Paper SS-05

Keeping Up Appearances:
Turning Specifications into SAS® Format Libraries and Statements

Sarah Woodruff, Westat, Rockville, MD

ABSTRACT

Specifications documents concerning the desired appearance of SAS data are often provided in an Excel spreadsheet format. While such an arrangement provides ease of use to the person creating it, typically the client to whom the final data delivery will eventually be returned, having format information set up this way is not directly conducive to its use in SAS. This paper describes a process by which formats presented in this way can be easily converted into both SAS format libraries and format statements. The process centers around the use of standard reports from Oracle Clinical to provide the basic information, but the methods can be applied to any specifications document in Excel.

Along with the final format products, this encompasses evaluating the appropriateness of format names and making broadly applicable changes as needed, ensuring unique variable names, particularly if data is coming from multiple sources, and building in appropriate quality control steps along the way. This conversion requires both DATA step work and several common procedures (IMPORT, CONTENTS, FREQ), but is accomplished entirely in Base SAS and does not need any special products. Flexibility remains in the process to compile or subdivide both the format catalog and statements as needed by the user or as applicable based on the requirements of a particular deliverable.

Keywords: format library, format statement, Oracle Clinical, Excel

INTRODUCTION

SAS formats provide great flexibility in the display of data. Numeric values can appear with character descriptions, minimal data entry can still yield a user-friendly appearance and flexibility in data sorting is increased. Beyond issues of readability, grouping and categorizing can be done with formats without having to alter the original values. All of this customization provides a variety of options when deciding how to best meet the needs of a client or if the client has particular specifications concerning deliverables.

Though it is very common for clients to have a well-defined goal when it comes to the appearance of final deliverables, it can be challenging to determine the most efficient way to make sure the goals for the data should be met. In this discussion, the focus is on the acquisition of format specifications via Excel since this is a common way to enter and transmit such guidelines. Furthermore, this paper utilizes information coming out of Oracle Clinical reports. The process which utilizes this information could be modified to utilize formats in any Excel document so long as all of the required components were included in the originating spreadsheet.

Not only is it necessary to make sure that the final delivery of the data is properly formatted, but a format catalog needs to accompany that data. This process meets that need as well, allowing for the creation of a format catalog which can be used during data creation and manipulation, and can then be included in the data delivery for use by the client during their continued work with the data itself. The only limitation is that the client needs to know what a SAS format catalog is and how to access it. Instructions on how to do so could be easily provided along with reference to `OPTIONS FMTSEARCH = ( format_libname ) ;` being included in all programs utilizing the data.

ACQUIRING FORMAT VALUES

If data has been entered into Oracle Clinical, then the OC developers will have been provided with specifications as to how the data should be formatted as it is entered. Use of OC also provides the ability to take advantage of the system's standard reports. The first of those to be used by this process is the DVG (discreet value group) and AlphaDVG report. This report includes information on DCMs (data collection modules) which will likely parallel the SAS views or data sets being utilized by the study or project, the questions as they appear in OC along with their lablels, the DVG being applied to each question, the possible values within each DVG and then the long values that tie to each value as it is entered. The values themselves would be the data that shows up in a SAS data set while the long value represents how that data appears once it is formatted.

The first step in SAS is to bring in the information available in the DVG/AlphaDVG report which is generated by OC as a listing in Excel.

```
PROC IMPORT datafile = '\\file-location\filename.xlsx'
     OUT = DVG_Report DBMS = Excel ;
     SHEET = "sheetname" ;
     MIXED = YES ;
     SCANTEXT = YES ;
RUN ;
```

While OC makes it easy to acquire this information, if a client has created a specifications document, it need only contain columns with similar information: a way to link each SAS view with the relevant variables, the variables themselves, a general designation for the grouping of the values, the values themselves and their long value equivalents. The most important aspect of the specifications document is that there is a line *per possible value* of each variable. If those values have been entered horizontally as opposed to vertically, a **PROC TRANSPOSE** could be used to create the proper arrangement, but if it is possible to give input at the beginning, consider requesting an entirely vertical listing so that this manipulation can be avoided.

If this is the first time working with the DVG/AlphaDVG report for a given project or the first time a client has provided a specifications spreadsheet, it is useful to then examine the contents of either one.

```
PROC CONTENTS data=DVG_Report ;
  title "DVG_AlphaDVG Report" ;
RUN ;
```

By looking at the **PROC CONTENTS** output, you can ensure that you have all of the variables you will need to proceed while also seeing how those variables have been imported in to SAS, thus making it easier to manipulate them later. In addition, you can examine the data types and lengths as this may be relevant to the type of data you will be working with as part of the project or expectations as to how it will be entered.

