filename rlink "c:\user\Documents\My SAS Files\9.3\unix_script.scr";
signon Td3;
```

# 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:

- ▶ Can be done on any platform, not just Windows

## Cons:

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

# CSV 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 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 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 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:

- ▶ 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:

- ▶ 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:\Personal Deposits\Data Governance & Analytics\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'x addr1;
run;
filename cmds dde 'excel|system';
data _null_;
  file cmds;
  put "[SAVE.as(""O:\Personal Deposits\Data Governance & Analytics\Reports\fee_waiver\Weekly Fee
Waiver.xlsxsm"")]";
  put '[QUIT()]';
run;
```

# Data Classification of Output



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.sas for jobs to run on Unix.
- ▶ sch\_host.sas for jobs to run on Host.
- ▶ sch\_windows.sas for 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 lst files 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

# Sch\_unix Macro Example



```
%include '/td/edw/general/general1/edwcore/dga_macro/sch_unix.sas';  
%let dt = %sysfunc(month(%sysfunc(today()))):%sysfunc(day(%sysfunc(today())));  
%sch_unix(w,0:\Personal Deposits\Data Governance & Analytics\Programs\macros\getrates.sas,  
          /td/imss/home/ellswr2/general/ellswr2/getrates,  
          getrates,&dt:17:30,d);
```

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 date 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 jcl on the host via ftp that jcl copies the SAS program to the internal reader.

Syntax: %sch\_unix(jcl,jobname,dt,acct,repeat);

- ▶ **jcl** – Program name and dsn for 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 jcl jobcard on 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

# Sch\_host Macro Example



```
%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);
```

In the above example we are running the program RBPU.ELLSWR2.JCLLIB(DATECARD) on the host. The jobname is datecards. Tomorrow's month and date 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 lst files 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

# Sch\_windows Macro Example



```
libname dgamacro remote '/td/edw/general/general1/edwcore/dga_macro/'
server=td3;
options intervalds=(BankingDays=dgamacro.BankDayDS);
%let dte = %sysfunc(intnx(BankingDays,
                        %sysfunc(intnx(month,%sysfunc(today()),0,e)),3));
%let dt = %sysfunc(month(&dte)):%sysfunc(day(&dte));
%include "O:\Personal Deposits\Data Governance &
         Analytics\Programs\macros\sch_windows.sas";
%sch_windows(w,O:\Personal Deposits\Data Governance &
            Analytics\Programs\FinanceFeed\Load_pil.sas,
            O:\Personal Deposits\Data Governance &
            Analytics\Programs\FinanceFeed\Log,load_pil,&dt:06:35,);
```

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.

## ► Example:

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

# 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 a error flag. The flag is used stop processing on error.

# Example Error Routine



```
%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;
```

# Macro %chk\_err()

The chk\_err macro checks sas error codes and sets a failed flag if there is an error, prints a msg to the log, and sends a email to the program 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;
```



# 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,&ds empty &msg);  
    %let failed = 1;  
  %end;  
%mend;
```

# Macro %chk\_not\_empty\_udb()



The `chk_not_empty_udb` macro 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=prdl 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;
```

# 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;
```

# 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.

Check header date on input file:

```
DATA TRANS&rgn ;
INFILE PDATRANS MISSEVER ;
INPUT @6   filedate 7.;
if filedate ^= &fchkdt then do;
  put "bkiflags &rgn not current " filedate " ^= &fchkdt";
  call symput('failed','1');
  stop;
end;
run;
```

Report already created:

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

# Validate Inputs Including Parameters



Check that data is available:

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

Check that passed date parameter consistent with current date:

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

Check that dataset exists:

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

# Validate Inputs Including Parameters



Check that sysparm is correct for host process:

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

Check that dataset doesn't exist:

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

Check that file exists:

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

# Validate Inputs Including Parameters



Check that creation date is consistent with today's date:

```
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;
```

# Programmatic Result Validation



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

Compare this month with last month:

```
data _null_;
  retain warnings; format warnings $32000.;
  merge finance.crdt_int_&pyymm (keep=plan volume rename=(volume=old_volume))
        finance.crdt_int_&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));
```



# 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 a 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.

Email at end of process if successfully completed:

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

# 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;
```

# 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 programming 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 analysts 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.

**Questions?**

