They're Closing Down the Office in Kalamazoo and You've Been Tapped to Run Their In-house "Production" Reporting System, OH MY!

David Cherry Welch, Citi

Abstract

The author's experience in running a homegrown reporting system for a major corporation is described as well as these features of the system:

- Setting the SAS environment using command line switches and the autoexec.sas file.
- Controlling program start times within the SAS data step.
- Creating and maintaining report distribution lists using SAS formats.
- Modularizing reports using the SAS macro language and %include

The benefits of this ad hoc system will be compared to other alternatives. The author will discuss the lessons learned from this assignment and suggestions for making projects easily supportable for others that may follow.

Introduction

A unit of a major corporation needed reports to run various aspects of their business. Due to realignment in the company, the group that had formerly supplied database services to the unit was sold to an outside firm. Instead of being able to request a report or revision and walk down the hall or pickup the phone and talk to the reporting analyst, the process to get a new report or to change an existing one became a bureaucratic mess as well as expensive. Someone remembered that at an old job, SAS was the reporting tool. The employee recounted that SAS was able to read data directly from the company database and produce very functional reports. Given the reality of the new cost structure, the unit hired a dedicated reporting resource and purchased a SAS license.

SAS was an excellent reporting solution. The newly hired reporting analyst could quickly build proto-type reports for business analysts in the unit. Soon, the SAS reports became part of most everyone's tool set and more and more requests came in for new reports. Soon a large portion of the reporting analyst's time was spent reproducing reports on a regular basis. Another reporting resource was hired and together they developed a mechanism in SAS that ran the reports in an overnight batch run and automatically delivered them to the business analysts. Everyone was delighted. The reporting team became the foundation of the unit's business intelligence.

The reporting team built hundreds of reports and had daily, weekly, monthly, quarterly, and annual tasks that were built into the batch processing system they had developed. The unit had filled their reporting need using a SAS system that was based on an ODBC hook into the production database. The reporting team was busy, the business analysts were pleased, and the unit flourished.

Unfortunately for the people that worked in the unit, the major corporation decided to sell it. The office was closed and everyone that worked there, including the reporting team, was furloughed. One problem with this closure was that no one was left to run the reporting system and other parts of the business that still needed to function during the sale period. At least there was a reporting system.

I worked on a team that produces ad hoc reports for various departments in the major corporation. Since the reporting team was reassigned to my group before the sale was announced, maintaining this system fell to our team after the unit's office was closed. This system was developed in the mid-1990s and is still functioning today but its days are numbered. The transitional period after the sale will be complete by the end of 2011. Working with this system has given me new insights about the flexibility of SAS and its use as a reporting engine. This paper will describe the system, some of its features, and alternatives to it for others pursuing a similar task.

Anatomy of the Reporting System

The purpose of the reporting system is to produce and deliver reports to the analysts in the business. At the outset, reports were built as separate programs that were run on demand. Soon the reporting analysts were only taking calls from the business analysts and running reports. Although there are better solutions within SAS now, like building stored processes that can run reports on demand, when the reporting system was developed, these features were either unavailable or unfamiliar to the reporting analysts. After analyzing the use of the reports, the reporting analysts realized that customers usually demanded reports on a regular basis. To meet the reporting demands, the reporting analysts wrote SAS control programs to execute daily, weekly, and monthly eventually adding quarterly and annual reports to the mix. These control programs were all composed of three parts: an initialization stage, a report execution stage, and a report distribution stage.

The initialization stage was handled when the reporting program was invoked using a customized AUTOEXEC.SAS file, command line options, and a header section. Since there was no reliable scheduling system, the data step WAKEUP function was used in the header section to force the program to lay dormant until after midnight when the reporting programs needed to run. The report code execution was accomplished by calling each separate report with a %INCLUDE directive in the SAS code. At the end of each report a SAS macro was called that read a SAS format library that held distribution directives for each report and this macro generated SAS code that would eventually direct the report to its final location. At the end of the reporting run, the program containing the system-generated code was executed, distributing the reports.

Initialization

Creating a shortcut to the SAS executable and altering the command in the Target window of the Properties tab modified the command used to start SAS. Here is the command string:

```
"C:\Program Files\SAS\SAS 9.1\sas.exe"
     –config "C:\progra~1\SAS\SAS 9.1\nls\en\SASV9.CFG"
     –autoexec "i:\busdvp\dw_auto\autors.sas"
     –sasinitialfolder 'i:\busdvp\qsas'
```

The command line options used during the invocation of the SAS program and the purpose of each is given in Table 1.

