Paper S105-2011

Utilizing SAS® for Complete Report Automation

Brent D. Westra, Mayo Clinic, Rochester, MN

ABSTRACT

Analysts in every field devote a great deal of time and effort to the tedious, manual task of creating the same report over and over again. This paper describes how you can use SAS to completely automate a daily operational report. We will explore SAS methods used for accessing data sources, transforming data, and creating attractive reports. Once the reporting logic is built, we will use the EMAIL file type and the Windows Scheduler to completely automate the report delivery process. Through this example, the goal is to provide a framework that other analysts can use to eliminate time spent in the creation of recurring reports.

INTRODUCTION

Imagine (I'm sure not for the first time) you're a reporting analyst working on the payer side of the healthcare industry with a focus on medical claims data. You've been asked to create a Microsoft Excel report of all Emergency Room (ER) claims that have not yet been paid, where there are multiple dates of service. The goal is to identify situations where a patient might be incorrectly charged an extra copayment because his/her emergency room visit spanned more than one day. (The exact nature of the request or the industry in which you work is not important. This example will merely serve as an illustration of the power SAS can provide in automating recurring reports.)

This request is straightforward and relatively simple to complete as an ad hoc report. Once you know the appropriate codes for identifying ER claims, querying these data from a data warehouse takes only a matter of minutes. From there you just need to export your output into Excel, format it, create titles and footnotes, save the file with an appropriate name and date, do some data quality checks, and finally email it out. All of this is routine and might only take a half hour in total to complete. No problem, right? Right...until management tells you they would like to receive this report every day. Now you are faced with the frightening prospect of having to performing these same steps to create this report every business day until your employment ends. At 30 minutes a day for 250 business days a year, this amounts to an astounding 125 hours spent per year on this single simple report request!

The problem is compounded by future requests for more daily reports based on different criteria. Before you know it, you might be spending half your time mindlessly churning out reports you could do in your sleep. Fortunately with the help of SAS, it is possible to automate the entire process, and in fact literally create and distribute these reports while you are sleeping.

Using the ER report from above as an example, I will show you a SAS automation process which includes:

- Connecting to a data source
- Querying data using PROC SQL
- Creating attractive output with PROC REPORT
- Using the Output Delivery System (ODS) and the ExcelXP Tagset
- Producing macro variable date labels for use in titles, filenames, and emails
- Checking for errors
- Printing the log
- E-mailing the report
- Scheduling the job with Windows Scheduler

CONNECTING TO A DATA SOURCE

For this example, we will use SAS/ACCESS to connect to a relational database management system via Open Database Connectivity (ODBC). The following syntax assigns the library name, "Claims", which provides a pointer to the files in the medical claims warehouse.

```
LIBNAME CLAIMS odbc
        NOPROMPT="DSN=Claims_DSN"
        SCHEMA=CLAIMS;
```

The NOPROMPT option allows your user name and password credentials to be picked up by the local ODBC connection settings on your system. When the LIBNAME statement is executed, SAS automatically looks at your system to see who is logged on. When you are logged into your computer, everything works great because the credentials for the ODBC data source can be retrieved from the system. However, since our goal is to automate the report delivery process, we prefer to not be logged on to our computer at the time the SAS program is executed.

One option to allow connection to your data source when you are not logged on is to hard code your user name and password into the LIBNAME statement.

```
LIBNAME CLAIMS odbc
        DSN=Claims_DSN
        UID= bwestra
        PASSWORD = "iluvjerseyshore"
        SCHEMA=CLAIMS;
```

Unfortunately, while this method is effective, there is the potential security issue of hard coding your password, particularly if it is a program that is used by others in your department or if it is saved in a public directory. Worse still, in the case of the above code, passwords can be deeply personal and could present embarrassment should your co-workers become aware your password.

To preserve its secrecy, you can use PROC PWENCODE to create an encoded value for your password. Run the following code in a new program on your C:\ drive or personal network drive. The encoded value of your password gets printed to the log.

```
PROC PWENCODE in='iluvjerseyshore';
RUN;
```

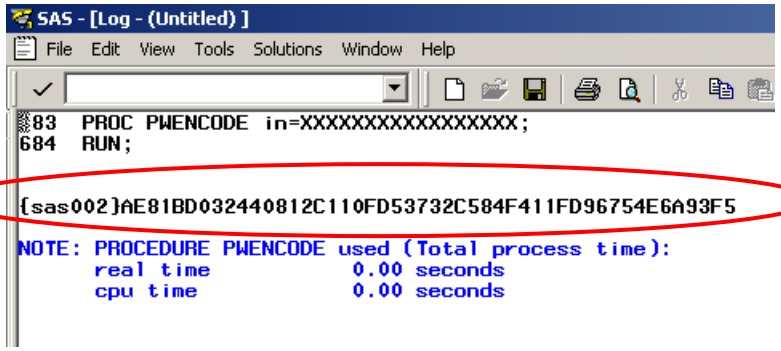


Figure 1: Log output from PROC PWENCODE

Next, take the encoded value from the log window and assign it as a new variable called "Pass" in the Local.password data set.

```
LIBNAME Local 'C:\';
DATA Local.password;
Pass = '{SAS002}AE81BD032440812C110FD53732C584F411FD96754E6A93F5';
RUN;
```

Your encoded password can now be converted to a macro variable and referenced for use in your main program.

```
LIBNAME Local 'C:\';
PROC SQL;
SELECT Pass INTO :password FROM Local.password;    /*Creates a macro variable called &password*/
QUIT;


LIBNAME CLAIMS odbc
        DSN=Claims_DSN
        UID= bwestra
        PASSWORD = "&password"    /*Make sure the macro variable is in double quotes*/
        SCHEMA=CLAIMS;
```

In addition to the obvious preference of not displaying your password within your program, using PWENCODE eliminates the need to change your hard coded password within each program where it is used. Simply run your new password once through the encoding process described above, and use the same macro variable for your password within all relevant programs.

