CHAPTER 16 Procedures

Procedures in the OS/390 Environment

Portable procedures are documented in the *SAS Procedures Guide*. Only the procedures that are specific to OS/390 or that have aspects specific to OS/390 are documented in this chapter.

BMDP

Calls any BMDP program to analyze data in a SAS data set

OS/390 specifics: all

Syntax

PROC BMDP <*options*>;
  VAR *variables*;
  BY *variables*;
  PARMCARDS;
  BMDP control statements;
  ;

Details

BMDP is a library of statistical analysis programs that were originally developed at the UCLA Health Sciences Computing Facility. Use the BMDP procedure in SAS programs to:

- □ call a BMDP program to analyze data in a SAS data set
- □ convert a SAS data set to a BMDP "save" file.

In order to use the BMDP procedure in a SAS session, the JCL EXEC statement must request the cataloged procedure SASBMDP rather than the usual cataloged procedure SAS. If the SASBMDP cataloged procedure is not available on your computer system, or if it has a different name, ask your computing center staff to help you set it up. Your SAS Installation Representative has the SAS System installation instructions, which include directions for setting up the procedure.

You can use BMDP programs to analyze SAS data sets by invoking this procedure. To analyze BMDP data with the SAS System, create a BMDP "save" file in a BMDP program, and then use the SAS CONVERT procedure or the BMDP engine to convert the "save" file to a SAS data set. (See "Introduction" on page 63 for more information about the BMDP engine.) You can use the BMDP procedure any number of times in a SAS job to invoke BMDP.

To use the BMDP procedure, first specify the name of the BMDP program you want to invoke in the PROC BMDP statement. The VAR and BY statements can follow, but they are optional. The BMDP control statements follow the PARMCARDS statement.

PROC BMDP Statement

PROC BMDP <*options*>;

The following options can be used in the PROC BMDP statement:

CODE=*save-file*
: assigns a name to the BMDP "save" file that the BMDP procedure creates from a SAS data set. The *save-file* corresponds to the CODE sentence in the BMDP INPUT paragraph. For example, you can use the following statement:

```
proc bmdp prog=bmdp3s code=judges;
```

Then, the BMDP INPUT paragraph must contain the following sentence:

```
CODE='JUDGES'
```

CODE= usually is not necessary in the PROC BMDP statement. When CODE= is not specified, the name of the BMDP "save" file is the SAS data set name.

If you are converting a SAS data set to a BMDP "save" file, include the CODE sentence in the BMDP INPUT paragraph to name the "save" file. To use the name of the SAS data set, specify that name in the BMDP INPUT paragraph. If you use a different name, it must match the name that is supplied in the CODE= option.

CONTENT=DATA | CORR | MEAN | FREQ
tells BMDP whether your SAS data set is a standard SAS data set (CONTENT=DATA) or whether it contains a correlation matrix (CORR), variable means (MEAN), or frequency counts (FREQ). You do not need to specify the CONTENT= option for specially structured SAS data sets that were created by other SAS procedures. If you omit the CONTENT= option, the data set's TYPE value is used.

Note: BMDP may use a structure for special data sets (for example, a correlation matrix) that is different from the SAS structure. Ensure that the input SAS data set is in the form that BMDP expects. △

DATA=*SAS-data-set*
specifies the SAS data set that you want the BMDP program to process. If you do not specify the DATA= option, PROC BMDP uses the most recently created SAS data set.

LABEL=*variable*
specifies a variable whose values are to be used as case labels for BMDP. Only the first four characters of the values are used. The LABEL= variable must also be included in the VAR statement if you use one.

LABEL2=*variable*
specifies a variable whose values are to be used as second case labels for BMDP. As with the LABEL= option, only the first four characters are used, and the LABEL2= variable must also be given in the VAR statement if you use one.

NOMISS
specifies that you want the BMDP program or "save" file to exclude observations that contain missing values.

PROG=BMDP*nn*
specifies the BMDP program that you want to run. For example, the following PROC BMDP statement runs the BMDP3S program:

```
proc bmdp prog=bmdp3s;
```

Note: If you want only to convert a SAS data set to a BMDP "save" file and do not want to run a BMDP program, omit the PROG= option and include the UNIT= option, which is described next. △

UNIT=*n*
specifies the FORTRAN logical unit number for the BMDP "save" file that the BMDP procedure creates. The value you specify for *n* must correspond to the UNIT= value that is specified in the INPUT paragraph of the BMDP control language.

If you omit this option, *n* defaults to 3 and FT03F001 is used as the fileref for the "save" file.

WRKSPCE=*nn* | PARM=*nn*
controls the allocation of a work space in BMDP. The WRKSPCE= or PARM= value is passed as a parameter to BMDP programs and corresponds to the WRKSPCE= feature in BMDP OS/390 cataloged procedures. The default value for *nn* is 30. If *nn* is less than 100, then its value represents kilobytes. If it is greater than 100, then its value represents bytes.

VAR Statement

VAR *variables*;

The VAR statement specifies which variables to use in the BMDP program. When you do not include a VAR statement, the BMDP program uses all the numeric variables in the SAS data set.

BY Statement

BY *variables*;

Use the BY statement with the BMDP procedure to obtain separate analyses of observations in groups. The groups are defined with the BY variables. When you use a BY statement, the procedure expects the input data set to be sorted in order of the BY variables or to have an appropriate index. If your input data set is not sorted in ascending order, you can do the following:

- use the SORT procedure with a similar BY statement to sort the data.
- if appropriate, use the BY statement options NOTSORTED or DESCENDING.
- create an index on the BY variables that you want to use. For more information about creating indexes and about using the BY statement with indexed data sets, see "The DATASETS Procedure" in the *SAS Procedures Guide*.

If a BY statement is used, it is included in the BMDP printed output to distinguish the BY group output.

For more information about the BY statement, see *SAS Language Reference: Dictionary*.

PARMCARDS Statement

PARMCARDS;

The PARMCARDS statement indicates that the BMDP control language follows.

BMDP Control Statements

Put your BMDP control language statements after the PARMCARDS statement. These statements are similar for all BMDP programs; see the most current BMDP manual for information about their forms and functions.

The BMDP INPUT paragraph must include UNIT and CODE sentences. The values of these sentences must match the UNIT= and CODE= values that are given in the PROC BMDP statement. (If the PROC BMDP statement does not specify a UNIT= value, then use 3 as the UNIT= value in the BMDP statements.) Use the SAS data set name as the CODE value unless you have used the CODE= option in the PROC BMDP statement to specify a different name. Omit the VARIABLES paragraph from the BMDP statements, because it is not needed when your input is a "save" file.

How Missing Values Are Handled

Before the BMDP procedure sends data to BMDP, it converts missing SAS values to the standard BMDP missing value. When you use the NOMISS option in the PROC BMDP statement, observations that contain missing values are excluded from the data set that is sent to the BMDP program.

Invoking BMDP Programs That Need FORTRAN Routines

Some BMDP programs, such as the programs for nonlinear regression, need to invoke the FORTRAN compiler and linkage editor before executing. All BMDP compilation and link editing must be completed before you use PROC BMDP.

Example of Creating and Converting a BMDP Save File

Here is an example of creating and converting a BMDP “save” file.

```
❶ data temp;
    input x y z;
    datalines;
    1 2 3
    4 5 6
    7 8 9
    10 11 12
   run;
❷ proc contents;
    title 'CONTENTS OF SAS DATA SET TO BE RUN
       THROUGH BMDP1D';
   run;

❸ proc bmdp prog=bmdp1d unit=3;
    parmcards;
    /input unit=3. code='TEMP'.
    /print min.
    /save unit=4. code='NEW'. NEW.
    /end
    /finish
   run;

❹ libname ft04f001 bmdp;
❺ data _null_;
    set ft04f001.new;
    put _all_;
   run;

❻ proc contents data=ft04f001._all_;
   run;

❼ proc convert bmdp=ft04f001 out=xyz;
```

The numbered lines of code are explained here:

1 This DATA step creates a SAS data set called TEMP.

2 The CONTENTS procedure shows the descriptive information for the data set TEMP.

3 PROC BMDP calls the BMDP program BMDP1D to analyze the data set TEMP.

   Note the BMDP program statements UNIT=3. and CODE='TEMP'. The results are stored in the BMDP "save" file, NEW.

   The word NEW must be in the SAVE paragraph. UNIT=*nn* should refer to the FT*nn*F001 fileref that was defined in your DD statement.

4 The LIBNAME statement associates the libref FT04F001 with the BMDP engine so that SAS knows which engine to use to access the data.

5 The DATA step reads the BMDP "save" file NEW, which was created in the previous PROC BMDP step. It uses the two-level name FT04F001.NEW to reference the file.

6 The CONTENTS procedure prints the information regarding all members that reside in the FT04F001 file. The _ALL_ member name is a special member name for the BMDP engine; it causes PROC CONTENTS to process all BMDP members in the file.

7 The CONVERT procedure converts the BMDP "save" file NEW to a SAS data set named XYZ. The NEW "save" file is on UNIT 4, that is, FT04F001.

The output from this SAS program is shown in Output 16.1 on page 228.

Output 16.1 NEW Save File Created from Data Set TEMP and Converted to SAS Data Set XYZ, Part 1 of 3

```
1
               CONTENTS OF SAS DATA SET TO BE RUN THROUGH BMDP1D
                                The CONTENTS Procedure
...

           -----Alphabetic List of Variables and Attributes-----

                    #    Variable    Type    Len    Pos
                    -----------------------------------
                    1    X           Num       8      0
                    2    Y           Num       8      8
                    3    Z           Num       8     16
...

PAGE   1  1D

BMDP1D - SIMPLE DATA DESCRIPTION
COPYRIGHT 1977, 1979, 1981, 1982, 1983, 1985, 1987, 1988, 1990
                 BY BMDP STATISTICAL SOFTWARE, INC.

       BMDP STATISTICAL SOFTWARE, INC.| BMDP STATISTICAL SOFTWARE
       1440 SEPULVEDA BLVD            | CORK TECHNOLOGY PARK, MODEL FARM RD
       LOS ANGELES, CA 90025 USA      | CORK, IRELAND
          PHONE (213) 479-7799        |    PHONE +353 21 542722
          FAX   (213) 312-0161        |    FAX   +353 21 542822
          TELEX 4972934 BMDP UI       |    TELEX 75659 SSWL EI

VERSION: 1990    (IBM/OS)          DATE: APRIL 27, 1999  AT 14:27:43
 MANUAL: BMDP MANUAL VOL. 1 AND VOL. 2.
 DIGEST: BMDP USER'S DIGEST.
UPDATES: STATE NEWS. IN THE PRINT PARAGRAPH FOR SUMMARY OF NEW FEATURES.

PROGRAM INSTRUCTIONS

/INPUT UNIT=3. CODE='TEMP'.
/PRINT MIN.
/SAVE UNIT=4. CODE='NEW'. NEW.
/END

PROBLEM TITLE IS
   APRIL 27, 1999      14:27:43

NUMBER OF VARIABLES TO READ . . . . . . . . . .       3
NUMBER OF VARIABLES ADDED BY TRANSFORMATIONS. .       0
TOTAL NUMBER OF VARIABLES . . . . . . . . . . .       3
CASE FREQUENCY VARIABLE . . . . . . . . . . . .
CASE WEIGHT VARIABLE. . . . . . . . . . . . . .
CASE LABELING VARIABLES . . . . . . . . . . . .
NUMBER OF CASES TO READ . . . . . . . . . . . . TO END
MISSING VALUES CHECKED BEFORE OR AFTER TRANS. . NEITHER
BLANKS IN THE DATA ARE TREATED AS   . . . . . . MISSING
INPUT UNIT NUMBER . . . . . . . . . . . . . . .       3
REWIND INPUT UNIT PRIOR TO READING. . DATA. . .     YES
NUMBER OF INTEGER WORDS OF MEMORY FOR STORAGE .  689662

INPUT BMDP FILE
CODE. . . IS     TEMP
CONTENT . IS     DATA
LABEL . . IS

VARIABLES
     1 X           2 Y           3 Z

VARIABLES TO BE USED
     1 X           2 Y           3 Z
  PRINT CASES CONTAINING VALUES LESS THAN THE STATED MINIMA.

------------------------------------------
BMDP FILE IS BEING WRITTEN ON UNIT       4
CODE. . . IS     NEW
CONTENT . IS     DATA
LABEL . . IS      APRIL 27, 1999         14:27:43
```