Table 1: Command Line Options Used when Starting SAS

Command Line Option	Purpose
sasinitialfoler	open the named folder (which holds the reporting code), when an Open command is issued in the program editor
config	fully qualified path to the configuration file used to initialize SAS
autoexec	fully qualified path to the file containing SAS statements that are executed when SAS is involved

The AUTOEXEC.SAS file executed a set of LIBNAME statements to define the various libraries used by the system. Although it would have been efficient to put the date initialization section in the AUTOEXEC.SAS file, it was included in the header section of each control program. Figure 1 is a typical header section.

Figure 1: Header Section of a Typical Control Program

```
proc format;
       picture rundate other='0000/00/00';
       picture rundat  other='00000000';
       picture runda   other='000000' (fill='000000');
run;

FILENAME QSAS    'i:\busdvp\QSAS';         /* where sas jobs found */

data null;
   format date 8.;
   today = today();
   call symput ('today', compress(20000000 + put (today, yymmdd6.)));
   /* Verify if last day was a Saturday or a Sunday */
   if compress (put (today-1,downame15.)) not in ('Saturday' , 'Sunday') then
              jour = 1000000 + put (today-1,yymmdd6.);
   /* Else Verify if 2 days ago was a Saturday or a Sunday */
   else if compress (put (today-2,downame15.)) not in ('Saturday' , 'Sunday') then
              jour = 1000000 + put (today-2,yymmdd6.);
   /* Else Verify if 3 days ago was a Saturday or a Sunday */
   else if compress (put (today-3,downame15.)) not in ('Saturday' , 'Sunday') then
              jour = 1000000 + put (today-3,yymmdd6.);
   /* Check for Jan 01, 2000 */
   if today gt 14609 then ldate = jour + 20000000 - 1000000; /* Long  date */
   else                   ldate = jour + 19000000 - 1000000; /* Long  date */
   date = jour;                                              /* Short date */
   call symput ('ldate',ldate);
   call symput ('date',date);
   call symput ('rundate'  , put (ldate, rundate.));
   call symput ('rundate2' , put (ldate, rundat.));
   call symput ('rundate3' , put ((ldate - 20000000), runda.));
   call symput ('wmonth', put (today-1,monname.));
   call symput ('wyear ', put (today-1,year.));
   call symput ('cmonth', (190000 + int (jour / 100)));
   call symput ('ccmonth', compress ((190000 + int (jour / 100))));
   call symput ('cmonth2', (int (jour / 100)));
   call symput ('Worddate', put (input (ldate, yymmdd12.),worddate.));
   call symput ('slday', put (today, ddmmyy8.))  ;
   call symput ('p15sday', today() - 15);
   call symput ('p30sday', today() - 30);
run;
%put &today;

/* Assign Daily jobs libraries */
libname q886    'i:\busdvp\sasprod\q886';
libname canc    'i:\busdvp\canc';
libname crlm    'i:\busdvp\credit limit';
libname cacc    'i:\busdvp\cacc';
libname q041110 'i:\busdvp\sasprod\q041110';
libname q050101 'i:\busdvp\sasprod\q050101';
```

Waiting for the Right Time

The system relies on the SAS WAKEUP data step function to wait for the appropriate time to begin the reporting run. This function takes one argument, a SAS datetime value. When the function is called, execution of the SAS program pauses until the time passed to the function is reached after which the program resumes execution. With the addition of some code to automatically set the starting time for early the following morning, the task of starting the reports was simplified to opening a PC/SAS session, selecting the control program, and submitting it. The SAS data step used to control the starting of the jobs is shown is Figure 2.

Figure 2: The SAS WAKEUP Function

```
data START_TIME;
  start_dtg = put(today()+1,date5.)||put(today()+1,year4.)||':01:01:00';
  start_dtg2 = input(start_dtg,datetime18.);
  slept=wakeup(start_dtg2);
run;
```

Since the weekly and monthly jobs were not run as often, calculating the starting time was not automated in these systems. Rather, the analyst would enter the starting time before submitting the jobs.

