Data Verification and Consolidation of Files with Dynamic Code Generation Using SAS Macro, Data Step Programming and System Utilities

Bill Pierce, Standard Technology, Inc., Falls Church, VA

ABSTRACT

This paper describes a process that can be used to verify and consolidate data contained in external non-SAS data sets and shows how SAS can be combined with system utilities to provide an integrated data management tool. The environment used to demonstrate the process centers around the scenario where a large number of data files, often numbering in the hundreds, must be checked for data errors and the data from all files consolidated into a single master SAS data set. The solution examined in this paper uses system utilities to dynamically generate a data set list which is passed as input to a SAS program and used to build the JCL and SAS statements to create a batch job stream to process the files. The core processing is accomplished by SAS Macro using code streams constructed in DATA steps. Although the specific examples are based on an IBM OS/390 mainframe environment, the concepts can easily be adapted for any platform.

INTRODUCTION

Let’s say that you have been given the task of checking some 300 raw data files that were just sent to your mainframe system by FTP (File Transfer Protocol). You need to verify that the data in each record is complete, report any problem records and the name of the file from which they came, and consolidate all problem-free records into a single SAS data set. And oh by the way, this is something that you are going to have to do monthly.

The goal is to develop a process that will implement a series of procedures to automatically accomplish the specific data management tasks stated above. A batch job can be constructed consisting of the following multiple steps:

1. Create a list of the raw data sets and output this list to a temporary data set.
2. Use a SAS DATA step to read the list of data sets and create an input JCL DD (data definition) statement for each data set and store this information in a member of a PDS (partitioned data set) - WORKINDS.
3. Using the DD statements stored in the WORKINDS member as input to a SAS DATA step, create a SAS MACRO call for each data set in the list and store the output in another member of a PDS - DSNMACRO.
4. Construct a SAS format to associate the DD names created in step 2 to the corresponding data set name and save this format in another member of a PDS - DSNFRMAT.
5. Read the data from each input data set, check for errors, and build the consolidated output data set. This step will use the information created and saved in steps 3 and 4.

The PDS mentioned above should be preallocated and available prior to running the process. Each of the above steps will be discussed in sequence.

GETTING STARTED

We will assume that the raw data files (referred to as data sets on mainframe systems) have a naming convention that allows you to produce a complete list by specifying a portion of the data set name. This is synonymous to having all the files contained in the same subdirectory on a PC or Unix system. The following data set names will provide a sample from which to reference in example code:

```
   RAW.INPUT.SITE0001.D021001
   RAW.INPUT.SITE0002.D021001
   RAW.INPUT.SITE0003.D021002
```

The data set names all begin with the same high level qualifier (RAW) and have the same name for the second level qualifier (INPUT). The remainder of the data set name contains information that identifies the origin of the data set and the date it was sent. Using a data set listing utility, you can list all the data sets by specifying the first two qualifiers. A consistent naming convention is necessary to be able to list all the data sets in this manner. You may need to perform an preliminary step to rename data sets so they conform to this criteria.

STEP 1 - LIST THE DATA SETS

Most operating systems provide a way to list data sets or files. On the IBM mainframe, data sets are normally cataloged and the LISTCAT command can be used to list them. LISTCAT is similar to the DIR command in PC-DOS or the ls command in Unix. In our case, we want to run the LISTCAT command in a batch process which requires that we create a step with the appropriate JCL statements. The following section of code will accomplish the first step of our process:

```
//STEP1   EXEC BATCHTSO
//SYSTPRT DD DSN=&&TMP1,DISP=(NEW,PASS),
//           UNIT=SYSDA,
//           DCB=(RECFM=FB,LRECL=132),
//           SPACE=(TRK,(10,10))
//SYSIN   DD *
LISTCAT LVL(‘RAW.INPUT')
/*
```