Output 16.2 NEW Save File Created from Data Set TEMP and Converted to SAS Data Set XYZ, Part 2 of 3

```
PAGE   2  1D  APRIL 27, 1999         14:27:43
VARIABLES ARE
     1 X           2 Y           3 Z

BMDP    FILE ON UNIT  4 HAS BEEN COMPLETED.
------------------------------------------
NUMBER OF CASES WRITTEN TO FILE        4

NUMBER OF CASES READ. . . . . . . . . . . . . .       4

 VARIABLE      TOTAL                STANDARD    ST.ERR   COEFF. OF   SMALLEST
 NO. NAME      FREQUENCY      MEAN  DEVIATION   OF MEAN  VARIATION   VALUE
             LARGEST
  Z-SCORE    VALUE   Z-SCORE    RANGE

   1 X            4         5.5000    3.8730    1.9365    .70418    1.0000
   -1.16     10.000    1.16     9.0000
   2 Y            4         6.5000    3.8730    1.9365    .59584    2.0000
   -1.16     11.000    1.16     9.0000
   3 Z            4         7.5000    3.8730    1.9365    .51640    3.0000
   -1.16     12.000    1.16     9.0000

NUMBER OF INTEGER WORDS USED IN PRECEDING    PROBLEM     530
CPU TIME USED      0.030 SECONDS
```

Output 16.3 NEW Save File Created from Data Set TEMP and Converted to SAS Data Set XYZ, Part 3 of 3

```
PAGE   3  1D

BMDP1D - SIMPLE DATA DESCRIPTION
COPYRIGHT 1977, 1979, 1981, 1982, 1983, 1985, 1987, 1988, 1990
              BY BMDP STATISTICAL SOFTWARE, INC.

       BMDP STATISTICAL SOFTWARE, INC.| BMDP STATISTICAL SOFTWARE
       1440 SEPULVEDA BLVD            | CORK TECHNOLOGY PARK, MODEL FARM RD
       LOS ANGELES, CA 90025 USA      | CORK, IRELAND
          PHONE (213) 479-7799        |    PHONE +353 21 542722
          FAX   (213) 312-0161        |    FAX   +353 21 542822
          TELEX 4972934 BMDP UI       |    TELEX 75659 SSWL EI

VERSION: 1990    (IBM/OS)          DATE: APRIL 27, 1999  AT 14:27:44

PROGRAM INSTRUCTIONS

/FINISH

NO MORE CONTROL LANGUAGE.

PROGRAM TERMINATED

              CONTENTS OF SAS DATA SET TO BE RUN THROUGH BDMP1D

                             The CONTENTS Procedure

Data Set Name: FT04F001.NEW                        Observations:         .
Member Type:   DATA                                Variables:            4
Engine:        BMDP                                Indexes:              0
Created:       14:27 Monday, April 27, 1999        Observation Length:   16
Last Modified: 14:27 Monday, April 27, 1999        Deleted Observations: 0
Protection:                                        Compressed:           NO
Data Set Type:                                     Sorted:               NO
Label:


           -----Alphabetic List of Variables and Attributes-----

                #    Variable    Type    Len    Pos
                -----------------------------------
                4    USE         Num       4     12
                1    X           Num       4      0
                2    Y           Num       4      4
                3    Z           Num       4      8

                            -----Directory-----
            Libref:        FT04F001
            Engine:        BDMP
            Physical Name: SYS99117.T145548.RA000.BDMP90V8.R0120689

                         #  Name  Memtype
                         ----------------
                         1  NEW   DATA
```

CATALOG

Manages SAS catalogs

OS/390 specifics: FILE= option

Details

The FILE= option in the CONTENTS statement of the CATALOG procedure is the only portion of this procedure that is host specific. Under OS/390, if the value that you specify in the FILE= option has not been previously defined as a fileref (using a FILENAME statement, FILENAME function, TSO ALLOCATE command, or JCL DD statement), then SAS uses the value to construct the physical file name.

In the following example, if the SAS system option FILEPROMPT is in effect, a requestor window asks whether you want to allocate the external file whose fileref is SAMPLE. If you reply **Yes**, then SAS attempts to locate the external file. If the file was not previously allocated, then SAS allocates it. To construct the data set name, SAS inserts the value of the SYSPREF= system option in front of the FILE= value (in this case, SAMPLE), and it appends the name LIST to it. In this example, if the value of SYSPREF= is SASDEMO.V8, then SAS allocates an physical file named SASDEMO.V8.SAMPLE.LIST.

```
proc catalog catalog=profile;
   contents file=sample;
run;
```

See Also

- *SAS Procedures Guide*

CIMPORT

Restores a transport file that was created by the CPORT procedure

OS/390 specifics: options

Details

The DISK option is the default for the CIMPORT procedure. Therefore, PROC CIMPORT defaults to reading from a file on disk instead of from a tape. If you want to read a file from tape, then specify the TAPE option.

When writing and reading files to and from tapes, you are not required to specify the DCB attributes in a SAS FILENAME statement or FILENAME function. However, it is recommended that you specify BLKSIZE=8000.

See Also

- *Moving and Accessing SAS Files across Operating Environments*
- “CPORT” on page 240
- *SAS Procedures Guide*

CONTENTS

Prints descriptions of the contents of one or more files from a SAS data library

OS/390 specifics: engine/host-dependent information, directory information

Syntax

PROC CONTENTS *<option(s)>*;

Details

Although most of the output that this procedure generates is the same on all operating environments, the Engine/Host Dependent Information is system-dependent and engine-dependent. Output 16.4 on page 233 shows sample PROC CONTENTS output, including the information that is specific to OS/390 for the BASE engine.

Output 16.4 CONTENTS Procedure Output, Including Engine/Host Dependent Information

```
                             CONTENTS PROCEDURE
 Data Set Name: WORK.ORANGES                        Observations:         4
 Member Type:   DATA                                Variables:            5
 Engine:        V8                                  Indexes:              0
 Created:       15:56 Monday, April 27, 1999        Observation Length:   40
 Last Modified: 15:56 Monday, April 27, 1999        Deleted Observations: 0
 Protection:                                        Compressed:           NO
 Data Set Type:                                     Sorted:               YES
 Label:

                -----Engine/Host Dependent Information-----

       Data Set Page Size:         6144
       Number of Data Set Pages:   1
       First Data Page:            1
       Max Obs per Page:           152
       Obs in First Data Page:     4
       Number of Data Set Repairs: 0
       Physical Name:              SYS96050.T153830.RA000.USERID.R0000004
       Release Created:            8.0000B1
       Release Last Modified:      8.0000B1
       Created by:                 USERID
       Last Modified by:           USERID
       Subextents:                 1
       Total Blocks Used:          1
                       Taste Test Results For Oranges

                             CONTENTS PROCEDURE

           -----Alphabetic List of Variables and Attributes-----

                    #    Variable    Type    Len    Pos
                    -----------------------------------
                    2    FLAVOR      Num       8      8
                    4    LOOKS       Num       8     24
                    3    TEXTURE     Num       8     16
                    5    TOTAL       Num       8     32
                    1    VARIETY     Char      8      0
```

The procedure output provides values for the physical characteristics of the SAS data set WORK.ORANGES. Important values follow:

Observations
: is the number of nondeleted records in the data set.

Observation Length
: is the maximum record size in bytes.

Compressed
: has the value NO if records are not compressed; it has the value CHAR or BINARY if records are compressed.

Data Set Page Size
: is the size of pages in the data set.

Number of Data Set Pages
: is the total number of pages in the data set.

First Data Page
: is the number of the page that contains the first data record; header records are stored in front of data records.

Max Obs per Page
: is the maximum number of records a page can hold.

Obs in First Data Page
: is the number of data records in the first data page.

The DIRECTORY option lists several host-specific data library attributes at the beginning of PROC CONTENTS output. Output 16.5 on page 234 shows the directory information that is listed by the following code:

```
proc contents data=test._all_ directory;
run;
```

Output 16.5 Engine/Host Dependent Information

```
                         CONTENTS PROCEDURE
                        -----Directory-----

         Libref:                  TEST
         Engine:                  V8
         Physical Name:           USERID.TEST.TESTLIB
         Unit:                    DISK
         Volume:                  TST810
         Disposition:             OLD
         Device:                  3380
         Blocksize:               6144
         Blocks per Track:        7
         Total Library Blocks:    105
         Total Used Blocks:       28
         Total Free Blocks:       77
         Highest Used Block:      28
         Highest Formatted Block: 35
         Members:                 2

                             #  Name  Memtype  Indexes
                             _________________________
                             1  TEMP  DATA
                             2  XYZ   DATA
```

The following list explains these data library attributes:

Total Library Blocks
: is the total number of blocks that are currently allocated to the data library. This value equals the sum of Total Used Blocks and Total Free Blocks. It also equals Blocks per Track multiplied by the number of tracks that are currently allocated to

the data library. The current number of allocated cylinders or tracks can be found in the DSINFO window, or in ISPF panel 3.2. These windows show what the allocation was the last time the data library was closed.

Total Used Blocks
: is the total number of library blocks that currently contain valid data. It equals the sum of the directory blocks and all the data blocks that are associated with existing members.

Total Free Blocks
: is the total number of currently allocated library blocks that are available for use by members or as extra directory blocks. This count includes any data blocks that were previously associated with members that have been deleted.

Highest Used Block
: is the number of the highest relative block in the data library that currently contains either directory information or data for an existing member.

Highest Formatted Block
: is the number of the highest relative block in the data library that has been internally formatted for use. Blocks are internally formatted before they are used, and they are formatted in full track increments. Therefore, the highest formatted block is equal to the Blocks per Track multiplied by the number of tracks that are currently used by the data library. The number of currently used cylinders or tracks can be found in the DSINFO window or in ISPF panel 3.2. These windows show what the allocation was the last time the data library was closed. This number is also the true End Of File marker. It corresponds to the DS1LSTAR field in the DSCB in the VTOC, which is the OS/390 operating environment's EOF flag.

Note: The same directory information that is generated by the DIRECTORY option in the PROC CONTENTS statement is also generated by the LIST option in the LIBNAME statement. △

See Also

- *SAS Language Reference: Dictionary*
- *SAS Procedures Guide*

CONVERT

Converts BMDP, OSIRIS, and SPSS system files to SAS data sets

OS/390 specifics: all

Syntax

PROC CONVERT <*options*>;

Details

PROC CONVERT produces one output SAS data set but no printed output. The new SAS data set contains the same information as the input system file; exceptions are noted in "How Variable Names Are Assigned" on page 237.

The procedure converts system files from these software packages:

- BMDP "save" files up to and including the most recent version of BMDP
- SPSS "save" files up to and including Release 9, along with SPSS-X and the SPSS Portable File Format
- OSIRIS files up to and including OSIRIS IV (hierarchical file structures are not supported).