Including the Reports

Figure 3 shows a SAS job that produces a typical report. Typically these reporting programs start with a query to the database followed by various data and procedure steps to produce the report. Since many of the reports were produced before the automated system was developed, they contain initialization sections that are redundant. When building such a system from scratch, it is more efficient to define commonly used parameters at the outset of the program.

Figure 3: A Typical Reporting Program

```
/*********** Regular enrolment *************************/

libname activ 'i:\busdvp\activation';

data _null_;
  today = today();
  call symput ('today', compress(20000000 + put (today, yymmdd6.)));
run;
%put &today &pros12ma;

proc sql;
  connect to odbc(dsn-"CAPCHAMP");
  create table pp  as select * from connection to odbc (
    select k0baci, k0prcc, k0cncd, k0pfsd, j1csid, j1acid, jycncd
         , jycand, jycbbc, jydols, jylngc, jyag01, JYag02, jyag03
         , jyblpr, jysadf, kotitl, kosexx, kocuss, kocusf, komnin
         , j9adty, j9efdt, j9corr, j9co30, j9hsbd, j9stnm, j9stna
         , j9dsnm, j9aadt, j9adrg, j9ctcd, j9bxno, j9apsc
      from tczpdtm050.tcupcpc
         , tczpdbf050.tcupsgf
         , tczpdtm050.tcupapd
         , tczpdtm050.tcuppcd
         , tczpdtm050.tcupcad
     where k0prco not in (3,4)
       and k0prcc  = 664
```

```
       and k0cncd  = '00'
       and k0csid  = j1csid
       and j1dtps >= &pros12ma
       and j1trcd in ('5004' , '5549')
       and j1acid  = jyacid
       and jycncd  = '00'
       and jycsid  = kocsid
       and jycsif  = j9csid
       and jysadf  = j9adty
  );
  disconnect from odbc;
quit;
```

All of the reports are initiated using %INCLUDE commands, which are pretty straightforward; however, the argument to the %INCLUDE directed can take several arguments. Table 2 shows the different types of arguments the %INCLUDE command can take. Like most directory based software systems, the reporting system referenced everything from a base directory, which could be changed if the underlying infrastructure changed. One paralyzing decision that was made early in the development process was to define SAS libraries in terms of drive letters instead of network locations. When designing a new system, it is best not to constrain the software to particular drive letters, if possible. To make the best of a bad construct, it is important to define all of the drive mappings in one commonly accessible location, like in the AUTOEXEC.SAS file, so that drive-mapping changes can all be made in one file. Another approach would be to start the programs with a DOS batch file that includes NET USE commands to unmap and map the appropriate shares before executing the SAS control programs.

Table 2: Different Forms of the %INCLUDE Command

Type	Example
External File	%include '/path/myfile.sas';
Fileref	%include myref;
Aggregate Fileref	%include dirref(myfile_in_dir);

Routing the Output

One of the best features of the reporting system is a centralized location for listing the report recipients and where each report is directed. This structure is accomplished using a SAS format to relate a report recipient to an ID, his name, location, and email address are stored in a SAS dataset that relates a recipient's ID to each report they receive along with the distribution type. This approach is akin to using a database tables to administer this functionality. Like using a database, the advantage of this approach is that there is a single place to administer the report recipients and the report distribution.

Figure 4 shows the first part of the SAS program that produces the user format.

Figure 4: Building a User Database Using a SAS Format

```
data names;
  input @   1 usern           $char12.
        @  14 email           $char40.
        @  55 user_f_name     $char13.
        @  68 user_l_name     $char13.
        @  86 organization    $char26.
        @ 112 location        $char15.
  ;
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+;
```

```
cards;
BILLY        William.Brewster@BigUn.com               William      Brewster    . . .
.
.
.
;

proc sql;
  create table email as
    select email      as label,
           usern      as start,
           'email'    as fmtname,
           'C'        as type
      from names
     where email contains '@'
    ;

  create table drive as
    select email      as label,
           usern      as start,
           'drive'    as fmtname,
           'C'        as type
      from names
     where email contains ':\'
    ;

  create table username as
    select distinct usern      as start,
                    username   as label,
                    'dcrptus'  as fmtname,
                    'C'        as type
      from names
    ;

  create table dcerprod.useremail as
    select email      as email_address,
           usern      as user,
           'email'    as fmtname
      from names
     where email contains '@'
     order
        by usern
    ;


  create table dcerprod.userdrive as
    select email      as drive,
           usern      as user,
           'drive'    as fmtname
      from names
     where email contains ':\'
     order
        by usern
    ;

  create table dcerprod.username as
    select distinct usern      as start,
                    username   as user,
                    'dcrptus'  as fmtname
      from names
    ;
quit;

proc format library=DW_FORM cntlin=email;

proc format library=DW_FORM cntlin=drive;

proc format library=DW_FORM cntlin=username;
```