QUERYING DATA USING PROC SQL

Once connection to the data source has been successfully established, we can query the tables needed to produce the requested information. We know from the report request that we need to identify ER medical claims that have not yet been paid. We also are only interested in claim lines that have a copayment amount.

```
PROC SQL;
  CREATE TABLE Work.Claims_01 AS
  SELECT  Companyno, Memberno, Claimnumbr, Servicedat, Copay
  FROM    Claims.Detail
  WHERE  Paiddate = ' '   AND Code = 'ER'    AND Copay gt 0;   /*The paiddate field in this table is a string variable*/
QUIT;
```

From there, we need to identify which of these ER claims have multiple service dates. To do this, we can count the unique number of service dates listed for each claim, and then using the "HAVING" clause, keep only those claims that have two or more different service dates.

```
PROC SQL;
  CREATE TABLE Work.Claims_02 AS
  SELECT Claimnumbr, Count(DISTINCT(Servicedat)) AS Total_Dates
  FROM   Work.Claims_01
  GROUP BY Claimnumbr
  HAVING Count(DISTINCT(Servicedat)) ge 2;
QUIT;
```

Now we have identified a list of relevant claim numbers for this analysis. The last PROC SQL step joins the Claims_02 and Claims_01 data sets from above, in order to add back the company, member, service date, and copayment detail to the list of identified claims numbers.

```
PROC SQL;
  CREATE TABLE Claims_03 AS
  SELECT b.*
  FROM Claims_02 a LEFT JOIN Claims_01 b
  ON a.Claimnumbr = b.Claimnumbr
  ORDER BY Servicedat;
QUIT;
```

The resulting data set contains all of the information needed to complete the report request.

	COMPANYNO	MEMBERNO	CLAIMNUMBR	SERVICEDAT	COPAY
1	4	123456	508756315472	01MAY2011	50
2	4	123456	508756315472	02MAY2011	50
3	3	789123	456456446556	29MAY2011	50
4	3	789123	456456446556	30MAY2011	50
5	9	456789	879215645648	03JUN2011	50
6	9	456789	879215645648	04JUN2011	50
7	5	234567	542157745444	08JUL2011	25
8	5	234567	542157745444	09JUL2011	25

Figure 2: Data set view of claims identified in the PROC SQL queries

CREATING ATTRACTIVE OUTPUT WITH PROC REPORT

Now that our data set contains all the requested information, we need to find a way to display this data in a more attractive fashion. This includes adding cleaner labels to the headings, creating groups of claim numbers, adding spacing between the groups, inserting a subtotal line that sums the copayment amounts by group, and finally highlighting and bolding the subtotal line. There are a variety of procedures that could be used to accomplish these tasks including PROC PRINT, PROC TABULATE, and PROC SQL, but I've found PROC REPORT to be the most effective tool for report creation.

```
PROC REPORT DATA = Claims_03 NOWD;
  COLUMN Companyno Memberno Claimnumbr Servicedat Copay;   /*Identifies columns to be used in report*/
  DEFINE Companyno  / GROUP  'Company';
  DEFINE Memberno   / GROUP  'Member Number'
  DEFINE Claimnumbr / GROUP  'Claim Number'   STYLE={tagattr='format:text'}; /*Tag Attributes are for Excel formats*/
  DEFINE Servicedat / GROUP  'Service Date';
  DEFINE Copay      /ANALYSIS 'Copay'    /*Summary column*/     STYLE={tagattr='format:$#,##0.00'};

  COMPUTE AFTER Claimnumbr;
        LINE ' ';         /*Creates a blank line between groups*/
        ENDCOMP;

  BREAK AFTER Claimnumbr /SUMMARIZE  style(summary)   ={font_weight = bold background = yellow};
RUN;
```

The following report gets printed to the SAS output window. The column headings are cleaner, data have been grouped together by claim number, and copayment amounts have been summarized. There is even a space that has been inserted between the groups as a result of the creation of a blank variable called "Line" in the Compute block.

Company	Member Number	Claim Number	Service Date	Copay
3	789123	4.5646E11	29MAY2011	50
			30MAY2011	50
3	789123	4.5646E11		100
4	123456	5.0876E11	01MAY2011	50
			02MAY2011	50
4	123456	5.0876E11		100
5	234567	5.4216E11	08JUL2011	25
			09JUL2011	25
5	234567	5.4216E11		50
9	456789	8.7922E11	03JUN2011	50
			04JUN2011	50
9	456789	8.7922E11		100

Figure 3: PROC REPORT output as viewed in the SAS Output window

Unfortunately, those are about the only positive things that can be said about this output. Most of our formatting efforts were unsuccessful. The claim numbers are displayed hideously in scientific notation. Dollar signs and decimals are absent from the copay amounts. The yellow background and bold style requested for the summary lines is completely M.I.A. Furthermore, our goal was not to have the final output displayed in SAS output window, but rather to be displayed in Microsoft Excel.

Now, if only there was a system for delivering our output to Microsoft Excel...

USING THE OUTPUT DELIVERY SYSTEM (ODS) AND THE EXCELXP TAGSET

Simply put, the Output Delivery System (ODS) is the method by which SAS data can be displayed in formats outside of SAS. Some of the more common formats include Portable Document Format (PDF), Rich Text Format (RTF), Comma Separated Values (CSV), HyperText Markup Language (HTML), and Extensible Markup Language (XML). While CSV