These software packages are products of other organizations. Therefore, changes may be made that make the system files incompatible with the current version of PROC CONVERT. SAS Institute cannot be responsible for upgrading PROC CONVERT to support changes to other vendor's software packages; however, attempts to do so are made when necessary with each new version of SAS.

Information associated with each software package is given in "Introduction" on page 63 .

PROC CONVERT Statement

PROC CONVERT <*options*>;

options can be from the following list. Only one of the options that specify a system file (BMDP, OSIRIS, or SPSS) can be included. Usually only the PROC CONVERT statement is used, although data set attributes can be controlled by specifying the DROP=, KEEP=, or RENAME= data set options with the OUT= option of this procedure. See *SAS Language Reference: Dictionary* for more information about these data set options. You can also use LABEL and FORMAT statements following the PROC statement.

BMDP=*fileref* <(CODE=*code-id* | CONTENT= *content-type*)>
: specifies the fileref of a BMDP "save" file. The first "save" file in the physical file is converted. If you have more than one "save" file in the data set, then you can use two additional options in parentheses after the libref or fileref. The CODE= option lets you specify the code of the "save" file you want, and the CONTENT= option lets you give the "save" file's content. For example, if a file CODE=JUDGES has a CONTENT of DATA, you can use this statement:

```
proc convert bmdp=bmdpfile(code=judges
   content=data);
```

DICT=*fileref*
: specifies the fileref of an physical file that contains the dictionary file for the OSIRIS data set. The DICT= option is required if you use the OSIRIS= option.

FIRSTOBS=*n*
: gives the number of the observation at which the conversion is to begin. This enables you to skip over observations at the beginning of the BMDP, OSIRIS, or SPSS file.

OBS=*n*
: specifies the number of the last observation to be converted. This enables you to exclude observations at the end of the file.

OSIRIS=*fileref*
: specifies a fileref for an physical file that contains an OSIRIS file. The DICT= option is required when you use the OSIRIS= option.

OUT=*SAS-data-set*
: names the SAS data set that will be created to hold the converted data. If OUT= is omitted, SAS still creates a data set and automatically names it DATA*n*, just as if you omitted a data set name in a DATA statement. That is, if it is the first such

data set in a job or session, then SAS names it DATA1; the second is DATA2, and so on. If you omit the OUT= option, or if you do not specify a two-level name in the OUT= option, then the converted data set is not permanently saved.

SPSS=*fileref*
: specifies a fileref for an physical file that contains an SPSS file. The SPSS file can be in any of three formats: SPSS Release 9 (or prior), SPSS-X format (whose originating operating environment is OS/390, CMS, or VSE), or the portable file format from any operating environment.

How Missing Values Are Handled

If a numeric variable in the input data set has no value or has a system missing value, PROC CONVERT assigns a missing value to it.

How Variable Names Are Assigned

The following sections explain how names are assigned to the SAS variables that are created by the CONVERT procedure.

CAUTION:
Because some translation of variable names can occur (as indicated in the following sections), ensure that the translated names will be unique. △

Variable Names in BMDP Output Variable names from the BMDP "save" file are used in the SAS data set, except that nontrailing blanks and all special characters are converted to underscores in the SAS variable names. The subscript in BMDP variable names, such as x(1), becomes part of the SAS variable name, with the parentheses omitted: X1. Alphabetic BMDP variables become SAS character variables of length 4. Category records from BMDP are not accepted.

Variable Names in OSIRIS Output For single-response variables, the V1 through V9999 name becomes the SAS variable name. For multiple-response variables, the suffix R*n* is added to the variable name, when *n* is the response. For example, V25R1 would be the first response of the multiple response V25. If the variable after or including V1000 has 100 or more responses, then responses above 99 are eliminated. Numeric variables that OSIRIS stores in character, fixed-point binary, or floating-point binary mode become SAS numeric variables. Alphabetic variables become SAS character variables; any alphabetic variable whose length is greater than 200 is truncated to 200. The OSIRIS variable description becomes a SAS variable label, and OSIRIS print format information is translated to the appropriate SAS format specification.

Variable Names in SPSS Output SPSS variable names and labels become variable names and labels without change. SPSS alphabetic variables become SAS character variables of length 4. SPSS blank values are converted to SAS missing values. SPSS print formats become SAS formats, and the SPSS default precision of no decimal places becomes part of the variables' formats. The SPSS DOCUMENT data is copied so that the CONTENTS procedure can display them. SPSS value labels are not copied.

Example of Converting a BMDP Save File

The following statements convert a BMDP "save" file and produce the temporary SAS data set TEMP, which contains the converted data. The PROC CONTENTS output would be similar to that shown in Output 16.1 on page 228.

```
filename ft04f001 'userid.bmdp.savefile';
proc convert bmdp=ft04f001 out=temp;
```

```
run;

title 'BMDP CONVERT Example';

proc contents;
run;
```

Example of Converting an OSIRIS File

The following statements convert an OSIRIS file and produce the temporary SAS data set TEMP, which contains the converted data. Output 16.6 on page 238 shows the attributes of TEMP.

```
filename osiris 'userid.misc.cntl(osirdata)';
filename dict 'userid.misc.cntl(osirdict)';
proc convert osiris=osiris dict=dict out=temp;
run;

title 'OSIRIS CONVERT Example';

proc contents;
run;
```

Output 16.6 Converting an OSIRIS File

```
                              OSIRIS CONVERT Example
                                The CONTENTS Procedure

 Data Set Name: WORK.TEMP                           Observations:         20
 Member Type:   DATA                                Variables:            9
 Engine:        V8                                  Indexes:              0
 Created:       9:46 Monday, April 27, 1999         Observation Length:   36
 Last Modified: 9:46 Monday, April 27, 1999         Deleted Observations: 0
 Protection:                                        Compressed:           NO
 Data Set Type:                                     Sorted:               NO
 Label:

                 -----Engine/Host Dependent Information-----

       Data Set Page Size:         6144
       Number of Data Set Pages:   1
       First Data Page:            1
       Max Obs per Page:           135
       Obs in First Data Page:     20
       Number of Data Set Repairs: 0
       Physical Name:              SYS99117.T152416.RA000.USERID.R0121907
       Release Created:            8.0000B2
       Release Last Modified:      8.0000B2
       Created by:                 USERID
       Last Modified by:           USERID
       Subextents:                 1
       Total Blocks Used:          1

             -----Alphabetic List of Variables and Attributes-----

  #  Variable  Type  Len  Pos  Format  Label
  --------------------------------------------------------------------------
  1  V1        Num     4    0          INTERVIEW NUMBER         REF=   1 ID=
  2  V2        Num     4    4          INTERVIEWER NUMBER       REF=   2 ID=
  3  V3        Num     4    8          PRIMARY SAMPLING UNIT    REF=   3 ID=
  4  V4        Num     4   12          REGION                   REF=   4 ID=
  5  V5        Num     4   16          CHUNK AND SEGMENT        REF=   5 ID=
  6  V6        Num     4   20          LANGUAGE OF INTERVIEW    REF=   6 ID=
  7  V7        Num     4   24          LANGUAGE OF INTERVIEW    REF=1621 ID=
  8  V8        Num     4   28          LNGTH OF INTERVIEW       REF=1620 ID=
  9  V9        Num     4   32  12.4    WEIGHT                   REF=1700 ID=
```

Example of Converting an SPSS File

The following statements convert an SPSS Release 9 file and produce the temporary SAS data set TEMP, which contains the converted data. The output generated by PROC CONTENTS is similar in format to Output 16.6 on page 238.

```
filename spss 'userid.spssfile.num1';
proc convert spss=spss out=temp;
run;

title 'SPSSR9 CONVERT Example';

proc contents;
run;
```

See Also

- “Introduction” on page 63

CPORT

Writes SAS data sets and catalogs into a special format in a transport file

OS/390 specifics: specification of transport file

Details

The DISK option is the default for the CPORT procedure; therefore, CPORT defaults to writing to a file on disk instead of on a tape. If you want to write to a file on tape, specify the TAPE option or assign the fileref or DDname of SASCAT to a tape.

You are not required to define the logical name SASCAT to your tape, and you are not required to specify the full DCB attributes. However, the BLKSIZE= value must be an integral multiple of 80; a value of 8000 is recommended.

Here is an example of exporting all the SAS data sets and catalogs in a SAS data library to a transport file on disk. Note that the FILENAME statement specifies BLKSIZE=8000.

```
libname oldlib 'SAS-data-library';
filename tranfile 'transport-file-name'
   blksize=8000 disp=(new,catlg);
proc cport library=oldlib file=tranfile;
run;
```

PROC CPORT writes a transport file to the physical file that is referenced by TRANFILE. The file contains all the data sets and catalogs in the SAS data library OLDLIB.

See Also

- *Moving and Accessing SAS Files across Operating Environments*
- “CIMPORT” on page 232
- *SAS Procedures Guide*

DATASETS

Lists, copies, renames, repairs, and deletes SAS files; manages indexes for and appends SAS data sets in a SAS data library; changes variable names and related information; prints the contents of a SAS data library

OS/390 specifics: output generated by CONTENTS statement, data library information

Details

Part of the DATASETS procedure output is system-dependent. The SAS data library information that is displayed in the SAS log depends on the operating environment and the engine. In Output 16.7 on page 241, the SAS log shows the information that is generated by the DATASETS procedure for the V8 (BASE) engine under OS/390.

Note: The information produced for other engines varies slightly. See "Using V8 Engines" on page 55 for information about other engines. △

Output 16.7 SAS Data Library Information from the DATASETS Procedure

```
                         -----Directory-----
       Libref:                 WORK
       Engine:                 V8
       Physical Name:          SYS96053.T145204.RA000.USERID.R0000128
       Unit:                   DISK
       Volume:                 ANYVOL
       Disposition:            NEW
       Device:                 3380
       Blocksize:              6144
       Blocks per Track:       7
       Total Library Blocks:   105
       Total Used Blocks:      31
       Total Free Blocks:      74
       Highest Used Block:     44
       Highest Formatted Block: 49
       Members:                1

                       #  Name     Memtype  Indexes
                       ----------------------------
                       1  ORANGES  DATA
                       2  PROFILE  CATALOG
```

For explanations of the fields in Output 16.7 on page 241, see "CONTENTS" on page 232.

See Also

- "CONTENTS" on page 232
- *SAS Language Reference: Dictionary*
- *SAS Procedures Guide*

DBF

Converts a dBASE file to a SAS data set or a SAS data set to a dBASE file

OS/390 specifics: all

Syntax

PROC DBF *options* ;

options

DB2 | DB3 | DB4 | DB5=*fileref*

specifies the fileref of a DBF file. The fileref may be allocated via a SAS FILENAME statement, a JCL DD statement (in batch mode), or a TSO ALLOC command (under TSO). For further information on fileref specification, see “Ways of Allocating External Files” on page 69 . The DBF file can be stored as a sequential data set (such as sasdemo.emp.dbf), a partitioned OS/390 data set member (such as sasdemo.dbf.pds(emp)), or a file in an hierarchical file system (such as /u/sasdemo/emp.dbf). For further information on file naming requirements, see “Referring to External Files” on page 78 .

If the fileref is allocated with a FILENAME statement, the statement may optionally specify RECFM=N to identify the DBF file as binary.

The DB*n* option must correspond to the version of dBASE with which the DBF file is compatible. Specify a DBF file with the DB*n* option, where *n* is 2, 3, 4, or 5. You can specify only one of these values.

DATA=<*libref.*>*member*

names the input SAS data set, using 1–32 characters. Use this option if you are creating a DBF file from a SAS data set. If you use the DATA= option, do not use the OUT= option. If you omit the DATA= option, SAS creates an output SAS data set from the DBF file.

