Paper 96-25

"Functioning" JCL Into a SAS® Relational Database Table: Your Portable Tutorial On Character Functions, Plus an MVS™ Batch Bonus

Doug Zirbel, Modis, Inc., Bloomington, MN

ABSTRACT

This SAS 6.09e Base SAS software program running on IBM® mainframe OS390™ (MVS) gives you plenty of explained examples of useful SAS string functions and others, as well as a handy PROC REPORT example. As an added bonus, it crunches that old mainframe mystery, JCL (Job Control Language), into a denormalized SAS file (table) / report which almost anyone in your IBM mainframe shop can understand.

INTRODUCTION

The job you see here I wrote to fill a need for my Y2K system testing team at a Fortune 500 company. We needed to rapidly analyze a great number of production batch jobstreams. In particular, we had to find out what a jobstream's input files were as well as its output files, and trace the history of those files' dispositions. If you have ever tried to comprehend lots of JCL by looking at it, you know the tedium involved. We simplified our work by running this code and downloading its output to a MS/Access database for the whole team.

The code itself contains an abundance of SAS character function examples as well as other useful Base SAS features.

These are the sections of that job.

1. Getting the TSO user name of the submitter (you?) and the batch job number into macro variables which you can use in the title of your report.

2. Getting the name of the external file (in this case it's a PDS) your job is reading into a macro variable you can use in the title of your report.

3. Using the little-known SAS proc called PROC SOURCE to turn the members of the PDS into flat files for the next step. The PDS, incidentally, contains one member for each batch job. You must hardcode the member names, in order of execution, for the jobstream you want to analyze.

4. A number of SAS functions to unstring and analyze these individual jobs and put their attributes into a big, whole-jobstream SAS file. The SAS file is in this case no different than a slightly denormalized relational database table, which offers the flexibility of later SQL reporting, if that is what is desired.

5. Writing the data in that table with PROC REPORT into an easy-to-understand report on the flow of the files in the jobstream.

6. Writing the data in the table with a "Data _null_ step" to another tab-delimited flat file (in EBCDIC) for downloading with a file transfer utility to the PC for MS/Access or MS/Excel.

You should have some familiarity with the IBM mainframe batch environment. On the other hand, you aren't expected to know a lot about Base SAS beyond the casual-user level. And although the examples here are indeed from the mainframe environment, many of the tricks are transferable to other operating systems.

MAIN LOGIC OF THE PROGRAM

For each DD statement in the jobstream your program is reading, one SAS file observation will be written. Note that to denormalize the table for ease of use, there is not one type of observation – nor a separate table – for a JOB statement, another for an EXEC statement and still another for a DD statement. Rather, there is one output observation per DD statement which also contains that DD's step and job information.

Therefore, an INPUT statement reading a JOB or an EXEC line will *not* output an observation at that point, but will RETAIN that information. When it gets to a DD statement, it may need to read several lines before it comes to the end of that DD's parameters. At that point, however, it writes out one observation containing each of the following columns, or variables:

DSN
job sequence number (useful if there is more than one job in the jobstream)
job name
step number within the job (not all step names in JCL are always in sequence)
step name
program or proc EXECuted
DDNAME
file DISPositions
RECFM
LRECL
BLKSIZE
UNIT
SPACE
PDS
and a multi-purpose note field.

JCL for the job itself

```
//YOURTSOJ JOB (08000C),'UNSTRING JCL,CLASS=A,MSGCLASS=X
//*************************************************************
//* READ PDS INTO FLAT FILE, UNSTRING W/SAS TO GET DSN INFO
//* ... ALSO PRINT REPORT OF EXTERNAL INPUT DSNS
//*************************************************************
//STEP05   EXEC SAS,OPTIONS='LS=132 PS=60  NOCENTER'
//*
//************ => YOUR PDS GOES BELOW ("PDS") ***********
//*
//PDS      DD DSN=YOURTSO.XYZ.JCLPDS,DISP=SHR
//*
//*
//SASLIST  DD SYSOUT=*
//FLATOUT  DD DSN=&&TEMP,DISP=(,DELETE),
//            SPACE=(TRK,(5,5),RLSE),DCB=(LRECL=80,RECFM=FB)
//OUT1     DD DSN=YOURTSO.DOWNL260,DISP=OLD
//SASDB    DD DSN=&&SASDB,DISP=(,PASS),SPACE=(CYL,(5,5)),
//            DCB=(LRECL=6144,RECFM=FS)  * COULD BE PERM
//SYSIN    DD  *
```

1) Getting the batch job number into a macro variable

This gets the job number and TSO user id from internal o/s file. The PEEK and PEEKC functions locate memory. We also encounter the substring function. **SUBSTR** means, "starting from a particular position, get a given number of characters." Here it says,
"in the field called 'JESTRING', starting at position 21 get 8 characters and put them into the field called 'JOBNUM'.
The TSO user ID can also be found in this same string.

