From SAS Access to E-Mail: Building an Automated Report Application

Karl Jensen, Magellan Health Services, Sandy, UT
Matt Greathouse, Magellan Health Services, Sandy, UT

ABSTRACT

Since its inception, SAS® has been a powerful tool for creating statistical reports. For the most part, however, these reports have lacked visual appeal. With the use of BASE SAS, Dynamic Data Exchange (DDE) and MS Excel, the look of reports can be substantially improved. Also, by using SAS ACCESS to ODBC, reports can be automated to pull data directly from their source using PROC SQL.

This paper will show the beginning and intermediate SAS user how to develop dialog boxes that prompt for report parameters using the WINDOW and DISPLAY statements, import the data through ODBC with the PROC SQL procedure, export the desired output to Excel templates using DDE, and then directly E-mail the report to desired recipients.

BUILDING WINDOWS

Creating and Displaying a Basic Window

The general syntax for building a basic window and displaying it is as follows:

```
Data _null_;
  Window <name>
         color=<xcolor> irow=<irow> icolumn=<icolumn>
         rows=<rows> columns=<columns>
         #<r> @<c> 'Your text goes here' color=<ycolor>;
  Display <name>;
  Stop;
```

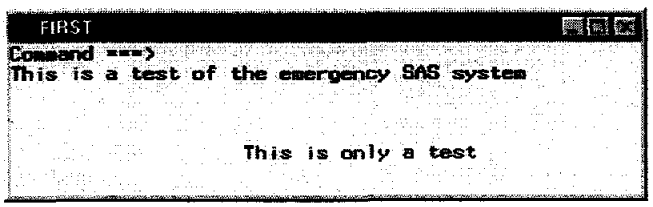
where
<name> is the Window name
<xcolor> is the color choice for the window background
<irow> is the initial row text is displayed in the window
<icolumn> is the initial column text is displayed in the window
<rows> is the number of rows in the window
<columns> is the number of columns wide in the window
<r> is the starting Row reference for the line of text
<c> is the starting Column reference for the line of text
<ycolor> is the color in the displayed line of text

The default for the IROW and ICOLUMN options is 1. The default for the ROWS and COLUMNS options is the last row and column of the display screen respectively.

The display screen is ended when the user hits the enter key or issues the END command - as long as the STOP command is included at the end of the DATA step. If the STOP command is not included the window will continue to be displayed.

The following example produces the display window below.

```
Data _null_;
  Window first color=yellow irow=2 icolumn=2
    rows=10 columns=50
  #1 @1 'This is a test of the emergency SAS system'
    color=blue
  #5 @20 'This is only a test' color=green;
  Display first;
  Stop;
```

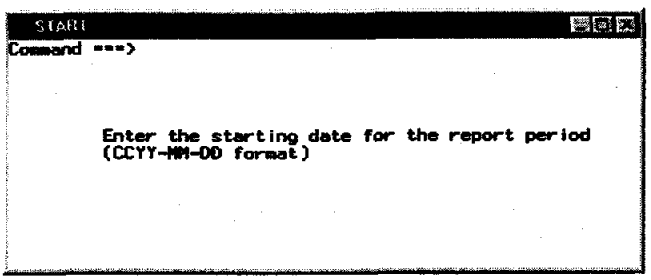


Prompting for Input Parameters

In addition to specifying where the text is displayed using the #<*r*> and the @<*c*>, as previously described, +<*n*> can also be used to move the cursor from its current position <*n*> columns.

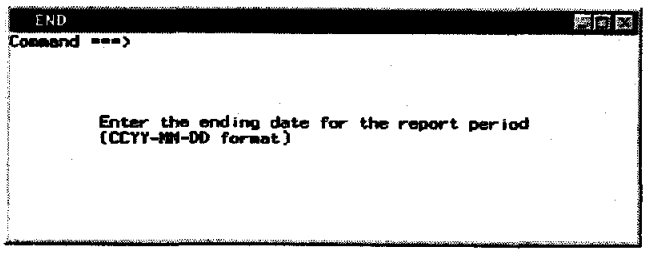
For the next example, a window is created to accept input from the user for the report start date. After SAS writes out the text, the cursor is moved two columns and waits for input for the variable SDATE. The DO WHILE loop, continues the display until a non-blank character is entered for the value of the SDATE variable. This feature can be extended for any number of non-desired inputs.

```
Data _null_;
  Length sdate $ 10;
  Window start
    #5 @10 'Enter the starting date for the report period'
    #6 @10 '(CCYY-MM-DD format)'
    +2 sdate;
  do while (sdate='');
    display start;
    end;
stop;
```



In the next example two windows are created accepting two inputs from the user for the report start and end dates. First, the START display window above appears, then the END display window below appears. The BLANK option in the DISPLAY statement clears the window before displaying the next window or continuing with the program.