The next step is to isolate only those variables which will be necessary going forward. This also provides an opportunity to make any changes needed to have values be more consistent or to change variable names if the ones imported into SAS are not particularly meaningful or user-friendly. This next block of code represents this step as well as the alterations that were made based on how the information came in from OC.

The KEEP statement for this data step isolates the general classification of the grouping, the specific numeric designation for the grouping (with these two together indicating a specific “format”), the possible value to be entered within each format and the long values possible for each of those data possibilities.

```
DATA DVG_Variables ( keep = DVG DVG_num DVG_Value DVG_Long_Value Alpha_DVG
                            Alpha_DVG_num Alpha_DVG_Value Alpha_DVG_Long_Value) ;
  RENAME _ = DVG_num ; /* _ did not provide a meaningful variable name for the
                          variable that was actually the numeric designation for the
                          DVG number, the number assigned to that particular format, so
                          the variable name was changed */
  RENAME _1 = Alpha_DVG_num ; /* _1 did not provide a meaningful variable name for the
                          variable that was actually the numeric designation for the
                          AlphaDVG number, the number assigned to that particular
                          format, so the variable name was changed */
  SET DVG_Report ;
  /* Multiple values ended up coming out of OC to indicate a lack of data so those
     needed to be standardized. */
  IF Alpha_DVG in ( ' ' , '(null)' ) then Alpha_DVG = ' ' ;
  IF Alpha_DVG_num in ( ' ' , '(null)' ) then Alpha_DVG_num = ' ' ;
RUN ;
```

Next, the listing needs to be sorted in such a way that the data type can subsequently be attached. In this step, the NODUPKEY option is utilized to ensure the list of possible values is unique since the original report is a listing based on all possible variables in all of the data sets.

```
PROC SORT data = DVG_Variables out = DVG_Variables_Sort
                ( where = ( DVG ne ' ' or Alpha_DVG ne ' ' ) )
                NODUPKEY ;
  by DVG DVG_num DVG_Value DVG_Long_Value Alpha_DVG Alpha_DVG_num
     Alpha_DVG_Value Alpha_DVG_Long_Value ;
RUN ;
```

ACQUIRING DATA TYPES

The DVG/AlphaDVG report in OC does not provide information on the type of data being stored in each SAS variable. Yet knowing whether a variable is character or numeric affects how that data is manipulated, sorted and otherwise handled, including how a format specification needs to be constructed. So this information is a critical addition.

The next step brings in OC's SDD (standard data definition) report in order to provide this information.

```
PROC IMPORT datafile = '\\file-location\filename.xlsx'
     OUT = SDD_Report DBMS = Excel ;
     SHEET = "sheetname" ;
     MIXED = YES ;
     SCANTEXT = YES ;
RUN ;
```

Again, if this is the first time working with SDD report for a given project or if the client is providing specifications information in separate spreasheets, it is useful to then examine the contents of either one.

```
PROC CONTENTS data = SDD_Report ;
  title "SDD Report" ;
RUN ;
```

The entries in the SDD report need to match what has been isolated from the DVG/AlphaDVG report. Meaningful variable names should also be provided as needed. The KEEP statement isolates only those variables to be used going forward.

```
DATA SDD_Variables ( keep = TYPE DVG DVG_num Alpha_DVG Alpha_DVG_num ) ;
  set SDD_Report ;
    rename _ =  DVG_num ;
    rename _1 = Alpha_DVG_num ;
  where DVG not in ( ' ' , '(null)' )  or Alpha_DVG not in ( ' ' , '(null)' ) ;
RUN;
```

Next, the SDD variables need to be sorted to be compatible with the data coming from the DVG/AlphaDVG report.

```
PROC SORT data = SDD_Variables out = SDD_Variables_Sort
                NODUPKEY ; /* Using NODUPKEY ensures matching based on the possible
                              format values. */
  by DVG DVG_num Alpha_DVG Alpha_DVG_num TYPE ;
RUN ;
```

In order to determine data type, only one instance of each “format designation” is required. However, as part of the quality assurance in this process, it is a good idea to check and see if any variable has ended up with both a character and a numeric designation. Clearly, this is not allowed in SAS and catching any such problems in the format information now saves time spent trouble-shooting later.

```
DATA SDD_Variables_Type ;
  set SDD_Variables_Sort ;
    by DVG DVG_num Alpha_DVG Alpha_DVG_num ;
    TYPE_OC = TYPE ;
  if not ( first.Alpha_DVG_num and last.Alpha_DVG_num )
  then TYPE_OC = 'BOTH' ; /* The values in variable TYPE_OC can then be examined to
                             check for conflicts in determination of data type. */
RUN ;
```

The values in the variables TYPE_OC can then be manually examined to check for conflicts in determination of data type. If there is an inconsistency in the data types, then the easiest resolution would be to make necessary changes to the specifications. Typically, such a change would be simpler than making changes to the actual data base.