CALL SYMPUT puts a SAS variable into a macro variable which can be used later in this job. These macro variables (job number and TSO ID) will be referred to in a later TITLE statement by &JOBNUM and &USER.

```
DATA _NULL_;
    LENGTH JESTRING $100 JOBNUM USER $08;
    ASCBADDR=PEEK(548,4);
    ASSBADDR=PEEK(ASCBADDR+336,4);
    JSABADDR=PEEK(ASSBADDR+168,4);
    JESTRING=PEEKC(JSABADDR,100);

    JOBNUM=SUBSTR(JESTRING,21,8);
    CALL SYMPUT('JOBNUM',JOBNUM);
    USER=SUBSTR(JESTRING,45,8);
    CALL SYMPUT('USER',USER);
RUN;
```

2) Getting the name of the external file

The useful but not-widely-known SAS SQL DICTIONARY TABLES allow you to find the PDS file name. Note that there will be only one observation in the DDTABLE. Be sure to try the DESCRIBE statement by itself sometime!

Some date functions are used here to put a date and a time into macro variables. The **DATETIME()** function extracts the system timestamp. To use the date part of it, you use the **DATEPART** function, and for the time part, you use the **TIMEPART** function.

The **PUT** function changes a numeric variable into a character variable with a particular format and is used here to put text into the macro variables for date and time.

To put the macro variables (from the CALL SYMPUT statements) into your title, you must use double-quotes.

```
PROC SQL;
   *DESCRIBE TABLE DICTIONARY.EXTFILES;

    CREATE TABLE DDTABLE AS
    SELECT FILEREF, XPATH FROM DICTIONARY.EXTFILES
    WHERE FILEREF='PDS';
QUIT;

DATA _NULL_;
    LENGTH RUNTIME $8 RUNDATE $29;
    SET DDTABLE;
        CALL SYMPUT('PDS',XPATH);
        RUNSTAMP=DATETIME();
        RDATE=DATEPART(RUNSTAMP);
        RTIME=TIMEPART(RUNSTAMP);
        RUNTIME=PUT(RTIME,TIME8.);
        RUNDATE=PUT(RDATE,WEEKDATE29.);
        CALL SYMPUT('RUNDT',RUNDATE);
        CALL SYMPUT('RUNTM',RUNTIME);
RUN;

FOOTNOTE
"* YOURCORP, USER=&USER &JOBNUM, JOBID=&SYSJOBID  &RUNDT
&RUNTM";
```

3) Turn the members of the PDS into flat files

PROC SOURCE is the tool for working with PDSs in SAS. Here, it unloads desired PDS members to a flatfile with a SELECT statement. For help on PROC SOURCE, type HELP HOST in SAS display manager.

INDD and OUTDD refer to DDNAMEs in your JCL even if you did not have a dictionary tables step for it as we did above.

```
*********************************************************;
*  => ENTER JOBS, IN ORDER, THAT YOU WANT PULLED FROM THE PDS.
*********************************************************;
PROC SOURCE INDD=PDS /*NOPRINT*/ NOTSORTED OUTDD=FLATOUT;
    SELECT MIJ3561X
           MIJ9999W
           ;
RUN;
```