The EXEC BATCHTSO statement invokes a system utility that allows us to enter a TSO (Time Sharing Option) command in batch mode. TSO is the user interface similar to the DOS command line or Unix shell. When the LISTCAT command executes, it places the list of data sets that meet the criteria specified in the LVL parameter in the temporary data set &&TMP1. This temporary data set is passed to the next step. The other JCL statements define the attributes of the temporary data set - record format and length, storage device (SYSDA) and space allocations. The output of the LISTCAT command looks like the following:

```
READY
LISTC LVL('RAW.INPUT')
NONVSAM ------- RAW.INPUT.SITE0001.D021001
     IN-CAT --- CATALOG.USERCAT3
NONVSAM ------- RAW.INPUT.SITE0002.D021001
     IN-CAT --- CATAOLOG.USERCAT3
NONVSAM ------- RAW.INPUT.SITE0003.D021002
     IN-CAT --- CATAOLOG.USERCAT3
```

As you can see, there is more contained in the output than just the data set names. The next step will select out just the data set information from this output.

STEP 2 - CREATE JCL DD STATEMENTS

The DD (Data Definition) JCL statement is how you define a file reference in a batch job and is identical to the use of the FILENAME statement in SAS. The DD name establishes a reference to an external file and access to the external file within a program is made using the DD name. In this step, we want to read the output of the data set listing generated in the previous step and create a DD statement for each of the data set names. The following section of code will do this:

```
//STEP2   EXEC SAS
//IN      DD DSN=&&TMP1,DISP=(OLD,DELETE)
//OUT     DD DSN=RAW.WORK.PDS(WORKINDS),
//           DISP=SHR
//SYSIN   DD *
DATA _NULL_;
  LENGTH CHCTR $3;
  INFILE IN;
  FILE OUT;
  INPUT @17 DSN $CHAR26.;
  IF _N_=1 THEN CNTR=0;
  IF SUBSTR(DSN,1,17)='RAW.INPUT.SEP2002')
    THEN DO;
      CNTR+1;
      CHCNTR=CNTR;
      PUT @1  '//DS' CHCNTR
          @10 'DD DISP=SHR,DSN=' DSN;
    END; /* DO */
/*
```

In the code above, as each input record is read, a counter is incremented and the counter value is assigned to a character variable which is concatenated to the character string '//DS" to form the DD name. The data set name is concatenated to the end of the DD statement and is written to the output file by the PUT statement. The counter value is initialized to zero on the first record and incremented only when input records containing a valid data set name are selected by the IF SUBSTR ... statement. If you were implementing this on a PC or Unix system, the code could be modified to construct FILENAME statements. The output of the above program is written to the WORKINDS member of the PDS and an example is listed below. Note: the data set namesin the example below have been truncated due to the short line size.

```
//DS1   DD DISP=SHR,DSN=RAW.INPUT.SITE0001
//DS2   DD DISP=SHR,DSN=RAW.INPUT.SITE0002
//DS3   DD DISP=SHR,DSN=RAW.INPUT.SITE0003
```

STEP 3 - GENERATE MACRO CALLS

The SAS MACRO facility is a powerful tool for encapsulating repetitive operations and decreasing the amount of text that must be entered into a program. It is very applicable to our process where the same logic must be applied to all of the raw input data files. We can simply embed the data checking and consolidation code inside a MACRO and then repeatedly call the MACRO passing the file reference as input. Since we are trying to automate the process, it is a simple task to take the DD names generated in the previous step and create a series of MACRO calls, storing them in a file to be referenced later. The following code is the third step in our multistep batch job:

```
  IF LAST THEN
    PUT ";";
```

```
//STEP3    EXEC SAS
//IN       DD DSN=RAW.WORK.PDS(WORKINDS),
//            DISP=SHR
//OUT      DD DSN=RAW.WORK.PDS(DSNMACRO),
//            DISP=SHR
//SYSIN    DD *
DATA _NULL_;
  INFILE IN;
  FILE OUT;
  INPUT @3 DD $CHAR5.;
  PUT @1 '%DSNPROC(INFILE=' DD +(-1) ');';
/*
```