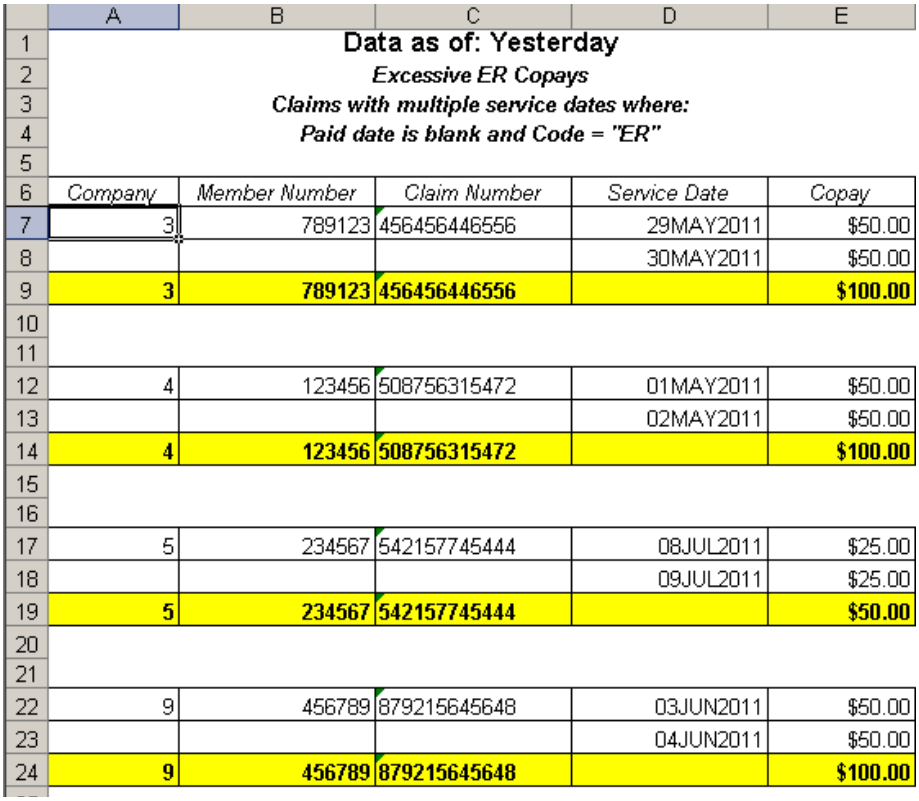
and HTML both work effectively with Microsoft Excel, using XML through the ODS ExcelXP Tagset provides the most flexibility in customizing the format of the final Excel output.

Almost any setup that can be done with the Page Setup window of Microsoft Excel, can be implemented using ExcelXP Tagset. These options include, but are not limited to: page orientation, headers and footers, frozen headers, row repeat, auto filtering, and column widths.

Some other features such as titles and print area margins also work seamlessly with ExcelXP Tagset but are actually SAS options rather than ExcelXP options, and thus can be indicated prior to opening the ODS destination. To view these SAS titles within your final Excel document, you need to specify Embedded_Titles = 'Yes' within the ExcelXP Tagset options. The ODS destination is opened using **ODS TAGSETS.EXCELXP** prior to running the procedures that produce your output (in this case PROC REPORT) and closed using **ODS TAGSETS.EXCELXP CLOSE** when you want to stop printing the output.

```
TITLE1 Height = 13pt bold 'Data as of: Yesterday';     /*Each Title statement will appear on a new row in your document*/
TITLE2 'Excessive ER Copays';
TITLE3 'Claims with multiple service dates where:';
TITLE4 'Paid date is blank and Code = "ER" ';

OPTIONS TopMargin = .75in LeftMargin = .5in RightMargin = .5in;   /*Sets print margins in Excel*/

ODS TAGSETS.EXCELXP    /*Opens the ExcelXP Tagset destination*/
  FILE = 'C:\ER Excessive Copays.xls'    /*Identifies file name and location for output*/
  STYLE = JOURNAL             /*To see list of available styles, type ODSTEMPLATE in the SAS command line */
  OPTIONS
   (Sheet_Name = 'Excessive ER Copays'
    Orientation='Portrait'
    Frozen_Headers='6'           /*Column headers freeze for scrolling at row 6*/
    Row_Repeat='6'               /*Column headers in row 6 will print on each page*/
    Embedded_Titles = 'yes'      /*Embeds Titles within the document*/
    Print_Footer = '&amp;amp;CPage &amp;amp;P of &amp;amp;N'  /*Prints footer page numbers in the "Page 1 of 2" format*/
    Absolute_Column_Width = '8,12,12,12,9') ;   /*Sets column width for columns in report.  Requires some trial and error*/


PROC REPORT DATA = Claims_03 NOWD;
  COLUMN Companyno Memberno Claimnumbr Servicedat Copay;
  DEFINE Companyno  / GROUP  'Company';
  DEFINE Memberno    / GROUP  'Member Number'
  DEFINE Claimnumbr  / GROUP  'Claim Number'   STYLE={tagattr='format:text'};
  DEFINE Servicedat     / GROUP  'Service Date';
  DEFINE Copay          /ANALYSIS 'Copay'    STYLE={tagattr='format:$#,##0.00'};

  COMPUTE AFTER Claimnumbr;
          LINE ' ';
          ENDCOMP;

  BREAK AFTER Claimnumbr /SUMMARIZE  style(summary)   ={font_weight = bold background = yellow};
RUN;

ODS TAGSETS.EXCELXP CLOSE;    /*Closes the ExcelXP Tagset destination.  Printing to the .xls file stops*/
```

The output from this code looks much better now (Figure 4). The tag attribute (tagattr) format for claim number specified an Excel text format, so when viewed in Excel those numbers appear as text and not in scientific notation. Similarly, the tag attribute for copay created an Excel format that includes dollar signs and decimals. The subtotal lines are now bolded and highlighted in yellow.



	A	B	C	D	E
1	Data as of: Yesterday				
2	*Excessive ER Copays*				
3	*Claims with multiple service dates where:*				
4	*Paid date is blank and Code = "ER"*				
5					
6	*Company*	*Member Number*	*Claim Number*	*Service Date*	*Copay*
7	3	789123	456456446556	29MAY2011	$50.00
8				30MAY2011	$50.00
9	**3**	**789123**	**456456446556**		**$100.00**
10					
11					
12	4	123456	508756315472	01MAY2011	$50.00
13				02MAY2011	$50.00
14	**4**	**123456**	**508756315472**		**$100.00**
15					
16					
17	5	234567	542157745444	08JUL2011	$25.00
18				09JUL2011	$25.00
19	**5**	**234567**	**542157745444**		**$50.00**
20					
21					
22	9	456789	879215645648	03JUN2011	$50.00
23				04JUN2011	$50.00
24	**9**	**456789**	**879215645648**		**$100.00**

Figure 4: ODS ExcelXP Output

The remainder of the formatting and appearance is determined by the use of a style template called "Journal" as indicated in the preceding code. SAS offers a wide selection of stock style templates. To view a list of these styles, type ODSTEMPLATE in the SAS command line, and then view the styles folder located under sashelp.tmplmst.