4) SAS functions to unstring and analyze these individual jobs and put their attributes into a big, whole-jobstream SAS file

```
*********************************************************;
* READ THAT FLAT FILE FOR JOB, STEP, DSN INFO
*********************************************************;
DATA SASDB.DSNINFO
       (KEEP=JOBNAME STEPNAME STEPNUM PGM DDNAME DSN DISP1 DISP2
             DISP3 UNIT SPACE LRECL RECFM BLKSIZE PDS NOTEFLD
             DISPNUM SEQ);
   LENGTH LINE1 $100
          PGM JOBNAME STEPNAME DDNAME UNIT $8 RECFM DISP1 $3
          DISP2 DISP3 $6 SPACE $35 DSN $44 LRECL $4 BLKSIZE $5
          PDS $44 DDWAITNG DISPNUM $1 NOTEFLD $37 SEQ 04;
   RETAIN PDS JOBNAME STEPNAME PGM DDNAME DSN DISP1 DISP2 DISP3
          SPACE LRECL RECFM BLKSIZE UNIT DDWAITNG NOTEFLD
          DISPNUM SEQ;
```

If it's the first time through this DATA step, the DDWAITNG switch is initialized, and will be subsequently used for telling the program when it has finally read to the end of a DD statement. SEQ tells Job sequence, important in a jobstream with more than one jobs in it.

```
    IF _N_ = 1 THEN DO;
        PDS="&PDS";
        DDWAITNG='N';
        SEQ=0;
    END;

    INFILE FLATOUT MISSOVER END=LASTREC;
    INPUT @01 LINE1    $CHAR80.
          @01 JOBIND   $CHAR12.
          @01 FLD01    $CHAR02.
          @03 FLD03    $CHAR08.
          @21 FLD21    $CHAR08. ;
```

The next line deserves some explanation: as you will see, an observation for a DD statement is not written out until you bump into the *next* DD or JOB or EXEC statement. This LASTREC statement handles the special end-of-file case where there are no more DD or JOB or EXEC statements to bump into.

```
    IF  LASTREC AND DDWAITNG = 'Y' THEN OUTPUT;
    IF  FLD01='//' AND FLD03=:'*' THEN DELETE;
```

By the way, that **=:** means 'begins with', and it is used here to prevent JCL comment lines from getting into your output file.

```
    *********************************************************;
* GET JOBNAME
*********************************************************;
    IF  JOBIND = './       ADD' THEN DO;
```

The '**./** ADD' is put there by PROC SOURCE.

The DDWAITING switch is turned on when a DD is first encountered. If it is 'on' here, you have come to the end of a DD statement and you can write out everything you now know

about that DD statement as a SAS observation, then re-set the switch.

Whenever you get to a JOB statement, you need to reinitialize the PGM variable as well. STEPNUM is needed because JCL steps do not always reveal what order they execute in. SEQ, again, identifies the order of the job currently being read by your program within the overall jobstream you are analyzing.

```
        IF  DDWAITNG = 'Y' THEN DO;
            OUTPUT;
            DDWAITNG='N';
        END;
        PGM= ' ';
        STEPNUM = 0;
        JOBNAME = FLD21;
        SEQ+1;
    END;
```

Now (else) test to see if you are about to read an EXEC statement

```
    ELSE
    IF  INDEX(LINE1,'EXEC ') NE 0 THEN DO;
```

The **INDEX** function looks for a string of characters (in quotes) and tells you the position of its first character. If it does not find your string it returns a 0. The statement above means
"If LINE1 contains the string EXEC, then do something".

In this case it means you have come to the end of another DD statement and the beginning of a new step.

```
*************************************************************;
* GET STEPNAME & PGM
*************************************************************;
        IF  DDWAITNG = 'Y' THEN DO;
            OUTPUT;                  /* OUTPUT PREV DD OBS */
            DDWAITNG='N';
        END;
        STEPNUM+1;  /*NEW STEP, SO INCREMENT STEP NUMBER */
        EXECLOC=INDEX(LINE1,'EXEC ');  /* SAVE THE POSTION
OF 'EXEC' */
```

A JCL step may execute a PGM or a cataloged JCL procedure. If it is a PGM, it says so: 'PGM=XYZ', but if a proc, it may say 'EXEC XYZ' *or* 'EXEC PROC=XYZ'. Either way, the **INDEX** function would not find a 'PGM='. Here, if you find a PGM (and the position of 'P'), you next use the **INDEXC** function to and the substring function together to find the character that comes after PGM name.

Here, **SUBSTR** means
"make PGMSTRNG equal to 14 characters from LINE1, beginning in position PGMLOC."
PGMSTRNG is thus a variable which is 14 bytes long.