Once conflicts are vetted, it is possible to confidently combine the data and format values with the data types, thus bringing together the information provided by the DVG/AlphaDVG and SDD reports.

```
DATA DVG_SDD_Listing Error ;
  merge DVG_Variables_Sort ( in = in1 )
        SDD_Variables_Type ( in = in2 drop = TYPE ) ;
    by DVG DVG_num Alpha_DVG Alpha_DVG_num ;
  if not ( in1 and in2 ) then output Error ;
  else output DVG_SDD_Listing ;
RUN ;
```

The inclusion of the Error data set in the above code then makes it easy to follow up on any values that show up in one data set, but not in the other. Building in checks like this as the process progresses also helps to minimize or simplify trouble-shooting as the steps become more complex.

CREATING FORMAT SPECIFICATIONS

With all the values for the formats as well as the data types now in one data set, the actual specifications can be created. This new data set includes variables for the format name, the format itself, the values possible for the format, and the type of data. It puts all of them in one consistent structure which will then provide the basis for the format catalog.

Prior to this step, it is necessary to look at the range of DVG and AlphaDVG values to determine what the first portion of the format name will be. When the data are coming out of OC, this value is based on the general grouping of the format and provides a good “top level” portion of a format name, much like a genus and species naming convention for living things. The “species” portion of the format name will come from DVG_num, a variable which comes straight from OC, but was renamed in an earlier step. This gives specificity to a format within a particular group. A similar process can be used with client-provided specifications if they have split out the format information in a similar way. Alternatively, if the client has already made the format nomenclature more specific, this step can be simplified.

```
DATA Format_Specifications ;
  set DVG_SDD_Listing ;
  length TYPE $ 1 ; /* variable holding data type */
  length FMTNAME $ 8 ; /* variable holding format name */
  length START $ 7 ; /* variable holding data values */
  length LABEL $ 200 ; /* variable holding format values */

  if DVG not in  ( ' ' , '(null)' ) then
```

```
    do ;
      select ;
         /* set DVG values based on those generated in OC or by client */
        when (DVG = "value")   FMTNAME = "FM1_" ;
        /* repeat above line for as many DVG values as needed */
      end ;
      FMTNAME = CATS ( FMTNAME , PUT ( DVG_num,BEST. ) ) ;
         /* concatenation of these two variables builds the genus-species level format
            name */
      START = left ( trim ( DVG_Value ) ) ;
      LABEL = left ( trim( DVG_Long_Value ) ) ;
    end ;
  if Alpha_DVG ne ' ' then
    do ;
      if Alpha_DVG = "value" then FMTNAME = "FMA_" ;
      FMTNAME = CATS ( FMTNAME , Alpha_DVG_num ) ;
      START = left ( trim ( Alpha_DVG_Value ) ) ;
      LABEL = left ( trim ( Alpha_DVG_Long_Value ) ) ;
    end ;
  /* following select statement sets the data type for each */
  select ;
    when ( TYPE_OC = "CHAR" )
      do ;
        TYPE = "C" ;
        FMTNAME = CATS ( FMTNAME , TYPE ) ;
        output Format_Specifications ;
      end ;
    when ( TYPE_OC = "NUMBER" )
      do ;
        TYPE = "N" ;
        FMTNAME = CATS ( FMTNAME , TYPE ) ;
        output Format_Specifications ;
      end;
       /* provides error checking and a simpler way to trouble-shoot the format output
          based on data type */
    when ( TYPE_OC = "BOTH" )
      do ;
        FMTNAME_HOLD = FMTNAME ;
        TYPE = "C" ;
        FMTNAME = CATS ( FMTNAME_HOLD , TYPE ) ;
        output Format_Specifications ;
        TYPE = "N" ;
           FMTNAME = CATS ( FMTNAME_HOLD , TYPE );
        output Format_Specifications ;
      end;
    otherwise put "Error: " TYPE_OC= ;
  end ;
  drop FMTNAME_HOLD ;

RUN ;
```