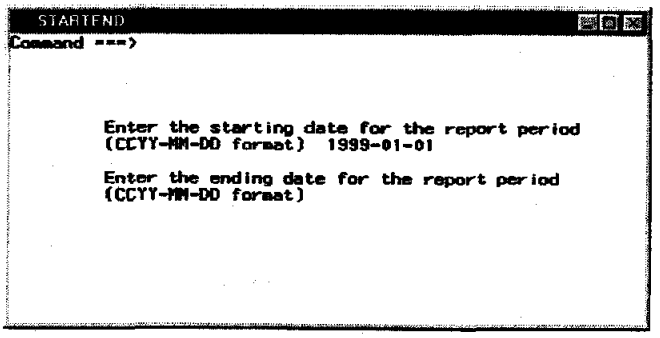
```
Data _null_;
  Length sdate edate $ 10;
  Window start
    #5 @10 'Enter the starting date for the report period'
    #6 @10 '(CCYY-MM-DD format)'
    +2 sdate;
  Window end
    #8 @10 'Enter the ending date for the report period'
    #9 @10 '(CCYY-MM-DD format)'
    +2 edate;
  do while (sdate='');
    display start blank;
    end;
  do while (edate='');
    display end blank;
    end;
stop;
```



The final example is similar to the previous, except only one window is created for the two input parameters. Additionally, the values of the start and end dates are passed through the CALL SYMPUT function to create macro variables. These variables will be used later in the program when extracting the data through the SQL procedure using SAS ACCESS. Because the parameters being passed are the selection criteria for date fields, they need to be surrounded by single quotes. The same would be true when passing parameters that select on text fields.

```
Data _null_;
  Length sdate edate $ 10;
  Window startend
    #5 @10 'Enter the starting date for the report period'
    #6 @10 '(CCYY-MM-DD format)'
    +2 sdate
    #8 @10 'Enter the ending date for the report period'
    #9 @10 '(CCYY-MM-DD format)'
    +2 edate;
  do while (sdate='' or edate='');
    display startend blank;
    end;
  entemp="'" || edate ||"'";
  sttemp="'" || sdate ||"'";

  call symput('stdate',sttemp);
  call symput('endate',entemp);
stop;
```



SAS ACCESS AND THE SQL PROCEDURE

Now that the parameters have been entered for the selection criteria, the next step is to pull the data from the source system(s). Because there are more platforms than can possibly be covered in one paper, this section will give examples for retrieving data from an AS/400 and a MS Access database through open database connectivity (ODBC).

Before data can be extracted from any database the appropriate ODBC drivers must be installed on the PC and an ODBC data source name (DSN) must be created. The data source name can be created by going to the ODBC Data Source Administrator in the Control Panel on your desktop. Dependent on the external database management system (DBMS) that is being connected to, various dialog boxes will prompt for required information such as the path, library, or server name of the specific data source.

The basic syntax for connecting to an ODBC data source in the SQL procedure is:

```
Proc sql;
  Connect to ODBC (dsn = "<dsn>");
  Disconnect from ODBC;
Quit;
```

where *<dsn>* is the user defined data source name

An alias can also be defined for the ODBC connection in the following manner:

```
Proc sql;
  Connect to ODBC as <alias> (dsn = "<dsn>");
  Disconnect from <alias>;
Quit;
```

In the example below, the 'MS Access 97 Database' data source name has been created which points to a reporting database that resides on the local area network (LAN).

```
Proc sql;
  Connect to ODBC as matt (dsn="MS Access 97 Database");

  Create table temp as
  Select * from
    Connection to matt
    (Select * from ReportLog);

  Disconnect from matt;
 Quit;
```

The next example, the 'AS400 Database' data source name has been created which points to a claims table in the CAPS library that resides on the AS400. In addition to specifying the DSN, the AS400 requires a user ID and password. If these are omitted, the DBMS will prompt for these two parameters.

```
Proc sql;
  Connect to ODBC as matt (dsn="AS400 Database"
                           uid=USERID
                           pwd=SASRULZ);

  Create table temp as

  Select * from
    Connection to matt
    (Select * from CAPS.claims);

  Disconnect from matt;
Quit;
```