INDEXC means
"find the position of the 1st of <u>any</u> of the characters in quotes."
Here it is looking for a comma or a space, and its position number will go into ENDPUNCT. The two functions could have been combined into 1 statement:

ENDPUNCT=INDEXC(SUBSTR(LINE1,PGMLOC,14),',') ');

```
        PGMLOC = INDEX(LINE1,'PGM=');
        IF  PGMLOC NE 0 THEN DO;
            PGMSTRNG=SUBSTR(LINE1,PGMLOC,14);
            ENDPUNCT=INDEXC(PGMSTRNG,', ');
```

Here, to fill the 8-byte PGM variable, we substring: "from line1, starting 4 characters after PGMLOC (remember: PGMLOC is where 'PGM=' begins in LINE1, & 4 bytes *after that* is where the PGM name begins!), take as many bytes as are equal to the position of the comma or space identified by ENDPUNCT minus 5. This is just another way of pointing (backwards) to the 1st byte after the '=' sign in 'PGM='. Look at this **ruler line illustration** and this will make sense.

```
----+----1----+----2----+----3----+----4--- . . .
//STEP010  EXEC PGM=XYZ00001
                 |   |
                 15  19
  ----+----1--|-
  PGM=XYZ00001     <= this is what is in PGMSTRNG.
```

So – ENDPUNCT = 13 <u>of PGMSTRNG</u>, which is the location of the 1st blank in PGMSTRNG.

```
            PGM=SUBSTR(LINE1,(4+PGMLOC),(ENDPUNCT-5));
        END;
```

which really means:
PGM=SUBSTR(LINE1,(4+15),(13-5));
or, PGM=SUBSTR(LINE1,19,8);

```
*************************************************************;
* ASSUME A PROC (NO DDS APPEAR), SO WRITE IT OUT ANYWAY
*************************************************************;
        ELSE DO;
            DSN='';   DISP1='';  DISP2='';   DISP3='';
            LRECL=''; RECFM='';  BLKSIZE=''; UNIT='';  SPACE='';
            DDNAME='';DISPNUM='';
            PGM=SCAN(LINE1,3);
            STEPNAME=FLD03;
            NOTEFLD='** CATALOGED PROC **';
            OUTPUT;
        END;
        STEPNAME=FLD03;
    END;
```

SCAN = "SCAN-FOR-WORDS". Just remember that. It comes with a list of characters that normally mark the <u>end</u> of words, but you can also supply your own. The defaults are:
`Blank .<(+|&!$*);¬-/,%|¢`

Above, it says,
"get the third word from LINE1."
It is expecting to see: //STEPABC EXEC PROCXYZ

```
ELSE
    IF  INDEX(LINE1,' DD ') NE 0 THEN DO;

*************************************************************;
* GET DDNAME & DSN FROM 'DD' LINE, THEN SEARCH SUCCESSIVE LINES
*************************************************************;
        IF  DDWAITNG = 'Y' THEN OUTPUT;
        ELSE DDWAITNG = 'Y';

        IF  FLD03 GT ' ' THEN DDNAME=FLD03;
*************************************************************;
* RE-INITIALIZE RETAINED FLDS
*************************************************************;
        DSN='';   DISP1='';  DISP2='';   DISP3=''; NOTEFLD='';
        LRECL=''; RECFM='';  BLKSIZE=''; UNIT='';  SPACE='';
        DISPNUM='';
```

The code above begins the DD statement analysis in earnest. It uses the INDEX function to search for a ' DD ' string, and if a previous DD statement has not yet been written but its RETAINed variables are 'waiting', then write out that previous DD statement observation now and get ready for this current DD statement's variables.

```
*************************************************************;
* ID 'DUMMY', 'DD *', 'SYSOUT=' DSN ALTERNATIVES
*************************************************************;
    DDLOC=INDEX(LINE1,' DD ');
    SYSLOC= INDEX(SUBSTR(LINE1,DDLOC,51),'SYSOUT=');
    IF SYSLOC GT 0 THEN DSN=SUBSTR(LINE1,(SYSLOC+DDLOC-1),16);
    ELSE DO;


*************************************************************;
* THIS HANDLES THE 'DD *'  POSSIBILITY
*************************************************************;
            ASTERLOC= INDEX(SUBSTR(LINE1,DDLOC,51),' *');
            IF ASTERLOC GT 0 THEN DO;
               DSN=LEFT(SUBSTR(LINE1,(ASTERLOC+DDLOC-1),4));
            END;
            ELSE DO;
```

```
*******************************************************;
* THIS HANDLES THE 'DD DUMMY'  POSSIBILITY
*******************************************************;
                DUMLOC= INDEX(LINE1,'DUMMY');
                IF DUMLOC GT 3 THEN DO;
                   DSN=LEFT(SUBSTR(LINE1,DUMLOC,5));
                   NOTEFLD='** DUMMY FILE **';
                   OUTPUT;
                   RETURN;
                END;
            END;
        END;
```