OUT=<*libref.*>*member*

names the SAS data set that is created to hold the converted data, using 1–32 characters. Use this option only if you do not specify the DATA= option. If OUT= is omitted, SAS creates a temporary data set in the WORK library. The name of the temporary data set is DATA1 [...DATA*n*]. If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the SAS data set that is created by PROC DBF remains available during your current SAS session (under the temporary data set name), but it is not permanently saved.

Details

You can use PROC DBF in the OS/390 environment if your site has a license for SAS/ACCESS for PC File Formats. To see a list of your licences, submit:

```
proc setinit; run;
```

If you are licensed, you will see an entry in your SAS log for SAS/ACCESS for PC File Formats.

The DBF procedure converts files in DBF format to SAS data sets that are compatible with the current SAS release. You can also use PROC DBF to convert SAS data sets to files in DBF format.

Before you convert a DBF file to a SAS file, you must first upload your DBF file from the Windows, OS/2, NT, or UNIX environment to the OS/390 environment, using a mechanism such as FTP (file transfer protocol). If you are licensed for SAS/CONNECT, you can use PROC UPLOAD:

```
filename out1 'sasdemo.emp.dbf';
proc upload infile='c:\employee\emp.dbf'
   outfile=out1 binary;
run;
```

In the OS/390 environment, sequential data sets are recommended for use with DBF, with the following attributes:

RECFM=FS

DSORG=PS

LRECL=6160

BLKSIZE=6160

The following example illustrates the specification of attributes for a sequential data set:

```
sasdemo.emp.dbf =
   (DSORG=PS,RECFM=FS,LRECL=6160,BLKSIZE=6160)
```

PROC DBF produces one output file but no printed output. The output file contains the same information as the input file but in a different format.

The DBF procedure works with DBF files created by all the current versions and releases of dBASE (II, III, III PLUS, IV, and 5.0) and with most DBF files that are created by other software products.

Converting DBF Fields to SAS Variables When you convert a DBF file a to SAS data set, DBF numeric variables become SAS numeric variables. Similarly, DBF character variables become SAS character variables. Any DBF character variable of length greater than 200 is truncated to 200 in SAS. Logical fields become SAS character variables with a length of 1. Date fields become SAS date variables.

DBF fields whose data are stored in auxiliary files (Memo, General, binary, and OLE data types) are ignored in SAS.

If a DBF file has missing numeric or date fields, SAS fills those missing fields with a series of the digit 9 or with blanks, respectively.

When a dBASE II file is translated into a SAS data set, any colons in dBASE variable names are changed to underscores in SAS variable names. Conversely, when a SAS data set is translated into a dBASE file, any underscores in SAS variable names are changed to colons in dBASE field names.

Converting SAS Variables to DBF Fields In DBF files, numeric variables are stored in character form. When converting from a SAS data set to a DBF file, SAS numeric variables become DBF numeric variables with a total length of 16. A SAS numeric variable with a decimal value must be stored in a decimal format in order to be converted to a DBF numeric field with a decimal value. In other words, unless you associate the SAS numeric variable with an appropriate format in a SAS FORMAT statement, the corresponding DBF field will not have any value to the right of the decimal point. You can associate a format with the variable in a SAS data set when you create the data set or by using the DATASETS procedure (see “DATASETS” on page 240).

If the number of digits—including a possible decimal point—exceeds 16, a warning message is issued and the DBF numeric field is filled with a series of the digit 9. All SAS character variables become DBF fields of the same length. When converting from a SAS data set to a DBF file that is compatible with dBASE III or later, SAS date variables become DBF date fields. When converting to a dBASE II file, SAS date variables become dBASE II character fields in the form YYYYMMDD.

Transferring Other Software Files to DBF Files You might find it helpful to save another software vendor’s file to a DBF file and then convert that file into a SAS data set. For example, you could save an Excel XLS file in DBF format, upload the file, and use PROC DBF to convert that file into a SAS data set. Or you could do the reverse; use PROC DBF to convert a SAS data set into a DBF file and then load that file into an Excel spreadsheet.

Example 1: Converting a dBASE IV File to a SAS Data Set

In this example, a dBASE IV file named SASDEMO.EMPLOYEE.DBF is converted to a SAS data set. A FILENAME statement specifies a fileref that names the dBASE IV file. The FILENAME statement must appear before the PROC DBF statement, as shown:

```
libname save 'sasdemo.employee.data';
filename dbfin 'sasdemo.employee.dbf';
proc dbf db4=dbfin out=save.employee;
run;
```

Example 2: Converting a dBASE 5 file to a SAS Data Set

In this example, a dBASE 5 file is converted to a SAS data set.

```
libname demo 'sasdemo.employee.data';
filename dbfout 'sasdemo.newemp.dbf' recfm=n;
proc dbf db5=dbfout data=demo.employee;
run;
```

FORMAT

Creates user-defined formats and informats

OS/390 specifics: LIBRARY= option in the PROC FORMAT statement

Details

To create a new format, specify a valid libref as the value of the LIBREF= option. This creates a new format in the Version 8 style in the FORMATS catalog. The FORMATS catalog is stored in the Version 8 SAS data library that is identified by the LIBRARY= option.

In SAS Version 8, you can no longer write Version 5 formats to a load library by using a DDname as the value of the LIBRARY= option. You can read Version 5 formats, but you cannot write them.

See Also

- *SAS Procedures Guide*

ITEMS

Builds, reads, and writes SAS itemstores

OS/390 specifics: all

Syntax

PROC ITEMS NAME=<*libref.*>*member*;

NAME

If no libref is specified, the libref is assumed to be WORK. If *libref.member* is specified, the libref must have been previously allocated. See “LIBNAME” on page 313 for details on allocation.

Details

An *itemstore* is a SAS data set that is made up of independently accessible chunks of information. SAS uses itemstores for online help, where the SAS help browser accesses an itemstore in the SASHELP library. You can use the ITEMS procedure to create, modify, and browse your own itemstores, which you can then access through the SAS help browser.

The contents of an itemstore are divided into directories, subdirectories, and topics. The directory tree structure emulates that of UNIX System Services, so that a given help topic is identified by a directory path (root_dir/sub_dir/item). This hierarchical structure allows the SAS help browser to supports HTML links between help topics.

The itemstores that SAS uses for HTML help can be written only by users with appropriate privilege. Though SAS Institute discourages rewrites of SAS help items, you can add items to the SAS help itemstores, and you can develop new itemstores of your own for any information that you wish to make available through the SAS help browser. For further information on writing your own HTML help, see “Using User-Defined Help” on page 27.

To access an itemstore, you must first allocate the library that contains the itemstore, unless the itemstore is a member of the WORK library. After you allocate the library, you issue the PROC ITEMS NAME=*fileref* statement to access the itemstore in SAS. Once the itemstore is available in SAS, you can use the LIST, IMPORT, EXPORT, MERGE, and DELETE statements to control itemstore contents. SAS applies all of these statements to the itemstore name in the last PROC ITEMS NAME= statement.

For information on the HTML tags that are supported by the SAS help browser, see “Using User-Defined Help” on page 27 .

HTC File Format Itemstores are physically stored in the operating environment as HTC files. HTC files can be imported and exported in HTC file format. Within the HTC file, individual help files are identified by a line that begins with five colons and ends with the filename:

```
:::::<filename>.htm
```

Directories in the HTC file are identified by a line that begins with file colons and ends with a path specification:

```
:::::<dirname1>/<dirname2>/<filename>.htm
```

If the HTC file containing the previous entry was imported into an itemstore with the IMPORT statement, the directory and subdirectory would be created as needed, and the file would be placed in the specified subdirectory. Any filename that lacks a path specification goes into the root directory or into the directory specified by the DIR= option, if it is specified in the IMPORT statement.

Alternate Syntax for the DIR= and ITEM= Statements You can use the forward slash path character (/) to specify a path in the DIR= and ITEM= options (described below) in all of the statements that take those options. For example, the following two statements are equivalent:

```
LIST DIR='usr' ITEM='mail';
LIST ITEM='usr/mail';
```

Note that a full path, starting with the directory just beneath the itemstore's root directory (with no initial forward slash) is required for access to anything except items in the root directory or to itemstores consisting of a single item.

Wildcards, using asterisks (*) as in UNIX, are not accepted in itemstore paths. Nor can you specify more than one path (a file concatenation) for each of the following statements.

LIST Statement

LIST <*options*;>

The LIST statement writes a list of item and/or directory names to the SAS log or to a specified file. Specifying no options writes a list of all items and directories to the SAS log.

Options

DUMP=*fileref*
: specifies the fileref that will receive the listing. If DUMP= is not specified, the output goes to the SAS log.

DIR='*dir-name*'
: specifies an itemstore directory whose item names you wish to list. If you specify the DIR= option alone, you will receive a listing of item names contained in that directory.

ITEM='*item-name*'
: specifies that you wish to list the contents of the named item in the named directory of the itemstore. If you specify an item without specifying a directory, you will receive the contents of the item with the specified name in the root directory of the itemstore.

IMPORT Statement The IMPORT statement imports a fileref into an itemstore. If the imported fileref contains items or directories that currently exist in the itemstore, the new items or directories overwrite (replace) the existing versions.

IMPORT FILEREF=*fileref* <*options*>;

Options

DIR='*dir-name*'
: specifies the itemstore directory that will receive the imported fileref. If a directory is not specified, the fileref is imported into the root directory of the itemstore.

ITEM='*item-name*'
: specifies the name of the item that will receive the imported fileref. If an item is not specified, the imported fileref is assumed to be an HTC file.

EXPORT Statement The EXPORT statement copies an item or itemstore to an external fileref in HTC format. If the fileref exists prior to the EXPORT statement, the new fileref overwrites (replaces) the previous version.

EXPORT FILEREF=*fileref*<*options*>;

Options

DIR='*dir-name*'
: specifies the itemstore directory that is the source of the export. If you do not specify a directory, the fileref receives the contents of the entire itemstore or the specified item from the root directory of the itemstore.

ITEM='*item-name*'
specifies an item for export. If you do not specify an item, the fileref receives the entire contents of the specified directory or itemstore, in HTC format.

MERGE Statement The MERGE statement merges the specified itemstore into the itemstore opened previously with PROC ITEMS.

MERGE SOURCE=<*libref.*>*member*;

A libref is required in the MERGE statement. If the two itemstores have directories with the same name and path, the contents of the new directory replace the contents of the old directory. If you merge into the root directory, the entire itemstore is replaced. If you merge a new item into a directory, the new item is merged into the old directory. If the old directory contains an item of the same name, the new item replaces the old item.

DELETE Statement The DELETE statement deletes all or part of the contents in an itemstore.

DELETE <*options*>;

Options

DIR='*dir-name*'
specifies the directory from which you wish to delete. When DIR= is not specified, either the entire contents of the itemstore are deleted or the specified item is deleted from the itemstore's root directory.

ITEM='*item-name*'
deletes the specified item from the specified directory in the itemstore, or if a directory is not specified, from the root directory of the itemstore.

See Also

- Information on the HELPLOC= system option in *SAS Language Reference: Dictionary*.

OPTIONS

Lists the current values of all system options in the SAS log

OS/390 specifics: host options displayed, host-specific options of OPTIONS statement

Details

Portable options are the same in all operating environments. To see a list of these options, submit:

```
proc options portable;
run;
```

Certain portable options have aspects that are specific to OS/390. All portable options with OS/390–specific aspects are documented in “Summary Table of SAS System Options” on page 419.