The distribution data set relates the user ID to the report ID, its title, and the distribution type. Figure 5 shows the data step that relates each report to its recipients.

Figure 5: Relating Users to Reports in a SAS Data Step

```
data dcerprod.dc_rptsu;
  input @   1 type             $char7.
        @   8 rpt              $char15.
        @  24 user             $char12.
        @  37 rptname          $char75.
  ;
  if (type eq '') then delete;
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+;
cards;
Email  Q060317         BILLY        Q060317 Off-side Delinquency                . . .
```

These data structures are used by the SAS macro that distributes the reports. The reporting macro takes the report name as its only argument. Based on the information stored in the reporting dataset, the report output can be routed to one of three destinations: email, a LAN directory, or a hard copy report; however, one of the three destinations, the hard copy report, is no longer used. In a newly created reporting system, the number of destinations can be expanded to fit whatever reporting needs exist. The next section describes the routing macro in detail.

Under the Covers—How the Routing Macro Works

As shown in Figure 5, the reporting dataset contains four variables: type, the destination of the output, either drive (a LAN location), email, or banner (hardcopy output); rpt, the report name; user; a specific recipient of the report; and rptname, a descriptive name on the report. Sending multiple copies of the same report is accomplished by having one line of data for each report recipient.

The macro is essentially a data step that:

1. selects the recipients based on the report name,
2. takes the required action for the type of distribution, and, if necessary
3. generates code that will eventually route the output.

To select data for a particular report, a SET statement is used in conjunction with WHERE processing as shown in Figure 6.

Figure 6: Selecting All Records Associated with a Report

```
data null;
  length uuser $ 300.;
  retain uuser;
  set dcerprod.dc_rptsu (WHERE=(rpt=trim("&rpt"))) end=eof;
```

The next section of the data step is a series of conditional statements that account for each distribution type. Except for the hardcopy option, which is deprecated, each option generates SAS statements that, when executed, route the output appropriately.
For reports copied to a location on the LAN, the generated code is a two-step process. First the report's destination and location in the reporting repository is defined followed by using these definitions to write a DOS copy command. This section of code is shown in Figure 7:

Figure 7: Copying a Report to a Location on the LAN

```
else if type = 'Drive' then
  do;                          /* LAN Output */
    drv  = compress (put (user, $drive.) || "\&rept&rpt_date..*;");

    file BNRmail mod;
    put " /* &rpt " rptname " */; ";
    put " ";
    put "options xsync noxwait;";
    put "  "   ;
```

```
    put '%let from_file =' ;put "&qsasrpts\&rept&rpt_date..*;";
    put '%let to_file   =' ;put drv ;
    put " ";
    put "data _null_;";
    put '   %sysexec copy &from_file &to_file ;';
    put "run;";
    put " ";
  end;
```

If the report is to be sent via email, the macro builds a distribution list of all of the report recipients and then uses this list while generating code that will send the reports once all of the routing records for the report are processed. Figures 8 and 9 show the code used to send a report via email. Compiling the list is accomplished by appending the email address of each recipient to the list of all recipients.

Figure 8: Compiling a List of Email Recipients

```
else if type = 'Email' then
  do;                    /* Email output */
    usr = put (user, $email.);
    if uuser = ' ' then uuser = usr;
                   else uuser = trim (uuser) || ', ' || usr;
  end;
```

Once the list is compiled and there are no other records to processed, the flag set by the END= option on the set statement is used to conditionally trigger the generation of code that will send the report.