DDLOC is the position of the space before DD. SYSLOC (if there is one) is the position in the 51-char field (beginning with DDLOC) where SYSOUT= begins. The DSN of a SYSOUT= is presumed to be no greater than 16 bytes. See the PGMLOC example above for a picture of how this works.

LEFT left-justifies whatever may be in a particular field, removing leading blanks. Notice the compounding of two functions into one statement.

```
*******************************************************;
* PERFORM THE STRING SEARCH SECTION ON THE LINE
*******************************************************;
        LINK FILEATTS;
    END;

    ELSE
    IF  FLD01= '//' AND FLD03='         ' THEN LINK FILEATTS;
    ELSE
    IF (FLD01 NE '//' AND DDNAME='SYSTSIN' AND
                      PGM='IKJEFT01')
    OR (FLD01 NE '//' AND DDNAME='SYSTSIN' AND
                      PGM='IKJEFT1B') THEN DO;
        CARDLOC=INDEX(LINE1,'RUN PROGRAM(');
        IF CARDLOC GT 0 THEN DO;
            NOTEFLD=SUBSTR(LINE1,CARDLOC,37);
        END;
    END;
RETURN;
```

At this point (above), you have read the first line of a DD statement. That line may have some or all of the DD's parameters on it, in which case you will execute the LINKed section (FILEATTS) which looks for any of those which might exist. If this pass of the program is reading a second or third line of a DD statement it will still execute that LINKed section.

On the other hand, it may have encountered instream control cards and these lines are written to the NOTEFLD variable. In a special case, if the previous PGM= statement was IKJEFT01 (batch TSO pgm), the step actually executes another program indicated in these control cards.

```
*******************************************************;
* PERFORMED (LINKED) SECTION, LOOKS FOR FILE ATTRS ON EA
LINE
*******************************************************;
FILEATTS:
    DISPLOC=INDEX(LINE1,'DISP=');
    LRECLOC=INDEX(LINE1,'LRECL=');
    RECFLOC=INDEX(LINE1,'RECFM=');
    BLKSLOC=INDEX(LINE1,'BLKSIZE=');
    UNITLOC=INDEX(LINE1,'UNIT=');
    SPACLOC=INDEX(LINE1,'SPACE=');
    DSNLOC=INDEX(LINE1,'DSN=');
    IF  DSNLOC NE 0 THEN DO;
        ENDPUNCT=INDEXC(LINE1,',');  /* FIND FIRST COMMA */
        IF  ENDPUNCT GT 0 THEN
            DSN=SUBSTR(LINE1,4+DSNLOC,(ENDPUNCT-DSNLOC-4));
        ELSE DSN=SUBSTR(LINE1,4+DSNLOC,44);
    END;

    IF  DISPLOC NE 0 THEN DO;
        DISPFLD=SUBSTR(LINE1,5+DISPLOC,21);
            /* LONGEST POSSIBLE: (OLD,UNCATLG,UNCATLG)) */
        IF  SUBSTR(DISPFLD,1,1) NE '(' THEN
            DISP1=SUBSTR(DISPFLD,1,3);
        ELSE
        IF  SUBSTR(DISPFLD,1,2) EQ '(,' THEN DO;
            DISP1='NEW';
            DISP2=SCAN(DISPFLD,1);
            RPAREN=INDEXC(SUBSTR(DISPFLD,3,19),')');
            RCOMMA=INDEXC(SUBSTR(DISPFLD,3,19),',');
            IF (RPAREN < RCOMMA) OR RCOMMA=0 THEN DISP3='';
            ELSE DISP3=SCAN(DISPFLD,2);
 END;
        ELSE DO;  /* DISP FLD DOES BEGIN WITH ( */
            DISP1=SCAN(DISPFLD,1);
            DISP2=SCAN(DISPFLD,2);
            RPAREN=INDEXC(SUBSTR(DISPFLD,6,16),')');
            RCOMMA=INDEXC(SUBSTR(DISPFLD,6,16),',');
            IF (RPAREN < RCOMMA) OR RCOMMA=0 THEN DISP3='';
            ELSE DISP3=SCAN(DISPFLD,3);
        END;
        IF  DISP1='NEW' THEN DISPNUM='1';
        ELSE IF  DISP1='MOD' THEN DISPNUM='2';
        ELSE IF  DISP1='OLD' THEN DISPNUM='3';
        ELSE IF  DISP1='SHR' THEN DISPNUM='4';
    END;
    IF  LRECLOC NE 0 THEN DO;
        ENDPUNCT=INDEXC(SUBSTR(LINE1,LRECLOC,12),',') ');
        LRECL=(SUBSTR(LINE1,(6+LRECLOC),(ENDPUNCT-7)));
    END;
    IF  RECFLOC NE 0 THEN DO;
        ENDPUNCT=INDEXC(SUBSTR(LINE1,RECFLOC,10),',') ');
        RECFM=(SUBSTR(LINE1,(6+RECFLOC),(ENDPUNCT-7)));
    END;
    IF  BLKSLOC NE 0 THEN DO;
        ENDPUNCT=INDEXC(SUBSTR(LINE1,BLKSLOC,14),',') ');
        BLKSIZE=(SUBSTR(LINE1,(8+BLKSLOC),(ENDPUNCT-9)));
    END;
    IF  UNITLOC NE 0 THEN DO;
        ENDPUNCT=INDEXC(SUBSTR(LINE1,UNITLOC,18),', ');
        UNIT=(SUBSTR(LINE1,(5+UNITLOC),(ENDPUNCT-6)));
    END;
    IF  SPACLOC NE 0 THEN DO;
        ENDPUNCT=INDEX(SUBSTR(LINE1,SPACLOC,35),', ');
        IF ENDPUNCT=0 THEN
            ENDPUNCT=INDEX(SUBSTR(LINE1,SPACLOC,35),'
')+SPACLOC;
        SPACE=(SUBSTR(LINE1,(6+SPACLOC),(ENDPUNCT-7)));
    END;
RETURN;          /* RETURN TO LINK STMT */
RUN;
```