A really nice feature that provides flexibility in your output's appearance is the ability to modify these styles to create your own custom style. In the output above, there are several small things that I would like to change. I would prefer the titles to be left justified with Arial font, for all the data within the cells to be right justified, for the column headings to be bold and without italics, and for the cell borders to be gray instead of black. Using PROC TEMPLATE, we can create a new style called "Mayo" that modifies the Journal style to invoke these preferences.

```
PROC TEMPLATE;
DEFINE STYLE Styles.Mayo;   /*New Style name*/
Parent = Styles.Journal;          /*Style you are basing new style on*/
          CLASS SystemTitle  / Font = ("Arial", 11pt, Bold) Just = Left;   /*Changes title style*/
          CLASS Data   / Just = Right;                                     /*Changes data style*/
          CLASS Header   / Font = ("Arial", 11pt, Bold )                   /*Changes column heading style*/
          CLASS Table  /  Bordercolor = Lightgray                          /*Changes cell border style*/
END;
RUN;
```

After creating the new style "Mayo", simply replace the Journal style with your new style in the STYLE statement of the ODS TAGSETS.EXCELXP code.

```
ODS TAGSETS.EXCELXP
  FILE = 'C:\ER Excessive Copays.xls'
  STYLE = Mayo        /*Formerly  "Journal"*/
```

The differences between using the new style (Figure 5) and the old style (Figure 4) are subtle, yet it is significant to demonstrate that by using PROC TEMPLATE, you have complete control of the customization of your final output.

	A	B	C	D	E
1	**Data as of: Yesterday**				
2	**Excessive ER Copays**				
3	**Claims with multiple service dates where:**				
4	**Paid date is blank and Code = "ER"**				
5					
6	**Company**	**Member Number**	**Claim Number**	**Service Date**	**Copay**
7	3	789123	456456446556	29MAY2011	$50.00
8				30MAY2011	$50.00
9	**3**	**789123**	**456456446556**		**$100.00**
10					
11					
12	4	123456	508756315472	01MAY2011	$50.00
13				02MAY2011	$50.00
14	**4**	**123456**	**508756315472**		**$100.00**
15					
16					
17	5	234567	542157745444	08JUL2011	$25.00
18				09JUL2011	$25.00
19	**5**	**234567**	**542157745444**		**$50.00**
20					
21					
22	9	456789	879215645648	03JUN2011	$50.00
23				04JUN2011	$50.00
24	**9**	**456789**	**879215645648**		**$100.00**

Figure 5: ODS ExcelXP Output using new style created with PROC TEMPLATE

MACRO VARIABLE DATE LABELS FOR USE IN TITLES, FILENAMES, AND EMAILS

There is only one more thing that I would like to change about the preceding output. Instead of having the title say "Data as of: Yesterday" (technically correct), I would rather show yesterday's actual date in that title.

To identify yesterday's date we can subtract one day from the Today() function.

```
Yesterday = Today() - 1;          /*An equivalent function is the Date() function.  i.e.;   Yesterday = Date() - 1*/
```

To display this date within filenames and titles, we need to convert it to a macro variable. We can use CALL SYMPUT to create a macro variable called "Yesterday".

```
DATA _NULL_;
CALL SYMPUT('Yesterday', Today()-1);
RUN;
```

This code generates a SAS date value of 18857. In SAS date value lingo, this means that yesterday took place 18,857 days since January 1, 1960. Even if you are a regular SAS user, displaying a date in this format could make your head hurt. We need to change the format of this date to something everyone can immediately read and comprehend.

```
DATA _NULL_;
CALL SYMPUT('Yesterday', put((Today()-1),Date9.));
RUN;
```

By modifying the code to use the Date9. format, the result is a macro variable with the more palatable value of 18AUG2011. There are dozens of date formats to select from besides Date9. You should be careful that the date format you select works for its intended purpose. For example the format MMDDYY10. produces a value of 08/18/2011 which, while it may have a preferred appearance, cannot be used in an Excel filename, because Excel does not allow the use of the slash "/" in a filename.

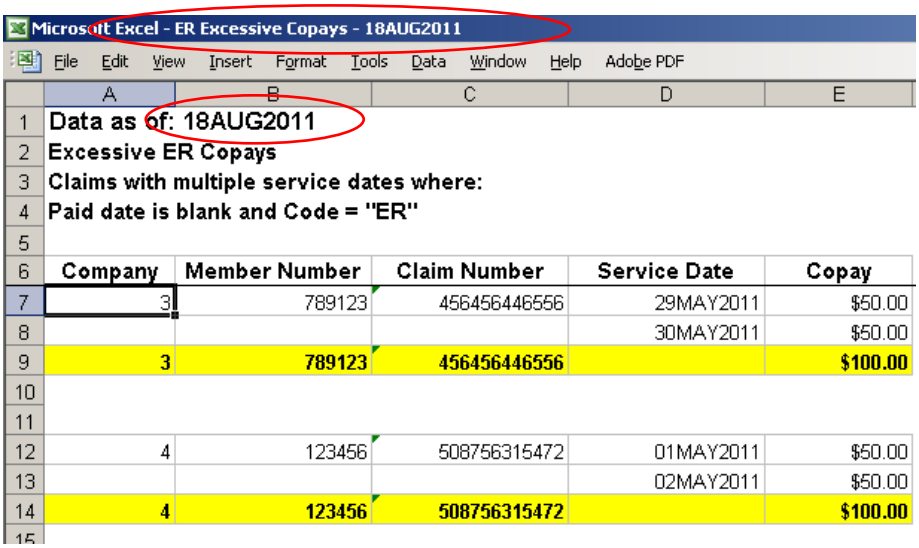
To create the Excel filename and the location where the file will be stored on your computer, we need to create yet another macro variable called &Filename that concatenates the path of the report (C:\), the report title (ER Excessive Copays), the resolved macro variable &Yesterday (18AUG2011) and the file extension (.xls).

```
DATA _NULL_;
CALL SYMPUT('Filename', '"' ||'C:\ER Excessive Copays'||" - "||RESOLVE('&Yesterday')||".xls"');
RUN;
```

The macro variable &Filename now contains the value: **'C:\ER Excessive Copays - 18AUG2011.xls'**