Other options are entirely specific to the OS/390 environment. To see a list of these options, submit:

```
proc options host;
run;
```

All options that are specific to OS/390 are documented in “Summary Table of SAS System Options” on page 419.

The following options cause the OPTIONS procedure to list the system options that are specific to the following SAS software products or applications. While the OPTIONS procedure still accepts the following one-word options, it is recommended that you use the associated GROUP= option instead:

ADB GROUP=ADABAS	SAS/ACCESS interface to ADABAS.
APF GROUP=INSTALL	system administrators
DB2 GROUP=DB2	SAS/ACCESS interface to DB2
DDB GROUP=DATACOM	SAS/ACCESS interface to CA-DATACOM/DB
IDM GROUP=IDMS	SAS/ACCESS interface to CA-IDMS
IMS GROUP=IMS	SAS/ACCESS interface to IMS-DL/I
ISP GROUP=ISPF	SAS interface to ISPF (see “SAS Interface to ISPF” on page 120).

For more information about SAS system options that are associated with a particular SAS/ACCESS interface, see the documentation for that SAS/ACCESS interface.

See Also

- □ “Displaying System Option Settings” on page 12
- □ “Summary Table of SAS System Options” on page 419
- □ *SAS Language Reference: Dictionary*
- □ *SAS Procedures Guide*

PDS

Lists, deletes, or renames members of a partitioned data set

OS/390 specifics: all

Syntax

PROC PDS DDNAME=*file-specification* <*options*>;

  DELETE *member-1* <. . . *member-n* >;

  CHANGE *old-name-1* =*new-name-1* < . . . *old-name-n* =*new-name-n* > ;

EXCHANGE *name-1=other-name-1* < . . . *name-n =other-name-n* > ;

Details

Partitioned data sets (PDS) are libraries that contain files called members. There are two kinds of partitioned data sets. One can contain source code, macros, cataloged procedures, and other data. The other, called a load library, can contain only load modules.

PROC PDS operates on the directory of a partitioned data set to list, delete, and rename members and aliases. (Partitioned data sets are not the same as SAS data libraries.) When members are deleted or renamed, PROC PDS updates the directory of the partitioned data set. Also, unless NOLIST is specified, PROC PDS writes an updated listing of the PDS member names to the SAS log.

PROC PDS operates with full capabilities on both extended partitioned data sets (PDSEs) and standard partitioned data sets (PDSs).

PROC PDS Statement

PROC PDS DDNAME=*file-specification* <*options*>;

DDNAME=*file-specification*
: specifies the physical file name (enclosed in quotes) or the fileref that is associated with the partitioned data set that you want to process. A fileref must have been previously assigned with a FILENAME statement, FILENAME function, a JCL DD statement, or a TSO ALLOCATE command. The DDNAME= argument is required.

The following options can appear in the PROC PDS statement:

NOLIST
: suppresses the listing of the member names and aliases in the directory of the partitioned data set.

KILL
: deletes all the members of the partitioned data set that is specified by DDNAME=.

REFRESH | NOREFRESH
: specifies whether to update the directory information of the file that is being processed after each operation. The default, REFRESH, updates the directory information after each operation. Unless the operations that are being performed by PROC PDS are dependent on each other, specify NOREFRESH for better performance.

STRICT
: causes error messages to be generated and sets the return code to 8 if no members match the selection criteria. The default behavior is for note messages to be generated and for the return code to be set to 0 if no members match the selection criteria.

DELETE Statement

DELETE *member-1* <. . . *member-n* >;

If you want to delete a member or members from the PDS, specify the member names in a DELETE statement.

When a specification in the DELETE statement is followed by a colon (:), all members whose names begin with the characters preceding the colon are deleted. For example, when the following statement is executed, PROC PDS deletes all members whose names begin with the characters PRGM:

```
delete prgm:;
```

CHANGE Statement

CHANGE *old-name-1 =new-name-1* < . . . *old-name-n=new-name-n* > ;

If you want to rename a member or members of the PDS, use the CHANGE statement. Specify the old name on the left side of the equal sign, and specify the new name on the right. For example, the following statements change the name of member TESTPGM to PRODPGM:

```
filename loadlib 'my.pgm.lib ';
proc pds ddname=loadlib;
   change testpgm=prodpgm;
run;
```

If multiple members have names that begin with the same sequence of characters and you want to change all of the names so that they begin with a different sequence, use a colon (:) after *old-name* and *new-name*. Here is an example:

```
change exam:=test:;
```

All of the members whose names began with the characters EXAM will subsequently have names beginning with the characters TEST.

Note: If changing the name of a member would make the name the same as that of an existing member, then the member is not renamed and a note is written to the SAS log. △

It is not necessary for the lengths of the character sequences that precede the colon to match. For example, the following statement is valid:

```
change am:=morn:;
```

However, if a new name is too long, then a note is written to the SAS log and no change is made.

EXCHANGE Statement

EXCHANGE *name-1=other-name-1* < . . . *name-n=other-name-n* > ;

Use the EXCHANGE statement to switch the names of members of the partitioned data set. For example, after the following statements are executed, the member originally called A will be member Z, and the member originally called Z will be member A.

```
proc pds ddname='my.pgm.lib';
   exchange a=z;
run;
```

If multiple members have names that begin with the same sequence of characters and you want to exchange that sequence with the sequence from another group of data sets, use a colon (:) after *name* and *other-name*. For example, after the following statement is executed, all data sets whose names began with ABC will begin with DEFG. In addition, all of the data sets whose names began with DEFG will begin with ABC, as shown:

```
exchange abc:=defg:;
```

It is not necessary for the lengths of the sequences of characters that precede the colons to match. However, if a new name is too long, then a note is written to the SAS log and no change is made.

Usage Note

Unlike other SAS procedures that deal with partitioned data sets (for example, PROC PDSCOPY and PROC SOURCE), PROC PDS does not make any distinction between a member name and an alias, other than to report which names in the PDS directory are aliases for which members. If an alias is renamed, it is still an alias. PROC PDS allows you to delete a member that has aliases in the PDS directory, but then other procedures (PROC PDSCOPY, for example) cannot process the aliases.

Example of Deleting and Renaming Members with the PDS Procedure

This example writes the names of the members of USERID.MVS.OUTPUT to the SAS log and then generates a second listing showing the member changes and deletions that are specified by the second PROC step. The results are shown in Output 16.8 on page 251.

```
filename pdstest 'userid.mvs.output';
proc pds ddname=pdstest;
run;

proc pds ddname=pdstest;
   delete tempout tempout2;
   change mem=out1603;
run;
```

Output 16.8 Deleting and Renaming Members with the PDS Procedure

```
1    filename pdstest 'userid.mvs.output';
2
3    proc pds ddname=pdstest;
4    run;
 SAS PROC PDS VERSION 8.00   04/27/99

   DSNAME=USERID.MVS.OUTPUT  VOL=SER=XXXXXX

   Members (aliases)

   MEM      OUT1601  OUT1602  TEMPOUT  TEMPOUT2

   Tracks Used        1.8
          Unused      1.2
          Total       3.0
   Extents              1

   Directory Blks      11
   Blocks Used          1

5
6    proc pds ddname=pdstest;
7       delete tempout tempout2;
8       change mem=out1603;
9    run;

   DSNAME=USERID.MVS.OUTPUT  VOL=SER=XXXXXX

   Members (aliases)

   MEM      OUT1601  OUT1602  OUT1603

   Tracks Used        1.8
          Unused      1.2
          Total       3.0
   Extents              1

   Directory Blks      11
   Blocks Used          1
```

PDSCOPY

Copies partitioned data sets from disk to disk, disk to tape, tape to tape, or tape to disk

OS/390 specifics: all

Syntax

PROC PDSCOPY INDD=*file-specification* OUTDD=*file-specification* <*options* >;
  EXCLUDE *member-name-1* <. . . *member-name-n*>;
  SELECT *member-name-1* <. . . *member-name-n* >;

Details

The PDSCOPY procedure can be used to copy an entire partitioned data set, or you can specify which members you want to copy. This procedure cannot be used to copy extended partitioned data sets (PDSEs). PROC PDSCOPY is useful for backing up source libraries and load module libraries to tape. If you use PROC PDSCOPY to copy a PDS to tape, then you must also use it if you want to copy that PDS back to *disk*. However, you can use either PROC PDSCOPY or other copy utilities to copy that tape to another *tape*.

When libraries are moved between disks that have different optimal block sizes, PROC PDSCOPY can be used to reblock the libraries. PROC PDSCOPY handles overlay programs and alias names. It also sets up the RLD count fields that are used by the FETCH program.

When a PDS contains load modules, it generally requires 13% to 18% less disk space after being copied by PROC PDSCOPY, because PROC PDSCOPY uses free space on a partially filled track to store records. The linkage editor constructs records that do not fit on a partially used track.

The PDSCOPY procedure does not copy scatter-loaded modules.

If errors are encountered during PDSCOPY processing, the return code for the job step is set to 8.

PROC PDSCOPY Statement

PROC PDSCOPY INDD=*file-specification* OUTDD=*file-specification* <*options*>;

INDD=*file-specification*
: specifies either the fileref or the physical file name (enclosed in quotes) of the library to copy. INDD= is required.

OUTDD=*file-specification*
: specifies either the fileref or the physical file name (enclosed in quotes) of the output partitioned data set. OUTDD= is required.

Options Some of the options that can appear in the PROC PDSCOPY statement apply to both source libraries and load module libraries. Others apply only to load module libraries. The following options apply to both source libraries and load module libraries:

ALIASMATCH=TTR

BLKSIZE=

INTAPE

NEWMOD

NOALIAS

NOREPLACE

OUTTAPE

SHAREINPUT

The following options apply only to load module libraries:

ALIASMATCH=BOTH | EITHER | NAME

DC

DCBS | NODCBS

MAXBLOCK=

NE

NOTEST

All the options that can appear in the PROC PDSCOPY statement are discussed in this section. In the discussion, the term *member* refers to both source library members and to load modules. The term *module* refers only to load modules.

ALIASMATCH=BOTH | EITHER | NAME | TTR

specifies how to match aliases with main members to determine whether they represent the same member.

BOTH

specifies that both the TTR (relative track and record) values and the names must match in order for a main module to be considered a match.

EITHER

specifies that a match for either the TTR value or the name is sufficient to identify the main module that corresponds to an alias. If more than one main module directory entry matches, it is impossible to predict which one will be used.

NAME

specifies that the main module name in the directory entry for the alias (at offset 36) is compared with main module names to find a match. The alias is assumed to represent the same module as the main module that has the matching name. When you specify ALIASMATCH=NAME, the TTR values do not need to match.

A situation in which names match even though TTR values do not match occurs when the main module is originally link edited specifying the alias names, and then link edited again without specifying them. In this case, the directory entries for the aliases still point to the old version of the module (that is, to a back-level version). Because of this, you should consider carefully whether to use the ALIASMATCH=NAME option or the NEWMOD option. ALIASMATCH=NAME updates the aliases to point to the current version of the main module rather than to the back-level version. The NEWMOD option causes the older version of the module to copy. Another alternative is to use TTR matching and not to copy the aliases when they are, in fact, obsolete names.

TTR

specifies that TTR values are compared. TTR is the default. An alias is assumed to represent the main member that has the same TTR value. If the TTR values match, then the directory entry for the main member and the alias currently point to the same place in the data set.

For load modules, the most common situation in which TTR values might match, but names may not match, occurs when the main module was renamed (for example, by using ISPF option 3.1) after the aliases were created. The alias directory entries may still contain the old main module name.