In the above code, we read the DD names as character variables and insert them into a string that forms a call to the MACRO named DSNPROC. This MACRO will be defined in step 5. Note that pointer control is required to omit the space that is automatically placed after the character variable DD when the PUT statement executes. The output of this SAS program is written to the member named DSNMACRO in the PDS and looks like the following:

```
%DSNPROC(INFILE=DS1);
%DSNPROC(INFILE=DS2);
%DSNPROC(INFILE=DS3);
```

With this code being stored in a separate data set, we can use it later in a SAS program through the use of the %INCLUDE statement. We will see this demonstrated in step 5.

STEP 4 - CREATE A DATA SET NAME FORMAT

The purpose of this step is to create a way to identify what data set a record came from. It may not be apparent why this step is required. In our implementation, when the MACRO is called, the only information it is given is the DD name, the reference to the data set. When we output data from within the MACRO, nothing is defined that allows us to output the data set name. We could have perhaps added another parameter to the MACRO call and passed both the DD name and the data set name. However, it is just as easy to dynamically create a format table, save it in a member of the PDS, and use it in the final step. The member WORKINDS in our PDS will provide the source for us to create the format which is written to the member DSNFRMAT. The code follows:

```
//STEP4  EXEC SAS
//IN     DD DSN=RAW.WORK.PDS(WORKINDS),
//          DISP=SHR
//OUT    DD DSN=RAW.WORK.PDS(DSNFRMAT),
//          DISP=SHR
//SYSIN  DD *
DATA _NULL_;
  INFILE IN END=LAST;
  FILE OUT;
  IF _N_=1 THEN
    PUT @1 'VALUE $DSMAP';
  INPUT @3  DD $CHAR5.
        @24 DSN $CHAR26.;
  PUT @3 "'"
      @4 DD $CHAR5.
      @9 "'='"
      @12 DSN $CHAR26.
      @37 "'";

/*
```

The program creates a VALUE statement that will be referenced with a %INCLUDE statement in the body of a PROC FORMAT.

Using our sample data set names, the output that is generated in the member DSNFRMAT is as follows:

```
VALUE $DSMAP
   'DS1  '='RAW.INPUT.SITE0001.D021001'
   'DS2  '='RAW.INPUT.SITE0002.D021001'
   'DS3  '='RAW.INPUT.SITE0003.D021002'
;
```

We now have a way of associating the data set reference to the actual data set name by way of a PROC FORMAT.

Step 5 - Process the Files

In this last step, we execute a SAS program that references the output of the previous steps to process all the data files. We need to have the same process performed on each file and defining a SAS MACRO is an easy and efficient way to accomplish this.

Let's first set up the necessary JCL statements that define the resources that will be needed by the SAS program. The JCL statements follow:

```
//STEP5     EXEC SAS
//INCL1     INCLUDE MEMBER=WORKINDS
//INCL2     DD DSN=RAW.WORK.PDS(DSNMACRO),
//             DISP=SHR
//FMT1      DD DSN=RAW.WORK.PDS(DSNFRMAT),
//             DISP=SHR
//OUT1      DD DSN=RAW.GOODDATA.SDS,
//             DISP=(NEW,CATLG),
//             SPACE=(TRK,(500,500))
//OUT2      DD DSN=RAW.BADDDATA.SDS,
//             DISP=(NEW,CATLG),
//             SPACE=(TRK,(500,500))
//SYSIN     DD *
```

The first JCL statement invokes the batch SAS procedure for executing the SAS program that will follow the JCL statements. The //INCL1 statement is a JCL INCLUDE statement that causes the insertion of the JCL statements contained in the member WORKINDS, the member created in step 2 that defines a data set reference for each of our input data files. This JCL INCLUDE function works similar to the way a %INCLUDE statement is used in a SAS program to insert SAS statements. However, in order to use the JCL INCLUDE statement, you must use a JCLLIB statement to define the library (the PDS) that contains the member with the JCL statements that are to be inserted. The JCLLIB statement must immediately follow the JCL JOB statement. The JOB statement is always the first statement in the batch job stream. The format of the JCLLIB statement is as follows:

```
//LIBS     JCLLIB ORDER=RAW.WORK.PDS
```