Figure 9: Code to Generate Code Used to Send Email Messges

```
/* Posting of email lists */
if eof and uuser ne ' ' then
  do;
    file BNRmail mod;
    put " /* &rpt " rptname " */; ";
    put " ";
    put "options xsync noxwait;";
    put " ";
    put '%let zip=.zip;';
    put '%let txt_file =' ;put "&qsasrpts\&rept&rpt_date..*;";
    put '%let txt_zip  =' ;put "&qsasrpts\&rept&rpt_date..zip;";
    put " ";
    put "data _null_;";
    put '   %sysexec wzzip -s&password -ycAES256 &txt_zip &txt_file ;';
    put "run;";

    put " ";
    put 'Data _null_; X=Sleep(1,1); run;';
    put " ";
    put "%"  "email (to='(" uuser ")', subject = " @;
    put rptname " &rpt_date2.
             ,attachment=&qsasrpts\&rept&rpt_date..zip);";
    put " ";
    put " ";
  end;
```

An example of this processing is shown in the next section. The entire routing macro, BANNER2, is listed in Appendix 1. Appendix 2 is the listing of the code embedded in the EMAIL macro, which is called when the generated code executes.

Seeing it in Action—An Example of Routing Output

A report; Q061203B, Daily Suspense File; is sent to three analysts: Larry, Sue, and Jesse. Sue needs to archive the report for audit purposes; therefore, she has her output is routed to a location on the network. She also needs a softcopy as she manages Larry and Jesse, who must take action on the items in the report, so all three have it delivered to their inbox. The first step is to enter each user's routing information into the program that relates that user's ID to it. Table 3 shows the values of the items associated with these users in the SAS dataset that is used to create the routing formats.

Table 3: Data Set Used to Build Formats for Distribution

USERN	EMAIL	USER_F_NAME	USER_L_NAME	ORGANIZATION	LOCATION
SUE	G:\RPTS	Sue	Swietzer	Operations	Memphis
SUE	sswietzer@a.com	Sue	Swietzer	Operations	Memphis
LARRY	ljones@a.com	Larry	Jones	Operations	Memphis
JESSE	jhivers@a.com	Jesse	Hivers	Operations	Memphis

Table 4 shows the observations in the report distribution data set used to build the program that will send the reports.

Table 4: Observations of the Routing Data Set Associated with the Daily Suspense Report

TYPE	RPT	USER	RPTNAME
Drive	Q061203B	SUE	Q061203B Daily Suspense File
Email	Q061203B	SUE	Q061203B Daily Suspense File
Email	Q061203B	LARRY	Q061203B Daily Suspense File
Email	Q061203B	JESSE	Q061203B Daily Suspense File

The PROC SQL code shown in Figure 4 shows how the characteristics of the LAN locations and the email addresses are used to select each type into separate datasets that are used to create the user defined character formats $DRIVE and $EMAIL used by the report distribution macro. Specifically, the presence of the sequence of characters, :\ in the variable EMAIL indicate the observation should be part of the $DRIVE format and an @ sign in the EMAIL variable selects the observation to be part of the $EMAIL format.

Given this structure, when the BANNER2 macro is called to distribute the Daily Suspense Report, the four observations associated with this report is the distribution dataset are selected. When Sue's request for a report copied to the LAN is encountered, her user ID is read using the $DRIVE format, which points to the drive and directory where the report is to be stored. The report name and date are used along with this information to create the DOS copy command created by the macro. In this case, the command is:

```
COPY I:\BUSDVP\DW_RPTS\Q061203B_20110908.XLS G:\RPTS
```

Note that this example shows the report is produced on September 8, 2011. Also, the value of the reporting directory is stored in the SAS Macro variable QSASRPTS.

When the observations containing the email addresses are encountered, only the email address portion is used and appended to the UUSER variable, which when the last report observation is processed, is used to address the email. The two sections of generated code are shown in Figure 10.