Whichever method you use, unmatched aliases are not copied to the output file unless you specify the NEWMOD option (see NEWMOD on page 256). Matched aliases in the output file always point to the main module to which they were matched (that is, they have the same TTR values), even if the TTR values were different in the input file (which might occur if ALIASMATCH=NAME or ALIASMATCH=EITHER was used). When modules are matched using the TTR values (that is, when TTR or EITHER was specified), the main module name in alias directory entries is changed in the output file.

BLKSIZE=*block-size*

specifies the maximum block size to use for reblocking the copied load modules on the output device. If the BLKSIZE= option is omitted, the default depends on the type of the output device and on the data set type:

- If output is to tape, the default is 32,760.
- If output is in tape (sequential) format on disk (that is, when the OUTTAPE option is used), the default is either the device track size or 32,760, whichever is less.
- If output is to disk, the default depends on the device type. However, it is never greater than 18K unless you use the MAXBLOCK= option (see MAXBLOCK on page 255). In addition, the default cannot exceed the device track size or 32,760, whichever is less.
- Unless the NODCBS option (described later) is specified and the output data set is a partitioned data set on disk, the default value is reduced to the data set control block (DSCB) block size of the partitioned data set, if that is smaller.

For tape (sequential) format output, the specified block size cannot be less than 1.125 times the maximum input device block size, nor greater than 32,760. For disk output, the specified block size cannot be less than 1,024.

DC

specifies that load modules that are marked downward compatible (that is, modules that can be processed by linkage editors that were used before MVS) are eligible for processing. After they are copied by PROC PDSCOPY, the load modules are not marked DC in their directory entry because PROC PDSCOPY does not produce downward compatible load modules nor does it preserve their attributes. If you do not specify the DC option and you attempt to copy load modules marked DC, PROC PDSCOPY issues an error message.

DCBS | NODCBS

tells SAS whether to preserve the data control block (DCB) characteristics of the output partitioned data set on disk. If NODCBS is specified, the data control block (DCB) characteristics of the output partitioned data set on disk can be overridden. The default value is DCBS.

If the NODCBS option is specified, PROC PDSCOPY changes the DSCB (data set control block) block size of the output partitioned data set to the maximum permissible block size for the device. Otherwise, the maximum permissible value of the BLKSIZE= option is the current block size value from the DSCB, and the DSCB block size is not changed.

Using the NODCBS option may enable PROC PDSCOPY to block output load modules more efficiently. However, changing the DSCB block size could cause problems when the data set is moved, copied, or backed up by a program other than PROC PDSCOPY, particularly if your installation has more than one type of disk drive. Consult your systems staff before specifying NODCBS.

INTAPE

specifies that the INDD= library is in tape (sequential) format. The INTAPE option is assumed if a tape drive is allocated to the input data set.

MAXBLOCK=*block-size* | MAXBLK=*block-size*

enables you to override the limitation of 18K on the block size of text records in the output library. (The value of BLKSIZE must be greater than or equal to the value of MAXBLOCK in order to get text records at MAXBLOCK size.) If the value of MAXBLOCK is not specified, then the maximum block size for text records is 18K; this is the largest text block that can be handled by the FETCH program in many operating environments. You can specify a block size greater than 18K for text records, but doing so may cause copied modules to ABEND with an ABEND code of 0C4 or 106-E when they are executed. You should use this parameter only if you are sure that your operating environment (or TP monitor) FETCH program supports text blocks that are larger than 18K. CICS and OS/390 FETCH programs, for example, support text blocks that are larger than 18K.

NE
: specifies that the output library should not contain records that are used in the link editing process. Although programs in the output library are executable, they cannot be reprocessed by the linkage editor, nor can they be modified by the AMASPZAP program. Using the NE option can reduce the amount of disk space that is required for the output library.

NEWMOD
: specifies that aliases that do not match a main member are to be copied as main members rather than being marked as aliases in the output file. The directory entry in the output file is reformatted to main member format. See the ALIASMATCH option for a description of how aliases are matched with main members. If you do not specify the NEWMOD option, unmatched aliases are not copied to the output file.

NOALIAS | NOA
: prevents automatic copying of all aliases of each member that you have selected for copying. Any aliases that you want to copy must be named in the SELECT statement. If you select only an alias of a member, the member (that is, the main member name) is still automatically copied, along with the selected alias.

NOREPLACE | NOR
: copies only members in the INDD= library that are not found in the OUTDD= library; that is, members or aliases that have the same name are not replaced.

NOTEST
: deletes the symbol records produced by the assembler TEST option from the copied load modules. Using the NOTEST option can reduce the amount of disk space that is required for the output library by 10% to 20%.

OUTTAPE
: specifies that the OUTDD= library is to be in tape (sequential) format. The OUTTAPE option is assumed if a tape drive is allocated to the output data set.

SHAREINPUT | SHAREIN
: specifies that the INDD= library is to be shared with other jobs and TSO users. SHAREINPUT is the default for PDSCOPY when the INDD= library is enqueued for shared control (DISP=SHR). This means that the INDD= library is shared with ISPF and the linkage editor rather than being enqueued exclusively. This makes it possible for more than one person to use an INDD= library simultaneously. (The OUTDD= library is always enqueued for exclusive control against ISPF and the linkage editor; therefore, it cannot be changed while PROC PDSCOPY is processing it.)

EXCLUDE Statement

EXCLUDE *member-name-1* <. . . *member-name-n*>;

Use this statement if you want to exclude certain members from the copying operation. The EXCLUDE statement is useful if you want to copy more members than you want to exclude. All members that are not listed in EXCLUDE statements are copied. You can specify as many EXCLUDE statements as necessary.

If you follow a specification in the EXCLUDE statement with a colon (:), then all members whose names begin with the characters preceding the colon are excluded.

Note: You cannot use both the SELECT statement and the EXCLUDE statement in one PROC PDSCOPY step. △

SELECT Statement

SELECT *member-name-1* <. . . *member-name-n*>;

Use this statement to specify the names of members to copy if you do not want to copy the entire library. You can specify as many SELECT statements as necessary.

If you follow a specification in the SELECT statement with a colon (:), then all members whose names begin with the characters preceding the colon are copied. In the following example all members whose names begin with the characters FCS are copied:

```
select fcs:;
```

Note: You cannot use both the SELECT statement and the EXCLUDE statement in one PROC PDSCOPY step. △

Output Data Set

The PDSCOPY procedure produces an output partitioned data set on disk or on tape. The output data set contains copies of the requested members of the input partitioned data set.

If you use PROC PDSCOPY to copy partitioned data sets that contain source members, then the RECFM and LRECL of the output data set must match those of the input data set. If they differ, an error message is displayed. The BLKSIZE values for the input and output data sets do not have to be the same, however.

Usage Notes

If a member that you specified in a SELECT statement does not exist, PROC PDSCOPY issues a warning message and continues processing.

PROC PDSCOPY enqueues the input and output data sets using the SPFEDIT and SPFDSN QNAMEs.

Output

The PDSCOPY procedure writes the following information to the SAS log:

- INPUT and OUTPUT, the data set names and volume serials of the input and output libraries
- MEMBER, a list of the members copied
- ALIAS, the members' aliases, if any
- whether the copied members replaced others members of the same name
- whether a selected member or alias was not copied and a note explaining why not.

If the output device is a disk, PROC PDSCOPY also writes the following information next to each member name:

- TRACKS, the size of the member, in tenths of tracks
- SIZE, the number of bytes in the member that was copied (in decimal notation).

Example of Copying Members Using the PDSCOPY Procedure

The following example copies all members and aliases that start with the letters OUT. In this example, the alias must match the main member both by name and by TTR in order for the alias to be copied.

```
filename old 'userid.mvs.output' disp=shr;
filename new 'userid.mvs.output2' disp=old;
proc pdscopy indd=old outdd=new aliasmatch=both
      shareinput;
   select  out:;
run;
```

Output 16.9 on page 258 shows the results.

Output 16.9 PDSCOPY Procedure Example

```
1    filename old 'userid.mvs.output' disp=shr;
2    filename new 'userid.mvs.output2' disp=shr;
3
4    proc pdscopy indd=old outdd=new aliasmatch=both shareinput;
5       select out:;
6    run;
    SAS PROC PDSCOPY VERSION 8.00   04/24/99

    INPUT  DSNAME=USERID.MVS.OUTPUT  VOL=SER=XXXXXX
    OUTPUT DSNAME=USERID.MVS.OUTPUT2  VOL=SER=XXXXXX

    MEMBER     TRACKS       SIZE
      ALIAS

    OUT1601     2.6        40019 replaced
    OUT1602    10.6       165519 replaced
    OUT1603    53.3       829081 replaced

    TRACKS USED    67.0
           UNUSED   8.0
           TOTAL   75.0
    EXTENTS         5
```

PMENU

Defines PMENU facilities for user-defined windows

OS/390 specifics: Some portable statements are ignored.

Details

The following statements and options are accepted without generating errors, but *with current device drivers* they have no effect under OS/390:

- ACCELERATE= option in the ITEM statement
- MNEMONIC= option in the ITEM statement
- HELP= option in the DIALOG statement.

See Also

- *SAS Procedures Guide*

PRINTTO

Defines destinations for SAS procedure output and for the SAS log

OS/390 specifics: UNIT= option; output destination

Details

In the SAS CLIST and the SAS cataloged procedure that are supplied by SAS Institute, no filerefs of the form FT*nn*F001 are predefined for the UNIT= option. Ask your SAS Installation Representative whether your site has predefined DDnames of the form FT*nn*F001.

Under OS/390, the destination of the procedure output or the SAS log can be specified by either of the following:

fileref
: sends the log or procedure output to a sequential data set or member of a partitioned data set that is identified by the fileref.

'*physical-filename*'
: sends the log or procedure output to a sequential data set, to a member of a partitioned data set, an extended partitioned data set, or to a file in a UNIX System Services hierarchical file system.

The following restrictions apply to PROC PRINTTO under OS/390:

- When writing log or procedure output files to a partitioned data set member, you must specify the NEW option; you cannot append data to a partitioned data set member.
- If you create a file to be used with the PRINTTO procedure and specify a record format that has no carriage control characters, the PROC PRINTTO output will not include carriage control characters.
- In order to simultaneously route both the SAS log and procedure output files to partitioned data set members, the members must be in different partitioned data sets.

See Also

- "Directing Output to External Files with the PRINTTO Procedure" on page 103
- *SAS Procedures Guide*

RELEASE

Releases unused space at the end of a disk data set

OS/390 specifics: all

Syntax

PROC RELEASE DDNAME=*file-specification* <*options*>;

Details

PROC RELEASE can be used with most sequential or partitioned data sets, not just with a SAS data library that contains SAS data sets. However, it cannot be used to release unused space from:

- □ the SAS WORK data library
- □ extended partitioned data sets (PDSEs)
- □ ISAM or VSAM data sets
- □ multivolume disk-format SAS data libraries
- □ external multivolume disk-format data sets.

If you delete some members from a SAS data library (using the DATASETS procedure, for example), you can use the RELEASE procedure to release the unused space at the end of the last member. You cannot use RELEASE to release embedded space. That is, you can release only unused space that follows the highest track or block that is currently in use, as indicated by the CONTENTS or DATASETS procedure.

In order to use PROC RELEASE on a SAS data library, the data library must be closed. If the library is open, SAS generates an error message. If you have assigned a libref to the data library and have used some members of that library in your SAS session, the library will be open. To close it, issue a LIBNAME statement of the following form:

LIBNAME *libref* CLEAR;

Then issue a new LIBNAME statement for the data library and immediately run PROC RELEASE. As an alternative to issuing a second LIBNAME statement, you can simply specify the data library's name (enclosed in quotes) as the value of the DDNAME= option in the PROC RELEASE statement, instead of using a libref.