Once the data set with all the format details is complete, some additional examination of the final assignments is a prudent step to take, even if a client-created version of the specifications seems simpler than what OC produced and was fed into this process. This first of two steps would be to ensure that the data type has been correctly assigned.

```
PROC FREQ data = Format_Specifications ;
  table TYPE_OC * TYPE / list missing ;
  title "Verify Assignment of TYPE Variable" ;
RUN ;
```

A second examination worth making is ensuring the name of each format was properly constructed. This can be particularly critical if the client has been very specific about how these were to be built. But even when working with output from OC, it is worth making sure that all such names are consistent with expectations for them. Double-checking this now makes it easier to work with the actual formats later.

```
PROC FREQ data = Format_Specifications ;
  table DVG * DVG_num * Alpha_DVG * Alpha_DVG_num * TYPE_OC * TYPE * FMTNAME
       / list missing ;
  title "Verify Assignment of FMTNAME" ;
RUN ;
```

A final sorting should be done before saving the SAS data set containing all of the format information. This can be incorporated into a prior step, but having it as its own distinct step can make it easier to follow the flow of data being manipulated.

```
PROC SORT data = Format_Specifications out = Format_Specifications_Sort;
  by FMTNAME TYPE START LABEL ;
RUN ;
```

Finally, the SAS data set with the format information needs to be saved in a permanent location for further use, and only the relevant variables need to be kept.

```
DATA permanent_location.Format_Specifications ;
  set Format_Specifications_Sort ;
  keep FMTNAME TYPE START LABEL;
RUN ;
```

Though not strictly necessary, it is useful to have this final data set stored as an Excel, XML, or HTML file; some file format that is easy to manipulate, search, and sort. Incorporate a final step to produce such output after saving the data to its permanent location.

CREATING FORMAT LIBRARY

With an established data set, a portable format catalog can now be created. A location for the catalog needs to be determined. Once the location is determined, the format catalog can be created with a single proc.

```
PROC FORMAT library = libname_location
            fmtlib
            cntlin = libname_location.format_specifications ;
  /* creates a FORMAT Catalog from a permanent SAS dataset */
  title "Project Format Catalog" ;
QUIT ;
```

This format catalog can now be referenced by including an `OPTIONS FMTSEARCH = (libname_location)` statement at the beginning of a program.

TROUBLE-SHOOTING BETWEEN VERSIONS

Often, additional components are added to a project or study. Even if new pieces are not incorporated, there is a reasonable chance the necessary format specifications may change over time. With this in mind, it is good practice to build in a way to compare the prior version of the formats to the current version in order to see what is different.

The first step involves renaming the prior version of the format data set before creating the new one. This is a simple manual change, but thought needs to go into making sure it happens. From there, the two data sets need to be sorted to make the comparison possible.

```
PROC SORT data = permanent_location.Format_Specifications
          out = current NODUPKEY ;
  /* prepares the new version of the format data set for comparison */
  by FMTNAME TYPE START LABEL ;
RUN ;

PROC SORT data = permanent_location.Format_Specifications_Old
          out = previous NODUPKEY ;
  /* prepares the former version of the format data set for comparison */
  by FMTNAME TYPE START LABEL ;
RUN ;
```

Merging the two data sets creates and easy way to see what is in one and not the other as well as what is in both. A **PROC COMPARE** could also be used for this purpose if so desired.

```
DATA newonly oldonly same ;
  merge current ( in = in1 ) previous ( in = in2 ) ;
    by FMTNAME TYPE START LABEL ;
  if in1 and not in2 then output newonly ;
  /* isolates formats only in the current version */
  if in2 and not in1 then output oldonly ;
  /* isolates formats only in the former version */
  if in1 and in2 then output same ;
  /* isolates formats that appear in both format data sets */
RUN ;
```

Once the data has been merged in this way, it makes sense to start with the observation counts that can be found in the log. If the NEWONLY and OLDONLY data sets are empty with all records ending up in the SAME data set, then nothing has changed. However, records that may appear in the NEWONLY or OLDONLY could be examined by looking at the data sets themselves or sending that information out with a simple **PROC PRINT**.