The next step is to insert the &Yesterday and &Filename macro variables into the Title and ODS statements, respectively, in order to produce output that is customized with yesterday's date labels.

```
TITLE1 HEIGHT = 13pt bold 'Data as of: ' &Yesterday;          /*Use macro variable date for yesterday */
TITLE2 'Excessive ER Copays';
TITLE3 'Claims with multiple service dates where:';
TITLE4 'Paid date is blank and Code = "ER" ';

OPTIONS TopMargin = .75in LeftMargin = .5in RightMargin = .5in;

ODS TAGSETS.EXCELXP
 FILE = &Filename            /*Filename and location for output retrieved by resolving macro variable &Filename*/
 STYLE = Mayo
 OPTIONS
  (Sheet_Name = 'Excessive ER Copays'
   Orientation='Portrait'
   Frozen_Headers='6'
   Row_Repeat='6'
   Embedded_Titles = 'yes'
   Print_Footer = '&amp;amp;CPage &amp;amp;P of &amp;amp;N'
   Absolute_Column_Width = '8,12,12,12,9') ;
```



	A	B	C	D	E
1	**Data as of: 18AUG2011**				
2	**Excessive ER Copays**				
3	**Claims with multiple service dates where:**				
4	**Paid date is blank and Code = "ER"**				
5					
6	**Company**	**Member Number**	**Claim Number**	**Service Date**	**Copay**
7	3	789123	456456446556	29MAY2011	$50.00
8				30MAY2011	$50.00
9	**3**	**789123**	**456456446556**		**$100.00**
10					
11					
12	4	123456	508756315472	01MAY2011	$50.00
13				02MAY2011	$50.00
14	**4**	**123456**	**508756315472**		**$100.00**

Figure 6: ODS ExcelXP Output with yesterday's actual date in file name and report title

Lastly, it is important to note that this example works when you are not logged on because it is saving the output to the C:\ drive. If you were trying to save to a mapped drive letter (e.g.: M:\claims\bdw), it would fail because the mapped drives in your user profile are not available to the operating system when you are logged off. The workaround for this is to use a universal naming convention (UNC), for the location of your network drive (e.g.:\\uwec\msi\tic\claims\bdw).

CHECKING FOR ERRORS

Now that the output is formatted precisely the way we want, we are almost ready to schedule this report to be emailed out. However, we would not want to email this report without validating that the SAS coding is error free and that the data are correct. Say, for example, there was a syntax error in our code that produced an error message in the SAS Log window. This error could cause our data to be incomplete, or even blank. Without any controls written into our code to prevent this, we very well could end up emailing erroneous data and not realize it until we receive an irritated response from one of the email recipients.

Perhaps the easiest way to avoid this issue is to activate an option called ERRORABEND that will cause SAS to abort if any errors are written to the log. Enter the following code at the beginning of your SAS program:

```
OPTIONS ERRORABEND;
```

From this point forward until your SAS session ends or until you specify otherwise, any time a syntax error occurs, SAS will immediately shut down.

To turn off this option, simply enter the following:

```
OPTIONS NOERRORABEND;
```

I recommend turning off ERRORABEND at the end of your automated program. There may come a time when you need to run one of your automated programs interactively. If you turn on the ERRORABEND feature and forget to turn it off, you may be surprised later when your SAS session abruptly ends, due to an error unrelated to your automated program. You will not receive a "Do you want to save?" message with ERRORABEND, so there is a real possibility of losing unsaved work if the feature is not turned off.

Using the ERRORABEND option works well for avoiding bad output that is caused by SAS syntax errors, but what happens when the incorrect output is the result of poor quality data? While there a multitude of different constraints you could put around your data to ensure cleanliness, I am going to show you one very simple example so you can get an idea of what is possible.

In the work.claims_03 data set created for the ER report, there is a column that contains copay dollar amounts. Let's say that we know that there should never be a copay amount greater than $200.00. We can build logic into an IF/THEN statement within the DATA step, to tell SAS to shut down if any copay record has a value greater than $200.00.

```
DATA _NULL_;              /*Use DATA _NULL_ if you do not need to create a new data set.*/
 SET work.claims_03;
 IF Copay > 200 THEN ABORT ABEND;
RUN;
```

Using ABORT by itself will only end processing within that DATA step. Processing of subsequent DATA steps and procedures will continue. In this case, we would rather have SAS halt all processing so we can avoid sending out erroneous data. To terminate the SAS session, it is recommended to add the ABEND command in conjunction with ABORT.

PRINTING THE LOG

Usually in SAS, when you encounter errors, the first place you go to troubleshoot those errors is the SAS Log window Not only does the log tell you where in the coding the error occurred, but it also provides a decent explanation of the cause of the error. However, the default setting makes the log viewable only during your active SAS session. How then, are we to consult the log for error explanations if our SAS session ends because of those errors?

Not surprisingly, SAS offers the capability to print the log to an external file. Near the beginning of your program, run the following PROC to initiate printing the log to an external text file. Until the SAS session ends or until otherwise specified, the log will print to this text file instead of the SAS Log window.

```
PROC PRINTTO LOG='C:\Excessive ER Copy Log.txt';
```

Maintaining a record of each daily log could be beneficial for auditing purposes. To avoid overwriting the prior versions of the external log, you can add the date to each log filename. This is a similar process to what we did earlier when we used the macro variable &Yesterday within the Excel filename.

```
DATA _NULL_;
CALL SYMPUT('Log_File', '"C:\ER Excessive Copay Log - " || RESOLVE('&Yesterday') ||".txt"');
RUN;
```

The macro variable &Log_File now contains the value: **'C:\ER Excessive Copay Log - 18AUG2011.txt'**

Simply use this macro variable in the PROC PRINTTO statement and you will now maintain a record of the external log file for each day the automated program is executed.

```
PROC PRINTTO LOG=&Log_File;
```

Any time you want to stop printing to the external file and resume printing to the SAS Log window, you can either start a new SAS session or execute the following code:

```
PROC PRINTTO;
RUN;
```

E-MAILING THE REPORT

Once we are confident that our data has no errors or quality concerns, we can send the file by using the EMAIL access method. By using EMAIL as the access method associated with the "myemail" fileref, we are unlocking all the SAS email options. We can specify who the email is to, who it is from, if we want to "copy" any recipients on the email, add a subject, attach files, and add a message to the body of the email. Essentially, it does everything a normal email does, without ever having to leave the confines of SAS.