The actual use of this statement is shown in the complete job listing at the end of this paper.
The //INCL2 statement is a DD statement that defines the member containing the SAS MACRO call statements that were created in step 3. These will be referenced with a SAS %INCLUDE
The last part of the MACRO make use of PROC APPEND procedures that append the temporary data sets GOOD and BAD to the permanent output SAS data sets OUT1 and OUT2 that were defined in the JCL.

statement inbedded in the body of the SAS program.

The //FMT1 statement is a DD statement that defines the member containing the format definitions created in step 4. These definitions will be referenced in a PROC FORMAT contained in the SAS program.

The //OUT1 and //OUT2 JCL DD statements define the output files for the consolidated good and bad data records.

The //SYSIN DD * statement identifies the end of the JCL statements and the beginning of the SAS program that is discussed next.

The first part of the SAS program is a PROC FORMAT to process the data set name format defined in step 4. The PROC FORMAT contains a %INCLUDE statement referencing the DD name in the JCL representing the source VALUE statement. The code follows:

```
PROC FORMAT;
 %INCLUDE FMT1;
```

Recall that the specification of the task was to separate bad records from good and construct a consolidated master data set of all the records. The core of the SAS program that processes the data files is made up of a DATA step and PROC APPEND contained in the bounds of a SAS MACRO. The MACRO definition follows:

```
%MACRO DSNPROC(INFILE=);
  DATA GOOD BAD;
    LENGTH DSN $5 FULLDSN $44;
    INFILE &INFILE;
    DSN="&INFILE";
    FULLDSN=PUT(DSN,$DSMAP.);
    INPUT
    /* List of input variables &
        informats */

    /* Error checking code goes here */

    IF NOERROR THEN OUTPUT GOOD;
    ELSE OUTPUT BAD;
    RUN;

  PROC APPEND BASE=OUT1.ALLGOOD
   DATA=GOOD FORCE;
  PROC APPEND BASE=OUT2.ALLBAD
    DATA=BAD FORCE;
%MEND DSNPROC;
```

When the MACRO is called, it is passed a file reference (INFILE=). The first part of the MACRO is a DATA step that inputs the records and based on the error checking code directs the records to either the GOOD or BAD temporary data set. Note that we have assigned the value of the MACRO variable &INFILE which is the data set reference to the variable DSN and then used this in the PUT function to assign the full data set name to the variable FULLDSN. We now have the data set name that can be output with each record.

The only thing left to mention is the SAS code that calls the MACRO passing to it the data set reference. This is done simply by using a %INCLUDE statement that references the data set containing the list of MACRO calls that were created in step 3.

```
    %INCLUDE INCL2;
```

This statement is placed in the SAS program after the MACRO definition.

SUMMARY

Combining the five steps discussed in this paper into a single job stream eliminates the need for manually manipulating numerous data files. Each step however can be executed individually in order to accomplish the same end. In addition, each step may be adapted for use in other data management tasks. For example, renaming, copying or deleting numerous files can be accomplished using a similar batch programming approach. Although the examples in this paper are for an IBM OS/390 batch environment, similar system utilities available on PCs and Unix systems can be used with the SAS code examples in this paper to accomplish the same tasks on those platforms. As demonstrated here, the SAS programming language can be an integral part of a data management application.