CREATING FORMAT STATEMENTS

In addition to wanting a full format catalog, format statements specific to each SAS view or data set may also be needed or desired, depending on what will be happening with the data. This can be accomplished using OC's SDD report or equivalent information coming from spreadsheet-based specifications from the client's. The spreadsheet needs to be read in again.

```
PROC IMPORT datafile = "\\file-location\filename.xlsx"
     OUT = SDD_Report DBMS = Excel ;
     SHEET = "sheetname" ;
     MIXED = YES ;
     SCANTEXT = YES ;
RUN ;
```

As before it is worth examining what is in the report, particularly if changes or additions have been to the formats. These changes would have been isolated in the final trouble-shooting steps between versions described above.

```
PROC CONTENTS data = SDD_Report ;
  title "SDD Report" ;
RUN ;
```

The relevant variables need to be isolated from the SDD report and variables should be renamed to include meaningful information if they do not already. Additional recoding or clean-up could be done as part of this step as needed.

```
DATA SDD_Variables
     ( keep = SAS_Name SAS_Label TYPE DVG DVG_num Alpha_DVG Alpha_DVG_num ) ;
     /* keep only those variables that will be relevant to the format statements in
        later steps */
  set SDD_Report ;
    /* create meaningful variable namaes */
    rename _ =  DVG_num ;
    rename _1 = Alpha_DVG_num ;
  where DVG not in ( ' ' , '(null)' )  or Alpha_DVG not in ( ' ' , '(null)' ) ;
RUN ;
```

As part of trouble-shooting this process, it is worth checking to make sure that there are not any duplicate variables within a given view or data set. Though this should have been part of the quality control in the work done by any OC developers, it could easily be more of an issue in client-supplied specifications.

```
PROC SORT data = SDD_Variables  out = SDD_Variables_Sort  ;
  /* sort the data to prepare to check for repeated variables */
  by SAS_Name SAS_Label ;
RUN ;

DATA error ;
  set SDD_Variables ;
    by SAS_Name SAS_Label ;
  /* output records indicative of a duplication */
  if not ( first.SAS_Label and last.SAS_Label ) then output ;
RUN ;
```

The ERROR data set created as part of this process can be examined directly or easily sent out to a convenient output location using a simple **PROC PRINT** step.

Next, the isolated SDD variables need to sorted to get single records based on SAS variable and label.

```
PROC SORT data = SDD_Variables_Sort out = SDD_Variables_NoDup NODUPKEY ;
  by SAS_Name SAS_Label ;
RUN ;
```

Additional information from OC's view definition report is also required. That spreadsheet needs to be imported into SAS and turned into a temporary data set.

```
PROC IMPORT datafile = "\\file-location\filename.xlsx"
     OUT = ViewDefinition DBMS = Excel ;
     SHEET = "sheet_name" ;
     MIXED=YES ;
     SCANTEXT=YES ;
RUN ;
```

If this is the first time the format statements are being created, become familiar with what the view definition report contains in the way of variables

```
PROC CONTENTS data = ViewDefinition ;
  title "View Definition Report" ;
RUN ;
```

The OC view definition variables need to be properly sorted in order to be linked with the previously established data set. This also provides the opportunity to further winnow down the variables being kept to only the most essential.

```
PROC SORT data = ViewDefinition
          out = ViewDefinition_Sort ( keep = SAS_Name SAS_Label View_Name ) ;
  by SAS_Name SAS_Label ;
RUN ;
```

Continuing with the process of in-program trouble-shooting, it is worth checking to be sure that there are no duplication SAS labels which have been generated. Though potentially less serious than duplicate variables, having duplicate labels can create confusion when viewing data and should be avoided.

```
DATA viewdefinition_error;
  set ViewDefinition_Sort ;
    by SAS_Name SAS_Label ;
  /* output duplicates for further examination */
  if not ( first.SAS_Label and last.SAS_Label)  then output ;
RUN ;
```

As before the VIEWDEFINITION_ERROR data set can be examined in place or can be sent to an output destination via a **PROC PRINT** step.