The previously created macro variables from the prompt window can now be used as selection criteria.

```
Proc sql;
  Connect to ODBC as matt (dsn="AS400 Database");

  Create table temp as

  Select * from
    Connection to matt
    (Select * from CAPS.claims
      where date between &stdate and &endate);

  Disconnect from matt;
Quit;
```

INTRODUCTION TO DDE

DDE uses a relationship between a client (in this case SAS) and a server (Excel). When using DDE from SAS, Excel must be running with each workbook open that will be used. The SAS system establishes a link between itself and Excel through use of the FILENAME statement and DDE Triplet. SAS then uses the PUT command within the DATA step to pass commands to Excel. The syntax is as follows:

```
FILENAME <fileref> dde
    'excel|[workbookname]sheetname!r<w>c<x>:r<y>c<z>';
        where   <fileref> is SAS file reference
                <w> is the starting row in Excel
                <x> is the starting column in Excel
                <y> is the ending row in Excel
                <z> is the ending column in Excel
```

When either importing or exporting data, the defined range (starting and ending rows and columns) must be sufficient to cover the amount of data imported/exported. If the range defined is too small, some of the data will be lost. In the case of importing data it is better to err on the side of defining too much rather than too little. SAS will read the entire range defined in the DDE Triplet and eliminate all records after the last viable record. The case of exporting data is easily overcome through the use of macros, which will be described later in this paper.

RUNNING EXCEL FUNCTIONS FROM SAS

Opening Excel from SAS

As stated previously, in order to use DDE from SAS, Excel must be running. This means that before running your SAS program you must either open Excel and the desired spreadsheet(s) or have SAS automatically open them for you. To open Excel from SAS the following commands need to be executed:

```
Options noxwait noxsync;
x 'c:\excel\excel';
```

The X command specifies that operating system commands are to follow. The NOXWAIT option allows you to return to SAS without having to type "EXIT" at the command prompt session. The NOXSYNC option allows you to return to SAS before you finish with your other application (in this case Excel).

It is a good idea, although not always necessary, to add the SLEEP(5) function to allow Excel sufficient time to open before SAS begins executing Excel commands.

```
Data _null_;
  x=sleep(5);
```

Running Excel Macro Commands

Another advantageous feature of DDE is to run Excel macro commands from the SAS program. This allows you to open a workbook, move between sheets within a workbook, save a workbook and other Excel macro commands. In order to do this you must first indicate to Excel that you are sending macro commands. This is again accomplished through the FILENAME statement.

```
Filename <fileref> dde 'excel|system';
```

Using the FILE *<fileref>* in a data step, in conjunction with the PUT statement, executes the Excel macro commands. For example, the following opens a specific spreadsheet called Claimdat.xls.

```
Filename cmds dde 'excel|system';
Data _null_;
  File cmds;
  Put '[Open("c:\Claimdat.xls")]';
```

Essentially, the PUT statement passes the quoted string, which contains the Excel 4.0 macro commands. The command is always contained within the square brackets ([ ]). With this any number of commands can be sent to Excel. The syntax below opens the "Claimdat.xls" spreadsheet, selects cell B4 in the sheet named "Provider Summary," inserts a formula which adds the contents in cells B1 through B3, saves the spreadsheet as "Provsum.xls", closes the spreadsheet and closes Microsoft Excel.

```
Data _null_;
  File cmds;
  Put '[Open("c:\Claimdat.xls")]';
  Put '[Workbook.Select("Provider Summary",
       "Provider Summary")]';
  Put '[select("R4C2")]';
  Put '[Formula("=Sum(R1C:R3C)")]';
  Put '[Save.As("c:\Provsum.xls")]';
  Put '[Close()]';
  Put '[Quit()]';
```

In addition to running Excel 4.0 macro functions from SAS, you can also run a macro created directly in Excel. The macro in Excel can be written in either 4.0 or visual basic code and is executed in SAS DATA step as follows:

```
Put '[Run("<macroname>",false)]';
```

Remember that when running Excel 4.0 macro commands from SAS, the PUT statement is essentially passing a character string (the characters between the single quotes) to Excel to be executed. Adding the value of a SAS macro variable into this character string would enable you to change the name of the Excel workbook, change the defined range within the DDE Triplet or a number of other things.