Author Contact: Standard Technology, Inc.
1422 Sultan Drive
Ft. Detrick, MD 21702-5020
Ph. 301-619-7702

APPENDIX - COMPLETE BATCH JOB LISTING

```
//MYJOB JOB ACCT,PIERCE,CLASS=W,MSGCLASS=X
//LIBS  JCLLIB ORDER=RAW.WORK.PDS
//STEP1   EXEC BATCHTSO
//SYSTPRT DD DSN=&&TMP1,DISP=(NEW,PASS),
//           UNIT=SYSDA,
//           DCB=(RECFM=FB,LRECL=132),
//           SPACE=(TRK,(10,10))
//SYSIN   DD *
LISTCAT LVL('RAW.INPUT')
/*
//STEP2   EXEC SAS
//IN      DD DSN=&&TMP1,DISP=(OLD,DELETE)
//OUT     DD DSN=RAW.WORK.PDS(WORKINDS),
//           DISP=SHR
//SYSIN   DD *
DATA _NULL_;
  LENGTH CHCTR $3;
  INFILE IN;
  FILE OUT;
  INPUT @17 DSN $CHAR26.;
  IF _N_=1 THEN CNTR=0;
  IF SUBSTR(DSN,1,17)='RAW.INPUT.SEP2002')
    THEN DO;
      CNTR+1;
      CHCNTR=CNTR;
      PUT @1  '//DS' CHCNTR
          @10 'DD DISP=SHR,DSN=' DSN;
    END; /* DO */
/*
//STEP3    EXEC SAS
//IN       DD DSN=RAW.WORK.PDS(WORKINDS),
//            DISP=SHR
//OUT      DD DSN=RAW.WORK.PDS(DSNMACRO),
//            DISP=SHR
//SYSIN    DD *
DATA _NULL_;
  INFILE IN;
  FILE OUT;
  INPUT @3 DD $CHAR5.;
  PUT @1 '%DSNPROC(INFILE=' DD +(-1) ');';
/*
//STEP4  EXEC SAS
//IN     DD DSN=RAW.WORK.PDS(WORKINDS),
//          DISP=SHR
//OUT    DD DSN=RAW.WORK.PDS(DSNFRMAT),
//          DISP=SHR
//SYSIN  DD *
DATA _NULL_;
  INFILE IN END=LAST;
  FILE OUT;
  IF _N_=1 THEN
    PUT @1 'VALUE $DSMAP';
  INPUT @3  DD $CHAR5.
        @24 DSN $CHAR26.;
  PUT @3 "'"
      @4 DD $CHAR5.
      @9 "'='"
      @12 DSN $CHAR26.
      @37 "'";
  IF LAST THEN
    PUT ";";
/*
//STEP5    EXEC SAS
//INCL1    INCLUDE MEMBER=WORKINDS
//INCL2    DD DSN=RAW.WORK.PDS(DSNMACRO),
//            DISP=SHR
//FMT1     DD DSN=RAW.WORK.PDS(DSNFRMAT),
//            DISP=SHR
//OUT1     DD DSN=RAW.GOODDATA.SDS,
//            DISP=(NEW,CATLG),
//            SPACE=(TRK,(500,500))
//OUT2     DD DSN=RAW.BADDDATA.SDS,
//            DISP=(NEW,CATLG),
//            SPACE=(TRK,(500,500))
//SYSIN    DD *
PROC FORMAT;
  %INCLUDE FMT1;
%MACRO DSNPROC(INFILE=);
  DATA GOOD BAD;
    LENGTH DSN $5 FULLDSN $44;
    INFILE &INFILE;
    DSN="&INFILE";
    FULLDSN=PUT(DSN,$DSMAP.);
    INPUT
    /* List of input variables &
        informats */

   /* Error checking code goes here */

   IF NOERROR THEN OUTPUT GOOD;
   ELSE OUTPUT BAD;
   RUN;
   PROC APPEND BASE=OUT1.ALLGOOD
      DATA=GOOD FORCE;
   PROC APPEND BASE=OUT2.ALLBAD
     DATA=BAD FORCE;
%MEND DSNPROC;
%INCLUDE INCL2; /* End of Job */
```