The key to understanding the FILEATTS section is to picture each string as in the 'ruler' illustration above.

5) Writing the data in that table with PROC REPORT

```
*************************************************************;
* FILTER OUT UNWANTED I/O AS DESIRED
*************************************************************;
DATA DSNINFO;
    SET SASDB.DSNINFO;
    WHERE DSN NE ' ' AND DSN NE 'DUMMY'
      AND DSN NOT =: 'SYSOUT='
   /* AND DSN NOT =: '*'          */
      AND DSN NOT LIKE '%.LOADLIB'
      AND DSN NOT LIKE '%.COB2LIB'
      AND DSN NOT LIKE '%.LINKLIB'
      AND DSN NOT LIKE '%CNTLCARD%';
TITLE2 "    (WHERE DSN NOT BLANK, DUMMY, SYSOUT, CNTLCARD, OR
LOADLIB)";
RUN;


%MACRO JOB;
*************************************************************;
* PRODUCE JOB SEQ REPORT                                    *
*************************************************************;
PROC REPORT DATA=DSNINFO NOWINDOWS HEADLINE HEADSKIP;
    COLUMNS SEQ JOBNAME STEPNUM STEPNAME PGM DDNAME DSN DISP1
DISP2;
    DEFINE SEQ  / ORDER WIDTH=3;
    DEFINE JOBNAME / ORDER WIDTH=8;
    DEFINE STEPNUM / ORDER NOPRINT;
    DEFINE STEPNAME / ORDER   WIDTH=8;
    DEFINE PGM / ORDER WIDTH=8;
    DEFINE DDNAME / DISPLAY WIDTH=8;
    DEFINE DSN  / DISPLAY WIDTH=44;
    DEFINE DISP1 / DISPLAY WIDTH=5;
    DEFINE DISP2 / DISPLAY WIDTH=6;
    BREAK AFTER STEPNUM / SKIP;
TITLE "*** &PDS.      DATABASE  (BY JOB SEQUENCE) ***";
TITLE2 "    (WHERE DSN NOT BLANK, DUMMY, SYSOUT, CNTLCARD, OR
LOADLIB)";
RUN;
%MEND;
```

A few words about this PROC REPORT for anyone unfamiliar with it. This proc begins with the PROC REPORT statement and some parameters indicating that it is batch, that the headings should be underlined and that the headings should be followed by a blank (HEADSKIP) line before detail lines appear.