Note that in the following code, we continue to make use of previously created macro variables &Yesterday and &Filename in order to attach the Excel file and to display yesterday's date in the subject and body of the email.

```
FILENAME myemail EMAIL;
DATA _NULL_;
FILE myemail;
TO= ("jsmith@mayo.edu" "manderson@mayo.edu")
CC= "bwestra@mayo.edu"
FROM="Brent Westra <bwestra@mayo.edu>"
SENDER="Brent Westra <bwestra@mayo.edu>"
SUBJECT="Excessive ER Copays: &Yesterday" /*Use macro variable date label*/
ATTACH= (&Filename);    /*Use macro variable &Filename to retrieve location of the file*/

PUT "Attached are the ER Claims with excessive copays as of &Yesterday";    /*Use macro variable date label*/
PUT  " ";   /*Creates a blank line*/
PUT  "Please let me know if you have any questions.  Thanks,";  /*Each 'PUT' statement creates a new line in the email*/
PUT " ";
PUT  "Brent";
RUN;

FILENAME myemail CLEAR;
```

When sending emails from SAS, you should "CC" yourself so that you can be certain the email was sent with no trouble. If the code above successfully sends out the email, then that is fantastic. You are ready to move on to scheduling your SAS program to run daily. However, it is more likely that when you execute your email code, you receive the following message from Microsoft Outlook (Figure 7).

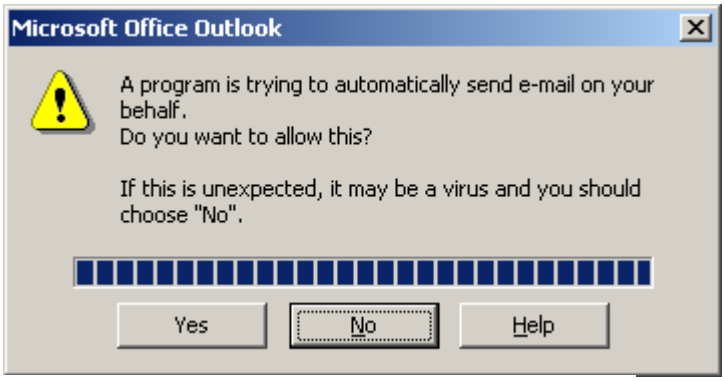


Figure 7: Microsoft Office Outlook error message

This is simply a security measure to make sure your computer has not unwittingly been turned into a spam robot. All you have to do is click "Yes" and the email is sent, so this is not a big deal. WRONG. Actually, it is a big deal. Remember, we are trying to *automate* this entire process. You will not be able to email your report at 4:00 AM while you are sleeping, if you are required to click "Yes" on this error message in order to complete the job. More importantly, I would be forced to change the title of this paper to "Utilizing SAS for *Partial* Report Automation."

Since I would prefer not to do that, there are a couple of options for avoiding this dialog box. The first option is to contact your email administrator and find out if they can turn off the security settings that are causing the error message. If your email administrator is not permitted to alter those settings, your second option is to change some of the SAS email settings so that the email is sent via a Simple Mail Transfer Protocol (SMTP) server instead of your local email client (i.e.; Microsoft Outlook).

For SAS 9.2, the settings to change are located in:

C:\Program Files\SAS\SASFoundation\9.2\nls\en\sasv9.cfg

Open the sasv9.cfg file using Notepad. At the top of the document type the following three SMTP email options:

-EMAILSYS SMTP

-EMAILHOST *your.smtpemail.server.com*

-EMAILPORT 25

Your email administrator should be able to provide the name of the SMTP server to use for EMAILHOST. For more details on these configuration changes, refer to the SAS Usage Note *19767: "* Using the SAS® System to send SMTP e-mail" at http*://support.SAS.com/kb/19/767.html, updated 2007*.

Once you've made these changes, save the configuration file. You'll also want to start a new SAS session to allow these new settings to take effect. The next time you run the email code, your email will be sent successfully without a dialog prompt thwarting your ultimate goal of complete report automation.

Before moving on to scheduling the job, there is one more concern related to sending out the email. What happens if the final output has zero records? In the ER example, if there are no claims that meet the criteria for the report, a blank Excel document would be created and emailed out. Sending out a blank document is not very professional and it is certainly not something you would do if you were performing this task manually.

To skirt this issue, you can use conditional logic to control the email you send. If there are no records, you would still like to send an email that explains in the body that there were no ER claims meeting the criteria. To facilitate this process, you need to first create a macro variable that stores a count of total records in the final data set.

```
PROC SQL NOPRINT;
        SELECT Count(Claimnumbr) INTO: Rec_Count
        FROM Claims_03;
QUIT;
```

The macro variable &Rec_Count can now be used in a conditional email statement to determine which email message to send.

```
FILENAME myemail EMAIL;

%MACRO sendmail;                        /*The conditional email is done within a macro program*/

%IF &Rec_count = 0 %THEN %DO;
        DATA _NULL_;
        FILE myemail
        TO= ("jsmith@mayo.edu" "manderson@mayo.edu")
        CC= ("bwestra@mayo.edu ")
        FROM="Brent Westra < bwestra@mayo.edu >"
        SENDER="Brent Westra < bwestra@mayo.edu >"
        SUBJECT="Excessive ER Copays: &Yesterday";
                                                        /*No Attachment if there are zero records*/
PUT "There are no claims with Excessive ER copays as of &Yesterday";   /*Message for cases with zero records*/
PUT " ";
PUT "Please let me know if you have any questions.  Thanks,";
PUT " ";
PUT "Brent";
RUN;
%END;

%ELSE %DO;      /*If there are not zero records, then attach the file and message as originally planned*/
        DATA _NULL_;
        FILE myemail
        TO= ("jsmith@mayo.edu" "manderson@mayo.edu")
        CC= ("bwestra@mayo.edu ")
        FROM="Brent Westra < bwestra@mayo.edu >"
        SENDER="Brent Westra < bwestra@mayo.edu >"
        SUBJECT="Excessive ER Copays: &Yesterday"
        ATTACH=(&Filename);

PUT "Attached are the ER Claims with excessive copays as of &Yesterday";
PUT " ";
PUT "Please let me know if you have any questions.  Thanks,";
PUT " ";
PUT "Brent";
RUN;
%END;

%MEND sendmail;             /*End macro program*/
%sendmail;                  /*Call the macro program*/

FILENAME myemail CLEAR;
```

In this example, there were eight total records in the final data set, so the attachment was sent with the email (Figure 8). Had there been zero records, there would have been no attachment and the message would have said "There are no claims with Excessive ER copays as of 18AUG2011."