In the control language, you can release unused space by using specifications such as SPACE=(,,RLSE) in the DD statement (in batch mode), or you can use the RELEASE operand of the TSO ALLOCATE command. However, releasing unused space with PROC RELEASE offers several advantages over methods provided by the operating environment. For example, with PROC RELEASE, the user, not the operating environment, controls when unused space is released. This advantage is especially applicable to TSO users.

Another advantage of PROC RELEASE is that you can use PROC RELEASE options to specify exactly how many tracks you want to keep or release. There is no danger of erasing all or part of a data set because PROC RELEASE frees only unused space. PROC RELEASE returns unused space to the pool of available space on the disk volume. Once released, the space is still available for allocation to the data set, provided a secondary space allocation is given for the data set in the control language or SAS statement, and provided all free space on the volume is not subsequently allocated to other data sets.

PROC RELEASE Statement

PROC RELEASE DDNAME=*file-specification* <*options*>;

DDNAME=*file-specification*
specifies either an physical file name (enclosed in quotes) or a fileref that refers to the physical file from which to release unused space. DDNAME= is required.

options
specify how much unused space to keep or release, and specify the unit boundary on which the data set should end.

TOTAL=*number* | TRACKS=*number*
specifies the total number of tracks that the data set should contain after unused space is released, that is, after PROC RELEASE has executed. For example, the following statement releases all but ten tracks for the data set that is referenced by the fileref SURVEY:

```
proc release ddname=survey total=10;
```

The procedure calculates the amount of space to release as follows:

amount of space allocated– (value of TOTAL= option) = amount of unused space released

If the value that you specify is smaller than the amount of used space in the data set, then SAS releases only the unused space at the end of the data set.

UNUSED=*number*
specifies the number of tracks of unused space that the data set should contain after PROC RELEASE has executed. The procedure calculates the amount of unused space to release as follows:

amount of space allocated– (used space + value of UNUSED= option) = amount of unused space released

If the value that you specify is greater than the amount of unused space at the end of the data set, then no space is released at the end of the data set.

RELEASE=*number*
specifies how many tracks of unused space to release. If the value that you specify is greater than the amount of unused space at the end of the data set, then SAS releases all the unused space at the end of the data set.

EXTENTS | EXTENT | EX
tells SAS to release only the space that is allocated to completely unused secondary extents. After the procedure releases unused space from the data set, the size of the data set is the sum of the primary extent plus all used secondary extents.

If you do not specify one of these options in the PROC RELEASE statement, then all unused space at the end of the data set is released.

Use the following option to specify the unit boundary on which the data set should end:

BOUNDARY=*type* | TYPE=*type*
specifies whether the data set will end on a track boundary or on a cylinder boundary.

After the total amount of space to be retained is calculated, this amount is rounded up to the next unit boundary. Any remaining space is then released. Remember that the total amount of space will include the space that is actually used and may also include unused space that was requested with other options. BOUNDARY=*type* then will increase the amount of unused space that is retained in the data set by the portion of the unit that is needed in order to reach (or round up to) the next boundary. TYPE can be one of the following:

DATASET | DSCB
: specifies that the data set will end on the next track or cylinder boundary depending on how space is currently allocated. If allocated in tracks, the total amount of space to be retained is calculated, and remaining unused tracks are released. If allocated in cylinders, the space to be retained is rounded up to the next cylinder boundary, and remaining unused space is released. This is the default boundary type.

CYLINDERS | CYLINDER | CYLS | CYL
: specifies that space to be retained is rounded to the next cylinder boundary before remaining unused space is released. This specification is effective only if the data set is currently allocated in multiples of cylinders.

TRACKS | TRACK | TRKS | TRK
: specifies that unused tracks are to be released. Because the minimum unit of space that can be released is a track, the space to be retained is not rounded up.

ALLOC | DD | JCL
: specifies that space to be retained is rounded to the next unit boundary (tracks or cylinders) depending on the allocation unit that was specified in the JCL statement, TSO ALLOCATE command, FILENAME or LIBNAME statement, or FILENAME or LIBNAME function. For example, the following, in combination with BOUNDARY=DD, is equivalent to specifying BOUNDARY=CYL:

```
//DD2 DD DISP=OLD,DSN=MY.DATA,
//       SPACE=(CYL,2)
```

Usage Notes

If the messages in the SAS log indicate that no space was released from the data set, check to see whether the data set is allocated to another job or to another user. When PROC RELEASE is invoked, the operating environment's disk space management function (DADSM) must be able to obtain exclusive control of the data set. If it cannot, then no indication that DADSM does not have control is passed to the SAS System, no space is released from the data set, and no error message is issued by the SAS System.

Output

PROC RELEASE writes the following information to the SAS log:

- how many tracks were allocated to the data set before and after the procedure was executed
- how many tracks were used
- how many extents were used.

Example

The following example releases the unused secondary extents for an physical file that is referenced by the fileref THISFILE:

```
filename thisfile 'my.pgm.lib';
proc release ddname=thisfile extents;
```

```
run;
```

See Also

- IBM's *OS/390 MVS JCL Reference*

SORT

Sorts observations in a SAS data set according to the values of one or more variables

OS/390 specifics: available OS/390 sort utilities and SORT procedure statement options; host-specific SAS system options

Details

You can direct the SORT procedure to use either the SAS sort program, which is available under OS/390 and under all other operating environments or a sort utility that is specific to OS/390. You can also use the SORTPGM= system option to tell SAS to choose the best sort program to use. (See "SORTPGM=" on page 403.)

The following SAS system options also affect any sorting that is done by SAS:

DYNALLOC	SORTEQOP	SORTSHRB
FILSZ	SORTLIB=	SORTSIZE=*
SORT=	SORTLIST	SORTSUMF
SORTALTMSGF	SORTMSG	SORTUADCON
SORTBLKMODE	SORTMSG=	SORTUNIT=
SORTBUFMOD	SORTNAME=	SORTWKDD=
SORTCUTP=	SORTOPTS	SORTWKNO=
SORTDEV=	SORTPARM=	SORT31PL
SORTDEVWARN	SORTPGM=	
SORTDUP=*	SORTSEQ=*	

* Options marked with an asterisk (*) are either portable or portable with host specifics. For information on these options, begin with *SAS Language Reference: Dictionary*.

You can see the value of the preceding options by submitting:

```
proc options group=sort; run;
```

SORT Procedure Statement Options

The following host-specific sort options are available in the PROC SORT statement under OS/390 in addition to the statement options that are available under all host operating environments. The list includes the portable EQUALS option because it has aspects that are specific to OS/390.

DIAG
: passes the DIAG parameter to the sort utility. If the utility supports this option, then it will produce additional diagnostic information if the sort fails.

EQUALS
: passes the EQUALS parameter to the sort utility program whether or not the sort utility supports it. The SAS System defaults to EQUALS by passing the parameter to the utility if the SAS system option SORTEQOP is in effect.

LEAVE=*n*
: specifies how many bytes to leave unallocated in the region. Occasionally, the SORT procedure runs out of main storage. If this happens, rerun the job and increase the LEAVE= value (which has a default value of 16000) by 30000.

LIST | L
: provides additional information about the system sort. Not all sort utilities support the specification of the LIST option; they may require that it be specified when the sort utility is generated or installed. This option is the default action if the SAS system option SORTLIST is in effect. Also, this option overrides NOSORTLIST if it is in effect.

MESSAGE | M
: prints a summary of the system sort utility's actions. This option is the default action if the SAS system option SORTMSG is in effect. Also, this option overrides NOSORTMSG if it is in effect. MESSAGE is useful if you run PROC SORT and the SAS log prints a message indicating that the sort did not work properly. Explanations of the message can be found in the IBM or vendor reference manual that describes your system sort utility.

SORTSIZE=*n* | *n*K | *n*M | *n*G | MAX | SIZE
: specifies the maximum virtual storage that can be used by the system sort utility. If not specified, the default sort size is given by the SAS system option SORTSIZE=.

SORTWKNO=*n*
: specifies how many sort work areas PROC SORT allocates. If not specified, the default is given by the SAS system option SORTWKNO=.

TECHNIQUE=*xxxx* | T=*xxxx*
: specifies a four-character sort technique to be passed to the system sort utility. SAS does not check the validity of the specified value, so you must ensure that it is correct.

See Also

- "Summary Table of SAS System Options" on page 419
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *SAS Procedures Guide*

SOURCE

Provides an easy way to back up and process source library data sets

OS/390 specifics: all

Syntax

PROC SOURCE <*options*>;

SELECT *member-1* < . . . *member-n* >;

EXCLUDE *member-1* < . . . *member-n* > ;

FIRST '*model-control-statement* ';

LAST '*model-control-statement* ';

BEFORE '*model-control-statement* '<*options* >;

AFTER '*model-control-statement*' <*options* >;

Details

Use PROC SOURCE to read PDS or PDSE libraries and produce sequential output.

You can use the SOURCE procedure to:

- □ write the contents of an entire library to the SAS log.
- □ process only the directory of a library in order to produce input for the SAS System, for a utility, or for other programs.
- □ route the members of a library to other programs for processing. By default, PROC SOURCE generates records for the IBM utility, IEBUPDTE, which reloads an unloaded data set.
- □ create a sequential, or unloaded, version of the library's directory records.
- □ construct an unloaded data set from a library. The unloaded data set is suitable for reloading by IEBUPDTE or other source library maintenance utilities, including the ability to recognize and properly handle aliases.

Using the SOURCE procedure, a source library can be copied into a sequential tape or disk data set to create either a backup or a manually transportable copy of the source data. This copy is called an *unloaded data set*; it consists of 80-byte records that contain the source data and the control information that are needed to restore the source to its original organization. When an unloaded data set is restored by the proper utility to a device that will support the data in their original form, the data is reconstructed, or *loaded*.

An advantage of having an unloaded data set is that one or more members can be retrieved without reloading the entire library.

PROC SOURCE has several advantages over IBM's IEBPTPCH utility. With PROC SOURCE you can

- □ list members in alphabetical order
- □ select members by specifying a wildcard or range
- □ list the number of records in each member
- □ list each member on a new page
- □ produce an unloaded version of the library that can be ported to some other host systems.

The *model-control-statements* in the FIRST, LAST, BEFORE, and AFTER statements are usually either utility or job control statements, depending on the destination given by the OUTDD= option in the PROC SOURCE statement.

PROC SOURCE Statement

PROC SOURCE <*options*>;

The following options are used in the PROC SOURCE statement:

DIRDD=*file-specification*
: specifies either the fileref or physical file name of the output data set to which PROC SOURCE writes a sequential, unloaded form of the PDS directory. Each directory record is written into one 80-byte record. Records are left-aligned and padded on the right with blanks. If specified, the fileref must match the reference name that was used in the FILENAME statement, FILENAME function, JCL DD statement, or TSO ALLOCATE command that allocated the output data set.

Note: The SELECT and EXCLUDE statements have no effect when the DIRDD= option is specified. △

INDD=*file-specification*
: specifies the fileref or the physical file name of an input PDS. The fileref, if specified, must match the reference name that was specified in the FILENAME statement, FILENAME function, JCL DD statement, or TSO ALLOCATE command that allocated the input library. If the INDD= option is not specified, the default fileref is SOURCE.

If OUTDD is specified, then the RECFM of the INDD file must be either F or FB. The fileref may not refer to a concatenation of data sets. If it does, then an error message is generated. If the member names in the INDD file are nonstandard, then specify FILEEXT=ASIS in an OPTIONS statement.