The next statement, COLUMNS, names the columns in left-to-right order.

Finally, the DEFINEs statements define each column a little further. The / ORDER option indicates that the data is to be sorted by this variable or column. It is like a BY statement in a PROC SORT, and each succeeding DEFINE with an / ORDER is like another by-variable (from left to right).

The BREAK AFTER STEPNUM / SKIP writes a blank line after the STEPNUM changes in control-break fashion.

There are other report macros you might include in this job. For example you could write a macro to produce a report ORDERed by DSN, SEQ, and STEPNUM. Thus you could follow a dataset throughout its life within a jobstream. You could also filter out all but the first appearances of each dataset – then wherever the DISP was not NEW, you would know that that would probably be an input dataset to the jobstream.

6) Creating a tab-delimited flat file for downloading to Excel

```
%MACRO DOWNLOAD;
************************************************************;
* CREATES TAB-DELIM JOB-SEQENCE FILE FOR MS/ACCESS
************************************************************;
PROC SORT DATA=DSNINFO OUT=DSNINFO;
    BY SEQ JOBNAME STEPNUM STEPNAME;

DATA _NULL_;
    LENGTH T $1;
    T='05'X;       /* EBCDIC REPRESENTATION OF TAB */
    SET DSNINFO;
    FILE OUT1;
    IF _N_=1 THEN
    PUT @001 'DSN'      @045 T
        @046 'SEQ'      @049 T
        @050 'JOBNAME'  @058 T
        @059 'S#'       @062 T
        @063 'STEPNAME' @071 T
        @072 'PGM'      @080 T
        @081 'DDNAME'   @089 T
        @090 'DSP'      @093 T
        @094 'DISP2'    @100 T
        @101 'DISP3'    @107 T
        @108 'RCF'      @111 T
        @112 'LRCL'     @116 T
        @117 'BLKSZ'    @122 T
        @123 'UNIT'     @131 T
        @132 'SPACE'    @177 T
        @178 'PDS'      @222 T
        @223 'NOTEFLD';

    PUT @001 DSN        @045 T
        @046 SEQ     3. @049 T
        @050 JOBNAME    @058 T
        @059 STEPNUM 3. @062 T
        @063 STEPNAME   @071 T
        @072 PGM        @080 T
        @081 DDNAME     @089 T
        @090 DISP1      @093 T
        @094 DISP2      @100 T
        @101 DISP3      @107 T
        @108 RECFM      @111 T
        @112 LRECL      @116 T
        @117 BLKSIZE    @122 T
        @123 UNIT       @131 T
        @132 SPACE      @177 T
        @178 PDS        @222 T
        @223 NOTEFLD;
RUN;

%MEND;
```

Now execute the two macros at the end of your job with

```
%JOB;
%DOWNLOAD;

/*
//
```

CONCLUSION

SAS character and other functions can allow you to read and then re-format information from other systems, such as OS/390 JCL, to make it more easily understood. The program in this paper was intended to illustrate this and to, by way of example, introduce and explain the functions and other techniques which were actually used.

TRADEMARKS

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. IBM, OS/390, and MVS are registered trademarks of International Business Machines Inc.

REFERENCES

SAS Language: Reference, Version 6, First Edition, #C56076, Cary, NC: SAS Institute Inc.
SAS Companion For The MVS Environment, #C55108, Cary, NC: SAS Institute Inc.
Nichols, Randall M. (1997),"Using SAS Software To Compare Strings Of Volsers In A JCL Job And A TSO CLIST," *Proceedings of SUGI 22*, 483-488.