Figure 10: Code Generated by BANNER2, the Report Distribution Macro

```
/*************************************************
  Note: do not delete the password lines
  No need to Update password on 1st business day of month to current year month: dceryymm
  Password is set to automatically update.
*************************************************/

data null;
  today = today();
  call symput ('password', compress('dcer' || substr(put (today, yymmdd6.),1,4 )));
run;
%put &password;
run;

/*******DON'T DELETE ANYTHING ON OR ABOVE THIS LINE ******************************/

 /* Q061203B Daily Suspense File */;

options xsync noxwait;

%let from_file =i:\dw_rpts\Q061203B_20110908.*;
%let to_file   =g:\rpts\Q061203B_20110908.*;

data _null_;
   %sysexec copy &from_file &to_file ;
run;

 /* Q061203B Daily Suspense File */;

options xsync noxwait;

%let zip=.zip;
%let txt_file =i:\dw_rpts\Q061203B_20110908.*;
%let txt_zip  =i:\dw_rpts\Q061203B_20110908.zip;

data _null_;
   %sysexec wzzip -s&password -ycAES256 &txt_zip &txt_file ;
run;

data _null_; X=Sleep(1,1); run;

%email (to='(sswietzer@a.com, ljones@a.com, jhivers@a.com )'
       , subject = Q061203B Daily Suspense File  20110908
       , attachment =i:\dw_rpts\Q061203B_20110908.zip);
```

Executing the Code

Once all of the reporting programs ran, MAIL.SAS, a separate program, executes the generated code. The analysts' original design was to run this program the following day after checking to make sure that the jobs ran without error. Once our team was running the reports, the code was modified to send out the reports as the last step of the reporting program. Also, the code was modified to save a copy of the generated code to provide a log of the messages that were sent. Figure 11 shows the program that archives and executes the generated code. Within the reporting system, this program is executed by this command:

```
%include mail_timer_none.sas
```

Figure 11: Executing the Generated Code with a Separate Program

```
options xsync noxwait; /* command prmpt window closes automatically when
* application finishes and SAS waits for the
                        * application to finish
                        */


/* wait for right day and time  */
data test;
   today = today();
```

```
   time = time();
   time2 = put (time,hhmm9.);
        weekday2 = compress (put (today,downame15.)) ;
        today2 = compress(20000000 + put (today, yymmdd6.)) ;
      call symput ('today', compress(20000000 + put (today, yymmdd6.)));
      call symput ('time2', compress(put(time,hhmm5.),' :')) ;
   put today2 time2 weekday2 ;
run;


/* Save mail program of mail sent */
data null;
  call symput ('copystr',
               "I:\busdvp\dw_rpts\mail.sas
I:\busdvp\dw_rpts\mail_sent\mail_&today._&time2..sas");
run;
%put &today &copystr ;

data _null_;
   %sysexec copy /Y  &copystr;
   put "copy /Y &copystr";
   /* timer code to give copy a chance to finish */
   timea = time();
   do until (i=2);
      timeb = time();
      if timeb - timea >= 7 then  i = 2;
      else i=1;
   end;
run;

/* reset mail.sas */
data _null_;
  %sysexec copy /Y I:\busdvp\dw_rpts\mail_shell.sas I:\busdvp\dw_rpts\mail.sas;
  put "copy /Y I:\busdvp\dw_rpts\mail_shell.sas I:\busdvp\dw_rpts\mail.sas ";
run;

/* send out mail */

%INCLUDE "I:\busdvp\dw_rpts\mail_sent\mail_&today._&time2..sas";
```

Conclusion

The system designed by the reporting analysts was robust producing reports for the business unit for over fifteen years. It was modular, flexible, and had a table driven distribution system. In order to limit the amount space reports used, the system compressed them before distributing them. The design was robust and has filled a reporting need in a cost efficient way.

The system could be enhanced by extending the archival of the program that distributes the reports to other artifacts of each reporting run. For instance, there is no automatic archival of SASLOG files. Running the program through a batch process would be one way that a structure could be built to archive all of the log files. A batch process could also be programmed to check the archived logs for errors. Although compressing the reports is a nice feature, the system could be extended to deliver uncompressed reports, if they were under some threshold, which would allow the report recipient to directly open the reports from an email message.

Working with this system allowed me as a technical resource with a strong SAS background to monitor and maintain a suite of reports for a system that I knew little about. The design of the system is sound and could be used along with a collection of ad hoc reports as the basis for other reporting systems.