Since the resolution of SAS macro variables requires double quotes, the single quotes are replaced with double quotes indicating to SAS that the string needs to be resolved prior to passing to Excel. Next, because some Excel 4.0 macros already contain double quotes, we need to tell SAS to mask these characters and pass them on, unresolved, to Excel. This is done using the SAS macro function %bquote. For example, lets say you have a SAS global macro variable *<filnm>* defined as "Provsum". You can save the previously opened Excel file as Provsum.xls using the following syntax:

```
Data _null_;
  File cmds;
  Put "[Save.As(%bquote("c:\&filnm..xls"))]";
```

Use of user defined SAS macro variables along with the %bquote function is a powerful tool in automating your report writing as will be seen later in this paper.

IMPORTING DATA

When SAS reads data from Excel, it reads each row as a string with the tab character between each column. These tab characters are converted when passed to SAS as blank spaces, which is the default SAS delimiter. This works nicely except when a cell value contains blank spaces, which are also interpreted as delimiters. Unfortunately, many times we are reading in character values that contain blank characters.

In order to overcome this problem, we first indicate to SAS to not convert the tabs into blank spaces. This is accomplished by adding the NOTAB option in the FILENAME or INFILE statement. Since this conversion does not take place, the tab character can be used as the delimiter. This is done by adding to the FILENAME or INFILE statement DLM='09'x. Through these procedures each column in Excel is read in as a different variable in SAS.

Some difficulty arises when cells in the Excel spreadsheet contain missing data points. In order to read these values properly and not skip over these cells, the DSD and MISSOVER options are used in connection with the INFORMAT statement. The DSD option added to the INFILE statement indicates to SAS that if two consecutive delimiters occur, this is interpreted as a missing data point. The MISSOVER option is used to indicate that SAS should not go to a new input line if it does not find the desired variable in the current line. The INFORMAT statement forces SAS to read input as a modified list, that is, it tells SAS exactly the format to read for each variable.

An example for importing data is as follows:

```
Filename frmexcel dde 'excel|[wrkbk.xls]clmdata!r2c1:r200c4';
Data Claims;
  Infile frmexcel dlm='09'x dsd notab missover;
  Informat patname $25.;
  Input Patname Claimno Amtpd;
```

EXPORTING DATA

Simple Example

When exporting data to Excel, blanks are converted into tab characters and these tabs are used as the column delimiters. Thus, if a variable in SAS contains the value "Magellan Health Services", "Magellan" would end up in the first cell, "Health" in the second and "Services" in the third. For each blank space within a variable, the tab character is created. The NOTAB option is again needed to overcome this problem.

With the NOTAB option in place you need to physically tell SAS when you want to move to a new cell. Use the '09'x option in connection with the PUT statement in the DATA step to accomplish this task. The example below exports the data set, Hmodx, containing fifty observations to the hmo.xls workbook.

Sample Data:

Hmo	*Dx*	*Days*	*Sess*
CA	CD	3	15
CA	CD	6	35
CA	CD	4	11
CA	MH	3	14
CA	MH	2	7
.	.	.	.
.	.	.	.
.	.	.	.

```
Filename Hmoout dde 'excel|[hmo.xls]sheet1!r1c1:r50c4' notab;
Data _null_;
  Set Hmodx;
  File Hmoout;
  Put Hmo '09'x Dx '09'x Days '09'x Sess;
```

When using an Excel workbook that contains only one sheet or when the sheet you are exporting to is the first sheet, you can exclude the sheet reference in the DDE Triplet. The square brackets ([ ]) may also be excluded.

Filename Hmoout dde 'excel|hmo.xls!r1c1:r50c4' notab;

Extra Line Before/After Subtotals

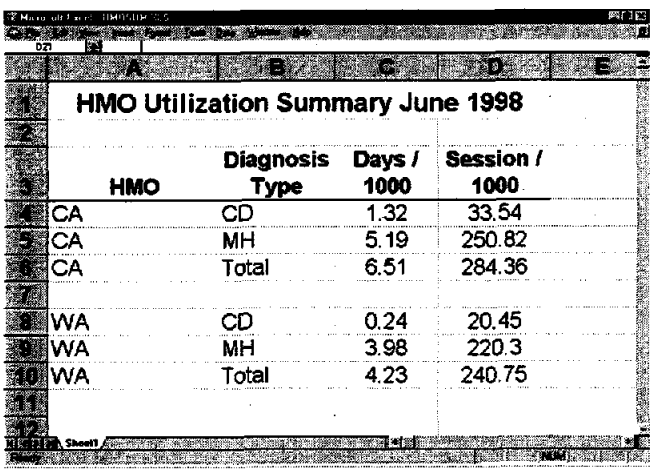
Many times you will want to separate sections of the report by a blank line. This is easily completed by using the last.<variable> or first.*<variable>* functions. In the following example, a blank line is inserted after the HMO summary line.