The information from the SDD and view definition reports are now brought together. As part of this merging process, the program builds in a step to look for records that are in the SDD, but not the view definition. Hopefully, this step is something of a redundancy; however, finding such issues early can keep them from becoming bigger problems later in the project.

```
DATA SDD_ViewDefinition SDDVD_error;
  merge SDD_Variables_NoDup ( in = in1 ) ViewDefinition_Sort ( in = in2 ) ;
    by SAS_Name SAS_Label ;
  if in1 ;
  if not in2 then output error ;
  output SDD_Report ;
RUN ;
```

With all the components now combined, a final check for duplicates should be performed. This is similar to such checks as described above and can be handled or examined in the same way.

```
PROC SORT data = SDD_ViewDefinition out = SDD_VD_Sort ;
  by View_Name SAS_Name ;
RUN ;

DATA SDD_VD_error ;
  set SDD_VD_Sort ;
    by View_Name SAS_Name ;
  if not ( first.SAS_Name and last.SAS_Name ) then output ;
RUN ;
```

With a polished data set now created, the format statements themselves can be constructed. This is similar to a step in the format specifications code described earlier.

```
DATA Format_Statements ;
  set SDD_VD_Sort ( where = ( DVG ne ' ' or Alpha_DVG ne ' ' ) ) ;
  length FMTNAME $ 8 ;
  TYPE = substr(TYPE,1,1) ;
  if DVG ne ' ' then
    do ;
      select ;
           /* set DVG values based on those generated in OC or by client */
        when (DVG = "value")   FMTNAME = "FM1_" ;
             /* repeat above line for as many DVG values as needed */
      end ;
      FMTNAME = CATS ( FMTNAME , PUT( DVG_num , BEST. ) ) ;
    end ;
  if Alpha_DVG ne ' ' then
    do ;
      if Alpha_DVG = "value" then FMTNAME = "FMA_" ;
      FMTNAME = CATS ( FMTNAME , Alpha_DVG_num ) ;
    end ;
  /* creates format designations for statements including required charcater symbol
designation */
  select ;
    when (TYPE = "C" ) FMTNAME = CATS( "$" , FMTNAME , TYPE ) ;
    when (TYPE = "N" ) FMTNAME = CATS( FMTNAME , TYPE ) ;
    otherwise put "Error: invalid TYPE " TYPE= ;
  end ;
RUN ;
```

The appropriate establishment of the format names can be confirmed by running and examining frequencies on the FORMAT_STATEMENTS data set as follows.

```
PROC FREQ data = Format_Statements ;
  table DVG * DVG_num * Alpha_DVG * Alpha_DVG_num * TYPE * FMTNAME / list missing ;
  title 'Verify Assignment of FMTNAME' ;
RUN ;
```

The final version of the FORMAT_STATEMENTS data set should be saved. For convenience, consider putting it in the same location as the FORMAT_SPECIFICATIONS data set.

Complete listings of format statements for each view or data set can now be generated and stored in simple text documents. These text files can be read into programs or used on their own, depending on whether all format statements are needed for a given manipulation or if having the list serves as a reference guide.

```
DATA _null_ ;
  set Format_Statements end = eof ;
  length out_file_name $ 200 ;
  foldername = "\\file-location\" ;
  /* creates the text document */
  out_file_name = CATS ( foldername , View_Name , ".inc" ) ;
  file out_file filevar=out_file_name ;
  length line $ 200 ;
  /* creates one line per format for each variable */
  line = "format " || left ( trim( SAS_Name ) ) || "
         left ( trim ( FMTNAME ) ) || ". ;
  put @1 line ;
RUN ;
```

At this point, the format catalog and statements are ready for use. Once the code has been established for the first round of their creation, it can be easily modified or updated to accommodate client changes or alterations to the project, as well as additions of new components to the project.

CONCLUSION

The use of SAS formats makes data manipulation and presentation more flexible. Gaining those advantages is predicated upon having good formats to work with and having the information about those formats exist in an accessible, easy-to-use form. When data is being entered into OC, the system provides a variety of standard reports on the type of data and range of format values. This process takes advantage of information spread across three of those reports to generate not only a format catalog but a complete listing of format statements. Though the code here has been based on what OC produces, a client-created set of specifications which covers the same information could be used to build the same end products. Even if such information is spread over a variety of documents, this process shows how the pieces can come together to create complete format information. Trouble-shooting is an ongoing focus in the code so that discrepancies can be addressed as they arise rather than finding them in the final format catalog or statements. The files created by these programs can be easily transferred and utilized by clients with a minimal amount of direction.

ACKNOWLEDGMENTS

I would like to thank Brice Hart at Westat for making the scope of this paper possible and Rick Mitchell for encouraging me to continue to write. I would like to thank Michael Raithel and Mike Rhoads at Westat for their administrative and editorial support. I would like to thank Westat for their institutional support of my participation in both local and regional SAS users' groups.

CONTACT INFORMATION

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

Sarah Woodruff
Westat
1600 Research Boulevard
WB 401
Rockville, MD 20850
240-314-7562
sarahwoodruff@westat.com