Appendix 1: The BANNER2 Macro

```
/* added date to subject line P R Hicks on 20081020  */

%macro banner2(rpt);
  %global file banner catalog report rept;
  option obs=MAX;

  filename BNRsas  "&qsasrpts\bnr.sas";  /* destination of SAS pgm for output */
  filename BNRmail  "&qsasrpts\mail.sas"; /* destination of SAS pgm for emails */
  filename BNRjob  "&qsasrpts";

  %LET REPT = &RPT;
  %LET FILE=FILENAME;
  %LET BANNER=BANNER;
  %LET CATALOG=CATALOG;
  %LET REPORT=REPORT;
  %LET B1=report.rp&prodtycm..banner.output;
  %LET B2="&file " "&banner " "&catalog " "'&B1'";
  %LET R1=report.rp&prodtycm..&rpt..output;
  %LET R2="&file " "&REPORT " "&catalog " "'&R1'";

  /* generate banner SAS job */

  data test;
     call symput ('rpt_date', '_' || compress (put (today(), yymmdd10.),'-'));
     call symput ('rpt_date2', compress (put (today(), yymmdd10.),'-'));
     /* Password */
     call symput ('password',
            compress('dcer'!!substr(compress(put(today(),yymmdd10.),'-'),3,4)));
  run;

%put password is &password. stop;

  data null;
    length uuser $ 300.;
    retain uuser;
    set dcerprod.dc_rptsu (WHERE=(rpt=trim("&rpt"))) end=eof;

    /* Regular Output -> hardcopy */
    if type = 'Banner' then do;
      file BNRSAS;
      xuser = put(user, $dcrptus.);
      put  @  1  "option obs=MAX;   ";
      put  @  1  "%let cat = report.rp&prodtycm..banner.output;    ";
      put  @  1  "proc printto print=&cat new;run;    ";
      put  @  1  "options linesize=210 pagesize=68 nodate nonumber;      ";
      put  @  1  "title;     ";
      put  @  1  "footnote;   ";
      put  @  1  "  data _null_;    ";
      put  @  1  "    set dcerprod.dc_rptcx    ";
      put  @  1  "      dcerprod.dc_rptsd (WHERE=(rpt=trim('&rpt') or type='DR'))     ";
      put  @  1  "      dcerprod.dc_rptsu (WHERE=(rpt=trim('&rpt')))      ";
      put  @  1  "      dcerprod.dc_rptcm    ";
      put  @  1  "      end=eof; ;    ";
      put  @  1  "    FILE PRINT n=ps LL=LL PS=55;   ";
      put  @  1  "    if _n_ = 1 then PUT _PAGE_;    ";
      put  @  1  "    if type = 'X'  then put @ 15 xdetails;    ";
      put  @  1  "    if type = 'X1' then do;  ";
      put  @  1  "      put @ 75 '------>' @ 90 '" xuser " '    ";
      put  @  1  "      //  @   1 '---------'"
      put  @  1  "          @  11 '---------'"
      put  @  1  "          @  21 '---------'"
      put  @  1  "          @  31 '---------'"
      put  @  1  "          @  41 '---------'"
      put  @  1  "          @  51 '---------'"
      put  @  1  "          @  61 '---------'"
      put  @  1  "          @  71 '---------'"
      put  @  1  "          @  81 '---------'"
      put  @  1  "          @  91 '---------'"
      put  @  1  "          @ 101 '---------'"
      put  @  1  "          @ 111 '---------'"
      put  @  1  "          @ 121 '---------'"
```

```
  put  @  1  "          @ 131 '---------'"
  put  @  1  "          @ 141 '---------'"
  put  @  1  "          @ 151 '---------'"
  put  @  1  "          @ 161 '---------'"
  put  @  1  "          @ 171 '----'";
  put  @  1  "    end;   ";
  put  @  1  "    if type = 'D'  then put @ 15 rdetails $char140.;      ";
  put  @  1  "    if type = 'DR' then put @ 15 rdetails $char140.;      ";
  put  @  1  "    if type = 'C'  then put @ 1 '-' @ 15 cdetails $char140. @ 175 '-'; ";
  put  @  1  "    if type = 'C1' THEN PUT  ";
  put  @  1  "          @   1 '---------'"
  put  @  1  "          @  11 '---------'"
  put  @  1  "          @  21 '---------'"
  put  @  1  "          @  31 '---------'"
  put  @  1  "          @  41 '---------'"
  put  @  1  "          @  51 '---------'"
  put  @  1  "          @  61 '---------'"
  put  @  1  "          @  71 '---------'"