Sample Summary Data:

Hmo	*Dx*	*_TYPE_*	*Daysper*	*Sessper*
CA	Total	1	6.51	284.36
WA	Total	1	4.23	240.75
CA	CD	3	1.32	33.54
CA	MH	3	5.19	250.82
WA	CD	3	0.24	20.45
WA	MH	3	3.98	220.30

```
Filename outp dde 'excel|Hmosum.xls!r4c1:r10c4' notab;
Proc sort; by hmo descending _TYPE_;
Data _null_;
  File outp;
  Set hmosum;
  By hmo;
  Put Hmo '09'x Dx '09'x Daysper '09'x Sessper;
  If last.hmo then Put '09'x;
```

The sample output would look something like:



HMO Utilization Summary June 1998

HMO	Diagnosis Type	Days / 1000	Session / 1000
CA	CD	1.32	33.54
CA	MH	5.19	250.82
CA	Total	6.51	284.36
WA	CD	0.24	20.45
WA	MH	3.98	220.3
WA	Total	4.23	240.75

Determining the Length of Exported Data

As stated earlier, when exporting data it is vital to define the range within the DDE triplet sufficient to cover the number of records in your data set. Unfortunately, the number of observations in a data set is typically not known until after the data set has been created. However, the process of determining the length of a data set can be automated through the use of simple DATA step options and a CALL SYMPUT statement.

The first step is to use the END=*<lastflg>* option in the SET statement. This options sets the variable *<lastflg>* equal to zero for all observations except the last, which is set equal to one. When the *<lastflg>* = 1, a macro variable can be created and set equal to the number of rows needed to export using the _n_ variable. If a header row is desired in Excel, the macro variable is set to _n_+1 in order to compensate for the additional row.

Because macro variables are, by default, BEST12., it is necessary to eliminate any blanks from the SAS macro variable through the use of the LEFT and TRIM functions.

In order for the DDE Triplet to evaluate the value of the SAS macro variable, double quotes replace the single quotes surrounding the DDE Triplet.

```
Data _null_;
  Set Clmdata END=lastn;
  If lastn then call symput('row',trim(left(_n_+1)));
Run;
Filename Toexcel dde
     "excel|[Claims.xls]claims data!r2c1:r&row.c3" notab;
```

It is important to note that the CALL SYMPUT statement is not executed until it reaches a step boundary and that the FILENAME statement is not considered a step boundary. Step boundaries include the RUN, DATA and PROC statements. Thus, in order for the DDE Triplet to resolve the macro variable, the RUN statement, or some other step boundary, needs to be included before the FILENAME statement.

Exporting to Multiple Sheets

Exporting data to multiple sheets within an Excel template requires simply changing the sheet name in the DDE Triplet. For example, perhaps you would like summary information in one sheet and detail in a different sheet. The FILENAME statements might look as follows:

```
Filename detail dde
     "excel|[Provider.xls]Prov Detail!r2c1:r&row1.c4" notab;
Filename provinfo dde
     "excel|[Provider.xls]Prov Summary!r2c1:r&row2.c2" notab;
```

Exporting to a Template

Versions prior to Excel 97

Sometimes you may have an Excel template set up in one sheet and you want to use this sheet multiple times for several different group variables. In the example below, a copy of the "Template" is created for each HMO plan existing within the data set.

```
Data _null_;
  Set hmolist END=lastn;
  tmp='[Workbook.Name("Template (2)"," ' ||trim(hmo)|| ' ")]';
  Call symput('h' || trim(left(_N_)),hmo);
  File cmds;
    Put '[Workbook.Select("Template","Template")]';
    Put '[Workbook.Copy("Template", "hmo.xls",2)]';
    Put '[Workbook.Select("Template (2)","Template (2)")]';
    Put tmp;
  If lastn then do;
    Put '[Workbook.Select("Template","Template")]';
    Put '[Workbook.Hide()]';
    Call symput('totn',trim(left(_N_)));
   End;
```

When creating multiple sheets from the template, the data set must only contain one record for each group. Otherwise Excel will attempt to name multiple sheets with the same name and an error will occur.