CONTACT INFORMATION

Doug Zirbel
zirbel@computerpro.com
(218) 525-5518

APPENDIX: (Output sample)

```
*** ABC.TEST.JCL..PDS DATABASE  (BY JOB SEQUENCE) ***                                                    1
    (WHERE DSN NOT BLANK, DUMMY, SYSOUT, CNTLCARD, OR LOADLIB)              12:54 Wednesday, June 30, 1999

  SEQ  JOBNAME   STEPNAME  PGM       DDNAME    DSN                                                DISP1  DISP2
  ---------------------------------------------------------------------------------------------------------

    1  MIJ3C18D  STEP020   MIP818FX  IN234     ABC.XYZF.RCVG.IN234.                               SHR
                                     RPT818A   ABC.XYZF.MIJ3C18D.RPT818A.                       NEW    CATLG
                                     RPT01     ABC.XYZF.RCVG.EXT.RPT.FILE.RPT01.                NEW    CATLG
                                     SYSIDMS   IDMS.CTRL.SYSIDMS(E18)                           SHR
                                     INP11     ABC.CTRL01.DIV.CTRL.INP11.CLUSTER                SHR

                  STEP030   MIP694FX  SYSIDMS   IDMS.CTRL.SYSIDMS(E18)                           SHR
                                     INP11     ABC.CTRL01.DIV.CTRL.INP11.CLUSTER                SHR
                                     IN234     ABC.XYZF.RCVG.IN234.                             SHR
                                     RPT694A   ABC.XYZF.MIJ3C18D.RPT694A.                       NEW    CATLG
                                     SORTWK01  &&SORTWK                                         NEW    PASS
                                     SORTWK02  &&SORTW2                                         NEW    PASS
                                     SORTWK03  &&SORTW3                                         NEW    PASS

                  STEP050   MIP697    IN271     ABC.XYZF.DIRECT.ORD.IN271.                       NEW    CATLG
                                     SYSIDMS   IDMS.CTRL.SYSIDMS(E18)                           SHR
                                     INP11     ABC.CTRL01.DIV.CTRL.INP11.CLUSTER                SHR

                  STEP060   SORT      SORTIN    ABC.XYZF.DIRECT.ORD.IN271.                       SHR
                                     SORTOUT   ABC.XYZF.DIRECT.ORD.SRTD.IN271.                  NEW    CATLG

                  STEP070   MIP695    IN271     ABC.XYZF.DIRECT.ORD.SRTD.IN271.                  OLD    KEEP
                                     RPT695A   ABC.XYZF.MIJ3C18D.RPT695A.                       NEW    CATLG
                                     SYSIDMS   IDMS.CTRL.SYSIDMS(E18)                           SHR
                                     INP11     ABC.CTRL01.DIV.CTRL.INP11.CLUSTER                SHR

    2  MIJW418D  STEP010   IEFBR14   FILE1     ABC.XYZF.FLOOR.STOCK.LCRD.                       MOD    DELETE
                                     FILE2     ABC.XYZF.PROM.HIST.SELECT.                       MOD    DELETE
                                     FILE3     ABC.XYZF.PROM.HIST.SELECT.SORT.                  MOD    DELETE
                                     FILE4     ABC.XYZF.PRC.DECLN.EXTRACT.SORT.                 MOD    DELETE
                                     FILE5     ABC.XYZF.PRC.DECLN.EXTRACT.                      MOD    DELETE
                                     FILE6     ABC.XYZF.MERGED.PRC.DECLN.                       MOD    DELETE
                                     FILE7     ABC.XYZF.RCVG.ITM.SELECT.                        MOD    DELETE
                                     FILE8     ABC.XYZF.RCVG.ITM.SELECT.SORT.                   MOD    DELETE
                                     FILE9     ABC.XYZF.MERGED.PRC.DECLN.SORT.                  MOD    DELETE

                  STEP020   MIP847    SYSIDMS   IDMS.CTRL.SYSIDMS(E18)                           SHR
                                     INP11     ABC.CTRL01.DIV.CTRL.INP11.CLUSTER                SHR
                                     INP01O    ABC.XYZF.FLOOR.STOCK.LCRD.                       NEW    CATLG
                                     IN211     ABC.TV0018.ABCER.IN.IN211.CLUSTER                SHR
                                     IN2847    ABC.XYZF.PROM.HIST.SELECT.                       NEW    CATLG

                  STEP030   SORT      SORTIN    ABC.XYZF.PROM.HIST.SELECT.                       SHR
                                     SORTOUT   ABC.XYZF.PROM.HIST.SELECT.SORT.                  NEW    CATLG

* YOURCORP, USER=YOURTSO   JOB02937, JOBID=YOURTSOS9         Wednesday, June 30, 1999 13:12:36
```