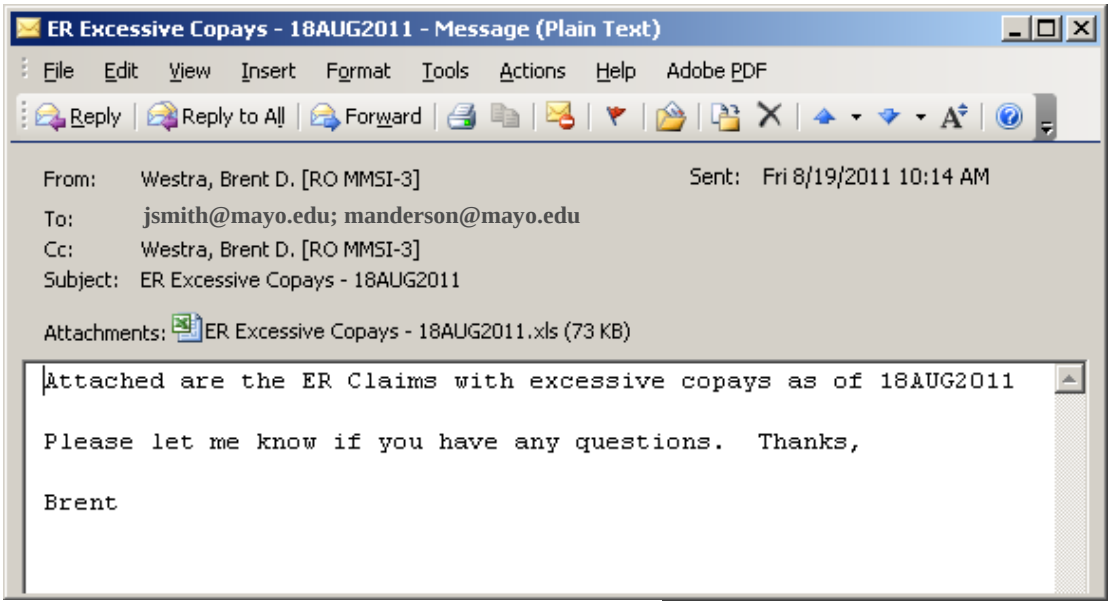
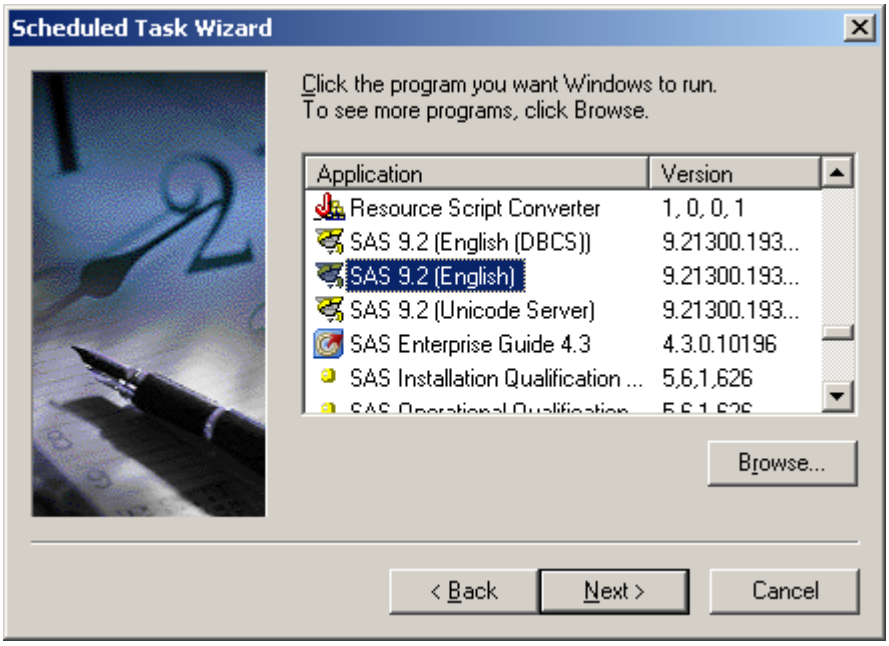


Figure 8: Email sent with attachment

SCHEDULING THE PROGRAM

The last step in the automation process is the only step done outside of SAS. We will use the Microsoft Windows Scheduler to schedule the SAS program to run each week day. To open Windows Scheduler in Windows XP, go to Start⟶Programs⟶Accessories⟶System Tools⟶Scheduled Tasks. Select "Add Scheduled Task" and a wizard will guide you through the setup.

Select SAS from the list of programs.