The variable *<tmp>* above is created so that the value of the variable *<hmo>* can be used as the name of the sheet. An alternative method is to use the %bquote function as follows:

```
Put "[Workbook.Name(%bquote("Template (2)",")"
  hmo +(-1) "%bquote("))]";
```

While the first method does require the creation of an additional variable, it is less complex than the %bquote method.

Excel 97 Version

In Excel 97 the function WORKBOOK.NAME does not work when passed through DDE from SAS. In order to get around this problem, a 4.0 macro sheet must be created in the template workbook and a macro created containing the WORKBOOK.NAME function. The macro may look something like

```
<Macro Name>
=WORKBOOK.NAME("Template (2)","Newsheet")
=RETURN()
```

with each line in a separate cell. The cell containing the WORKBOOK.NAME function is replaced from SAS for each new group sheet.

The macro is defined by choosing the INSERT-NAME-DEFINE option in Excel. In the example below the macro is named "Rename". The macro is called from SAS as previously described in this paper.

```
Data _null_;
  Set hmolist END=lastn;
  tmp='[Formula("=Workbook.Name(""Template (2)"", "" '
||trim(hmo)|| ' "")")]';
  Call symput('h' || trim(left(_N_)),hmo);
  File cmds;
    Put '[Workbook.Select("Template","Template")]';
    Put '[Workbook.Copy("Template", "hmo.xls",2)]';
    Put '[Workbook.Select("Macro1","Macro1")]';
    Put '[Select("R2C1")]';
    Put tmp;
    Put '[Run("Rename",False)]';
  If lastn then do;
    Put '[Workbook.Select("Template","Template")]';
    Put '[Workbook.Hide()]';
    Call symput('totn',trim(left(_N_)));
   End;
```

After the creation of the group sheets, information specific to each group is exported using the macro variables created with the CALL SYMPUT statement as follows:

```
%Macro Outhmo;
 %Do i=1 %to &totn;
  Filename outp dde "excel|[hmo.xls]&&h&i!r8c3:r17c4" notab;
  Data _null_;
    File outp;
    Set Hmosum;
    Where hmo="&&h&i";
    Put hmo '09'x dx;
  %End;
%Mend;
%Outhmo
```

Using Multiple DDE Triplets Within the DATA Step

Each time a record is read into the DATA step, the FILE *<fileref>* statement writes the data on the next row from where it left off from the previous record. This continues until a step boundary occurs. Thus if multiple FILE *<fileref>* statements are used within the same DATA step, SAS remembers where it left off in the DDE Triplet range for each FILE *<fileref>* statement. This can be an advantageous tool in manipulating the data as it is exported to Excel.

Sample Data:

Patnm	*LvlCare*	*Dx*	*Servdate*
Kelly Kocktosen	OP	295.40	12/1/97
Kelly Kocktosen	OP	303.90	12/15/97
Kelly Kocktosen	IP	291.5	1/18/98
Borlan Wheaton	OP	780.52	10/1/97
Borlan Wheaton	OP	305.90	10/8/97
Jason Joshua	IP	290.11	11/4/97
Jason Joshua	IP	290.12	12/18/97
Jason Joshua	OP	290.10	1/8/98

The desired result is to have one line for each patient and level of care, showing each date of service and the related DSM code.

Patnm	*LvlCare*	*Dx1*	*Servdt1*	*Dx2*	*Sevdt2*
Kelly	OP	295.40	12/1/97	303.90	12/1/97
Kelly	IP	291.5	1/18/98		
Borlan	OP	780.52	10/1/97	305.90	10/8/97
Jason	IP	290.11	11/4/97	290.12	12/18/98
Jason	OP	290.10	1/8/98		