  put  @  1  "          @  81 '---------'"
  put  @  1  "          @  91 '---------'"
  put  @  1  "          @ 101 '---------'"
  put  @  1  "          @ 111 '---------'"
  put  @  1  "          @ 121 '---------'"
  put  @  1  "          @ 131 '---------'"
  put  @  1  "          @ 141 '---------'"
  put  @  1  "          @ 151 '---------'"
  put  @  1  "          @ 161 '---------'"
  put  @  1  "          @ 171 '----'";
  put  @  1  "    end;   ";
  put  @  1  "    if type = 'D'  then put @ 15 rdetails $char140.;      ";
  put  @  1  "    if type = 'DR' then put @ 15 rdetails $char140.;      ";
  put  @  1  "    if type = 'C'  then put @ 1 '-' @ 15 cdetails $char140. @ 175 '-'; ";
  put  @  1  "    if type = 'C1' THEN PUT  ";
  put  @  1  "    if type = 'U'  then do; xuser = put(user, $dcrptus.);  ";
  put  @  1  "        put @ 40 xuser; end;    ";
  put  @  1  "    IF eof then do;    ";
  put  @  1  "        PUT # 75 @ 45 'Report produced by Global Systems & Technology';";
  put  @  1  "    end;   ";
  put  @  1  "    return;     ";
  put  @  1  "run;   ";
  put  @  1  "proc printto; run; /* resets back to normal */   ";

  put  @  1  &B2 ;

  put  @  1 ";";
  put  @  1  &R2 ;
  put  @  1 ";";

  put  @  1  "data _null_;    ";
  put  @  1  "  infile banner;     ";
  put  @  1  "  input;  ";
  put  @  1  "  file print notitle;      ";
  put  @  1  "  put _infile_;      ";
  put  @  1  "run;      ";

  put  @  1  "data _null_;    ";
  put  @  1  "  infile report;     ";
  put  @  1  "  input;  ";
  put  @  1  "  file print notitle;      ";
  put  @  1  "  put _infile_;      ";
  put  @  1  "run;      ";

  put  @  1  "option noxwait;    ";
end;

else if type = 'Drive' then do;          /* LAN Output */
  drv  = compress (put (user, $drive.) || "\&rept&rpt_date..*;");

  file BNRmail mod;
  put " /* &rpt " rptname " */; ";
  put " ";
  put "options xsync noxwait;";
  put "  "   ;
  put '%let from_file =' ;put "&qsasrpts\&rept&rpt_date..*;";
```

```
      put '%let to_file   =' ;put drv ;
      put " ";
      put "data _null_;";
      put '   %sysexec copy &from_file &to_file ;';
      put "run;";
      put " ";
    end;

    else if type = 'Email' then do;        /* Email output */
      usr = put (user, $email.);
      if uuser = ' ' then uuser= usr;
      else uuser = trim (uuser) || ', ' || usr;
    end;

    /* Posting of email lists */
    if eof and uuser ne ' ' then do;
      file BNRmail mod;
      put " /* &rpt " rptname " */; ";
      put " ";
      put "options xsync noxwait;";
      put " ";
      put '%let zip=.zip;';
      put '%let txt_file =' ;put "&qsasrpts\&rept&rpt_date..*;";
      put '%let txt_zip  =' ;put "&qsasrpts\&rept&rpt_date..zip;";
      put " ";
      put "data _null_;";
      put '   %sysexec wzzip -s&password -ycAES256 &txt_zip &txt_file ;';
      put "run;";

      put " ";
        put 'Data _null_; X=Sleep(1,1); run;';
      put " ";
      put "%"  "email (to='(" uuser ")', subject = " rptname @;
      put "&rpt_date2. ,attachment=&qsasrpts\&rept&rpt_date..zip);";
      put " ";
      put " ";
    end;

  run;
%mend;
```

Appendix 2: The EMAIL Macro

```
filename mail email ;
data _null_;
   file mail;
      * put '';
      put '!EM_TO!' "&to"
       / '!EM_cc!' "&cc"
       / '!EM_SUBJECT!' "&subject"
       / '!EM_ATTACH!'  "&attachment"
       / "Please review the attached report. If you discover any issues
then reply to this message as soon as possible with your concerns. "
       / "Otherwise no response is required. "
       / " "
       / "Thanks "
       / "Diners Canada &dept. "
       / '!EM_SEND!'
       / '!EM_NEWMSG!'
       / '!EM_ABORT!';
    %DTGStamp ;
run;

%mend;
```