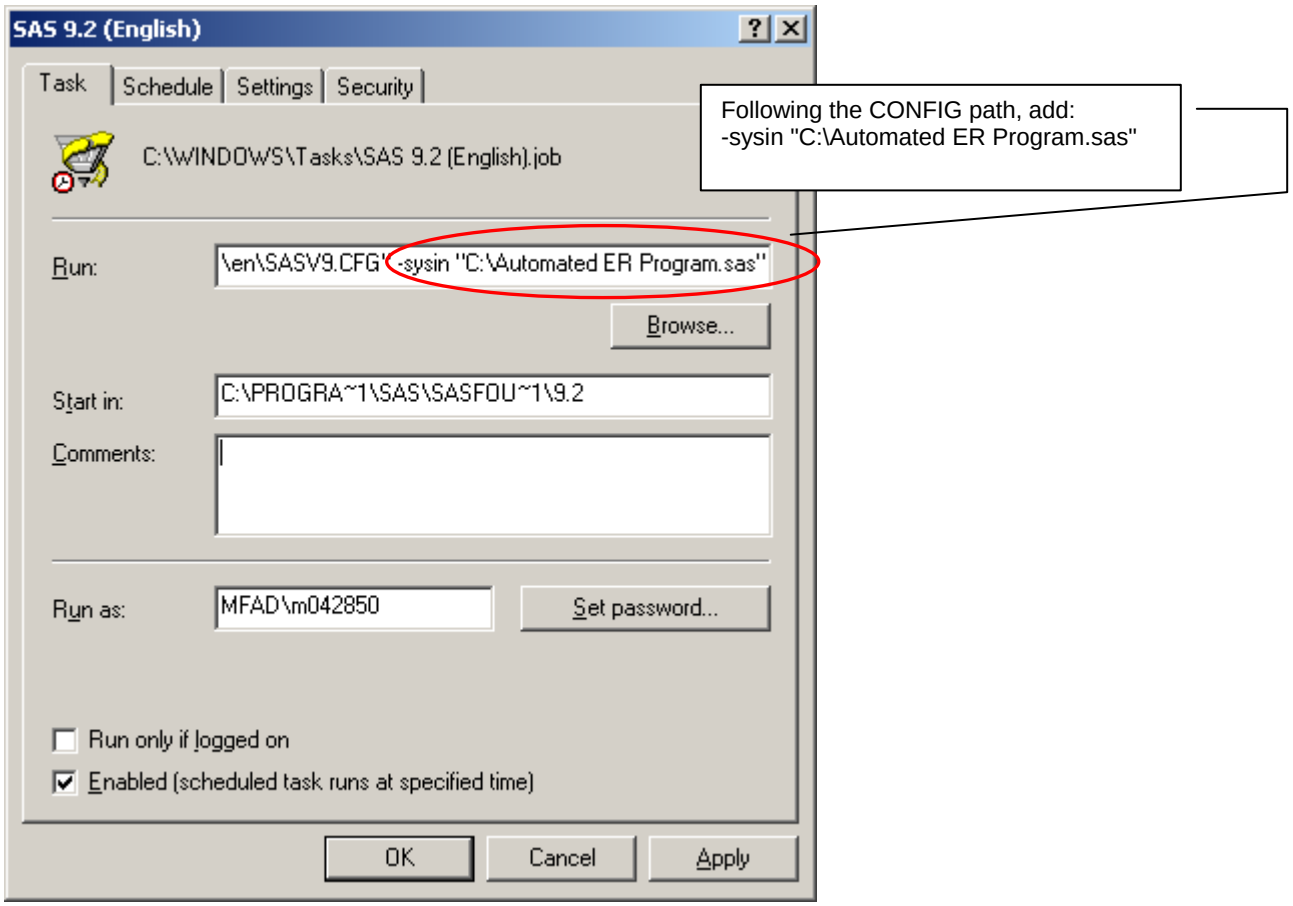
Choose the time and day of week you want the program to run. For this request, we are interested in running the report weekdays at 4:00 AM.

Enter your Windows username and password. Note: If you change your Windows password, you will need to update your password within the Windows Scheduler in order for your scheduled jobs to continue working.

If you follow the wizard, the "Run" window below will be automatically populated with the following:

C:\PROGRA~1\SAS\SASFOU~1\9.2\SAS.exe -CONFIG "C:\Program Files\SAS\SASFoundation\9.2\nls\en\SASV9.CFG"

While this effectively opens SAS, you need to also add the path and name of your automated program to get it run. The word "-sysin" must precede the path of your program. Again, if your program is located on a mapped network drive, you must use its universal naming convention in order for it to work, since your drives will not be mapped if you are logged off.



Leave "Run only if logged on" unchecked, as it is our wish to submit this program early in the morning when we are not logged on to the system. Finally, select "Apply" for all settings to take effect. At this point, your computer is scheduled to open SAS and run your program at 4:00 AM the next morning.

CONCLUSION

Now more than ever, the business climate stresses the need to work more efficiently. With the SAS report automation process, a recurring report that would have required hundreds of employee hours to produce over its lifespan, now takes no time at all. The magnitude of time savings is multiplied by the ease of applying this automation template to other existing and future recurring reports. This increased efficiency comes at almost no cost in relation to the time saved. The data can still be extracted, manipulated, and formatted exactly as you wish. Quality checks can be put in place to ensure the reliability of the report. Finally, the report can be scheduled to be sent out at any time via a fully customizable email. This process injects flexibility into your work schedule and creates additional time for you to pursue other projects and opportunities.

REFERENCES

Andrews, Rick. "Printable Spreadsheets Made Easy: Utilizing the SAS ExcelXP Tagset," NESUG 2008: Applications Big and Small, *http://www.nesug.org/proceedings/nesug08/ap/ap06.pdf*

SAS Institute, " Usage Note *19767:* Using the SAS® System to send SMTP e-mail". *http://support.SAS.com/kb/19/767.html, updated 2007*

SAS Institute, Base SAS(R) 9.2 Procedures Guide: PROC PWENCODE Statement. *http://support.SAS.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002595992.htm*

Schacherer, Chris, Ph.D.: Clinical Data Management Systems, LLC. "Introduction to SAS® for Health Care Data Analysts" ~ Six-week Training Session, 2011. *CSchacherer@CDMS-LLC.com*

Worgen, Jeanina and Jones, Philip. "You've Got Mail - Emailing Messages and Output Using SAS EMAIL Engine," SUGI 29 Paper 178-29, *http://www2.SAS.com/proceedings/sugi29/178-29.pdf*

ACKNOWLEDGEMENTS

I would like to thank Chris Schacherer who has literally taught me everything I know about SAS. Also, thank you to Kristi Jensen, Amy Johnson, and Arlene Guindon at Mayo Clinic, for helping to make this paper possible.

CONTACT INFORMATION

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

Brent D. Westra
Mayo Clinic
200 First Street S.W. Rochester, MN, 55905
Phone: 651-246-5303
E-mail: westra.brent@mayo.edu