First, the data is sorted by patient name and level of care. Then, each time the record contains a new level of care (First.Lvlcare) within a patient name, the record is exported to the first section (*<serv1>*). If the record is also the last level of care for the specific patient, blanks are exported to the second section (*<serv2>*). If it is not the last level of care, the second DSM code and service date is exported to the second section (*<serv2>*). This same methodology can be expanded to include the largest possible number of treatments a patient my have. The example below assumes the total possible treatments is four.

```
Proc Sort Data=Patient;
  By Patnm Lvlcare Servdate;

Filename serv1 dde 'Excel|Patdetl.xls!r2c1:r10c4' notab;
Filename serv2 dde 'Excel|Patdetl.xls!r2c5:r10c6' notab;
Filename serv3 dde 'Excel|Patdetl.xls!r2c7:r10c8' notab;
Filename serv4 dde 'Excel|Patdetl.xls!r2c9:r10c10' notab;
```

```
Data _null_;
  Set Patient;
    By Patnm Lvlcare;
  If First.Lvlcare then do;
    File serv1;
    Put Patnm '09'x Lvlcare '09'x Dx '09'x Servdate;
    n=1;
   End;
  Else do;
    n+1;
    If n=2 then File serv2;
    Else if n=3 then File serv3;
    Else if n=4 then File serv4;
    Put Dx '09'x Servdate;
   End;
  If Last.Lvlcare then do;
    If n=1 then do;
      File serv2;
      Put '09'x;
      File serv3;
      Put '09'x;
      File serv4;
      Put '09'x;
     End;
    If n=2 then do;
      File serv3;
      Put '09'x;
      File serv4;
      Put '09'x;
     End;
    If n=3 then do;
      File serv4;
      Put '09'x;
     End;
   End;
```

DISTRIBUTING YOUR REPORT USING E-MAIL

When sending out the report via email, you need to make sure you have a VIM or MAPI compliant program and that it is open. To send out the completed report is as simple as creating a FILENAME statement and a null data step.

```
Filename <fileref> email subject=<text> to=<xrecipient>
cc=<yrecipient> attach=<filename>;
Data _null_;
  File <fileref>;
```

where
<*fileref*> is a valid SAS file reference
<*text*> is the text displayed in the subject line
<*xrecipient*> is the name of the person(s) to receive the email
<*yrecipient*> is the name of the person(s) copied on the email
<*filename*> is the file name of the report to be attached

A couple of things to remember when sending e-mail from SAS: (1) The recipient name must match the name in your email list, or be a valid internet e-mail address, and (2) you must include the path for the file that is being attached.

```
Filename mymail email subject='EAP Satisfaction Report'
        to='Karljens@Hotmail.com'
        cc='Greathouse, Matt L.'
        attach='c:\hmodata.xls';
Data _null_;
  File mymail;
```

In order to include multiple recipients, and/or attachments, all recipients (attachments) must be included in parenthesis. Additionally, since macro variables can be passed to the filename statement, a report created and saved previously in the SAS program can be attached easily.

```
Filename mymail email subject='EAP Satisfaction Report'
        to=('Karljens@Hotmail.com' 'Greathouse, Matt L. ')
        attach=("&filen" 'c:\hmodata.xls');
Data _null_;
  File mymail;
```

CONCLUSION

Dynamic SAS programs as described in this paper allow those who have little knowledge of SAS to run reports, using the WINDOW and DISPLAY statements to input the report parameters. The data is then imported to SAS using ODBC and SAS ACCESS and exported into Excel using the DDE Triplet. With a properly defined range in the DDE Triplet, data can be exported to any cell(s) within an Excel template. The template can be copied multiple times for several groups within the data set allowing consistency in the report format. These templates can also include tables linked to graphs giving the report a nice visual appeal. Using the DDE Triplet, Excel 4.0 macro commands and SAS macro function expands the reporting capabilities of SAS to include all the dynamics that Excel has to offer. Then the Excel report can be directly E-Mailed to all recipients from SAS, creating a completely seamless automated report application.

References

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Product names are registered trademarks or trademarks of their respective companies.

SAS Institute Inc., *SAS Companion for the Microsoft Windows Environment, Version 6, First Edition*, Cary, NC: SAS Institute Inc.
SAS Institute Inc., *SAS Macro Language: Reference, First Edition*, Cary, NC: SAS Institute Inc., 1997. 304 pp.
SAS Institute Inc., *SAS Language: Reference, Version 6, First Edition* Cary, NC: SAS Institute Inc., 1990. 1042 pp.

Contact Information

Matt Greathouse - Greathom@Cougar.Netutah.Net
Karl L. Jensen - Karljens@Hotmail.Com

Magellan Health Services
P.O. Box 57986
Salt Lake City, UT 84157-0986