MAXIOERROR=*n*
: specifies the maximum number of I/O errors to allow before terminating. Normally, PROC SOURCE detects, issues a warning message about, and then ignores I/O errors that occur while reading the library members. When the number of errors specified by MAXIOERROR= has occurred, however, PROC SOURCE assumes that the library is unreadable and stops. The default MAXIOERROR= value is 50.

NOALIAS
: treats aliases as main member names. Therefore, PROC SOURCE does not generate

```
./ ALIAS
```

cards or alias BEFORE and AFTER cards.

NODATA
: specifies that you do not want to read the members in the input PDS. In other words, PROC SOURCE produces only control statements and a list of the member names; it does not produce the contents of the members. The list of member names includes any aliases. NODATA is particularly useful when you want to process only the directory of a library.

NOPRINT
: specifies that you do not want to generate the list of member names and record counts. (These listings are produced even when the PRINT option is not specified.) The NOPRINT option is ignored when PRINT is specified.

NOSUMMARY
: specifies that you do not want to generate the member summary. The NOSUMMARY option is ignored when the NODATA, NOPRINT, or PRINT option is specified.

NOTSORTED
: causes PROC SOURCE to process PDS members in the order in which they either appear (in SELECT statements) or remain (after EXCLUDE statements).

Normally, PROC SOURCE processes (that is, unloads, writes to the SAS log, and so on) the PDS members in alphabetical order by member name.

NULL
: specifies that null members (PDS members that contain no records, just an immediate end-of-file) should be processed. Such members occasionally appear in source PDSs, but they are not normally unloaded because IEBUPDTE and most other PDS maintenance utilities do not create null members. If you are using a source library maintenance utility that can properly recognize and create a null member, then specify this option and provide the appropriate BEFORE (and possibly, AFTER) statements.

OUTDD=*file-specification*
: specifies the fileref, PDS or PDSE member name, or UNIX System Services filename of the output file to which PROC SOURCE writes the unloaded (sequential) form of the input PDS and any records that FIRST, LAST, BEFORE, and AFTER statements generate. If specified, the fileref must match the reference name used in the FILENAME statement, FILENAME function, JCL DD statement, or TSO ALLOCATE command that allocated the data set. This option cannot be used when the INDD file contains variable-length records.

PAGE
: begins the listing of the contents of each member on a new page.

PRINT
: lists the contents of the entire PDS. The PRINT option is ignored when NODATA is specified.

SELECT Statement

SELECT *member-1* < . . . *member-n* >;

When you use the SELECT statement, only the members that you specify are processed. You can use any number of SELECT statements.

Use a colon (:) to indicate that you want to select all members whose names begin with the characters that precede the colon. (See the second example below.)

You can include an alphabetic range of names in the SELECT statement by joining two names with a hyphen (-). The two hyphenated members and all members in between are processed. For example, if a library contains members called BROWN, GRAY, GREEN, RED, and YELLOW, and you want to process the first four members, use this SELECT statement:

```
select brown-red;
```

The colon (:) and hyphen (-) notation can be used together. For example, the following statement produces the same results as the previous SELECT statement:

```
select br:-gr: red;
```

EXCLUDE Statement

EXCLUDE *member-1* < . . . *member-n* > ;

When you use the EXCLUDE statement, all members except those that you specify are processed. You can use any number of EXCLUDE statements.

Use a colon (:) to indicate that you want to exclude all members whose names begin with the characters that precede the colon.

You can include an alphabetic range of names in the EXCLUDE statement by joining two names with a hyphen. The two hyphenated members and all members in between are excluded from processing. (See the SELECT examples in the SELECT statement description.)

The colon and hyphen notation can be used together.

Sometimes it is convenient to use SELECT and EXCLUDE statements together. For example, you can use the colon or hyphen notation in a SELECT statement to select many members, then use the EXCLUDE statement to exclude a few of the selected members. Suppose there are 200 members called SMC1 through SMC200, and you want to copy all of them except SMC30 through SMC34. You could use these statements:

```
select smc:;
exclude smc30-smc34;
```

When you use both EXCLUDE and SELECT statements, the EXCLUDE statements should specify only members that are specified by the SELECT statements. However, excluding unselected members has no effect other than to generate warning messages.

FIRST Statement

FIRST '*model-control-statement* ';

The FIRST statement generates initial control statements that invoke a utility program or that are needed only once. The specified *model-control-statement* is reproduced, left-aligned, on a record that precedes all members in the unloaded data set. You can use any number of FIRST statements. One FIRST statement can specify one model control statement. Each model control statement generates a record.

LAST Statement

LAST '*model-control-statement* ';

The LAST statement generates final control statements that terminate a utility program or that are needed only once. The specified *model-control-statement* is reproduced, left-aligned, on a record that follows all members in the unloaded data set. You can use any number of LAST statements. One LAST statement can specify one model control statement. Each model control statement generates a record.

BEFORE Statement

BEFORE '*model-control-statement*' <*options*>;

The BEFORE statement generates a utility control statement before each member. You can use any number of BEFORE statements. One BEFORE statement can specify one model control statement. Each *model-control-statement* that you specify is reproduced, left-aligned, on a record that precedes each member in the unloaded data set.

By default, PROC SOURCE generates control statements for the IBM IEBUPDTE utility program before each member of an unloaded data set. You can use the BEFORE and AFTER statements to override the default and generate control statements for other utility programs. To prevent PROC SOURCE from generating these statements, use the BEFORE statement with no parameters.

Options for the BEFORE and AFTER statements are the same. A list of these options follows the description of the AFTER statement.

AFTER Statement

AFTER '*model-control-statement*' <*options*>;

The AFTER statement generates a utility control statement after each member. You can use any number of AFTER statements. One AFTER statement can specify one model control statement. Each *model-control-statement* that you specify is reproduced, left-aligned, on a record that follows each member in the unloaded data set.

By default, PROC SOURCE generates control statements for the IBM IEBUPDTE utility program after each member of an unloaded data set. You can use the AFTER statement to override the default and generate control statements for other utility programs.

The following options are used in the BEFORE and AFTER statements:

ALIAS
: tells SAS to produce a record containing the *model-control-statement* only for each defined alias. (The alias is placed into the record at the specified column, if any.)

column number
: tells SAS to substitute the member name in records that are generated by BEFORE and AFTER statements in an 8 byte field beginning in this column. The beginning column can be any column from 1 to 73. Aliases, as well as main member names, are substituted. The name is left-aligned in the field unless the RIGHT option is specified, and it is padded on the right with blanks unless the NOBLANK option is specified.

NOBLANK
: is meaningful only if *column number* is specified. When the member name is substituted in records that are generated by the BEFORE and AFTER statements, NOBLANK eliminates blanks between the end of the member and any text that follows. In the following record, a member name precedes the text; NOBLANK has *not* been specified:

```
name ,text text text
```

When NOBLANK is specified, the same record looks like this:

```
name,text text text
```

RIGHT
: is meaningful only if *column number* is specified. When the member name is substituted in records that are generated by the BEFORE and AFTER statements, RIGHT causes the member name to be right-aligned in the specified field. By default, the name is left-aligned in an eight-column field.

Output

PROC SOURCE writes the following information to the SAS log:

- the contents of the entire PDS, if the PRINT option is specified
- a listing of the member names in the PDS (unless you specify NOPRINT)
- the number of records for each member (unless you specify NOPRINT or NODATA)
- a summary of the attributes and contents of the PDS.

Even when PRINT is not specified, some records may still be written to the log. The signal NAME: or ENTRY: or AUTHOR: beginning in column 5 of a record in the library starts the listing; the signal END beginning in column 5 stops it. If you do not want SAS to list this subset of records, specify the NOSUMMARY option.

Example of Printing Selected Members from a PDS

The following example writes to the SAS log the contents of the member ORANGES4 from the PDS USERID.TASTE.TEST:

```
proc source indd='userid.taste.test' print;
   select oranges4;
run;
```

The log is shown in Output 16.10 on page 270.

Output 16.10 Selecting a Member from a Source Statement Library

```
19   proc source indd='userid.taste.test' print;
20   select oranges4; run;
ORANGES4
data oranges;
   input variety $ flavor texture looks;
   total=flavor+texture+looks;
   datalines;
   navel 9 8 6
   temple 7 7 7
   valencia 8 9 9
   mandarin 5 7 8
   ;

proc sort data=oranges;
   by descending total;

proc print data=oranges;
   title 'Taste Test Result for Oranges';
17 - RECORDS

NOTE: INDD=SYS00158 data set is :
      Dsname=USERID.TASTE.TEST,
      Unit=3380,Volume=XXXXXX,Disp=SHR,Blksize=23055,
      Lrecl=259,Recfm=FB.

        3348      Members defined in source library.
           0      Aliases defined in source library.
           1      Members selected.
          17      Records read from source library.
```

Example of Building and Submitting a Job to Assemble Programs

The following PROC SOURCE program builds and submits a job to compile assembler programs. It writes the output directly to the internal reader so that the compile job can be executed.

```
filename out sysout=a pgm=intrdr lrecl=80
      recfm=f;
proc source indd='userid.asm.src' nodata
      outdd=out;
   first '//COMPILE JOB (0,ROOM),''DUMMY'',';
   first '// NOTIFY=,REGION=4M,TYPRUN=HOLD';
   first '/*JOBPARM FETCH';
   last  '//';
   before '//XXXXXXXX EXEC ASMHCL,' 3;
   before '// MAC2=''XXX.MACLIB'' ';
   before '//SYSIN DD DISP=SHR,';
   before '// DSN=USERID.ASM.SOURCE(XXXXXXXX)' 26 NOBLANK;
run;
```

The output that is written to the internal reader is shown below. Note that this output shows only the statements that are generated by PROC SOURCE, before they are executed.

Output 16.11 Building and Submitting a Job to Assemble Programs

```
//COMPILE JOB (0,ROOM),'DUMMY',
// NOTIFY=,REGION=4M,TYPRUN=HOLD
/*JOBPARM FETCH
//OUT1601  EXEC ASMHCL,
// MAC2='XXX.MACLIB'
//SYSIN DD DISP=SHR,
// DSN=USERID.ASM.SRC(OUT1601)
//OUT1602  EXEC ASMHCL,
// MAC2='XXX.MACLIB'
//SYSIN DD DISP=SHR,
// DSN=USERID.ASM.SRC(OUT1602)
//OUT1603  EXEC ASMHCL,
// MAC2='XXX.MACLIB'
//SYSIN DD DISP=SHR,
// DSN=USERID.ASM.SRC(OUT1603)
//
```

Example of Producing Directory Records

The following PROC SOURCE program produces directory records. The subsequent DATA step extracts the ISPF statistics, if any are present.

```
filename indd 'userid.sas.src' disp=shr;
filename out  '&temp';
/* Build directory records. */
proc source indd=indd nodata noprint dirdd=out;

/* Read directory records and extract   */
/*      ISPF statistics.                */
data test;
infile out;
file print header=h;
input member $8. ttr pib3. ind pib1. @;
datalen = 2*mod(ind,32);
if (datalen = 30)
then do;
  input ver pib1. mod pib1. blank pib2.
        ccreate pib1.
         create pd3.
        cchanged pib1.
         changed pd3. hh pk1.
        mm pk1. size pib2. init pib2.
        modl pib2. userid $8.;
  yyyydddc = (ccreate * 100000) + 1900000 + create;
  jcreate = datejul(yyyydddc);
  yyyydddx = (cchanged * 100000) + 1900000 + changed;
  jchange = datejul(yyyydddx);

/* Print the results. */
  put @4 member $8.
      @15 jcreate yymmdd10.
      @27 jchange yymmdd10.
      @39 hh 2. ':' mm 2.
      @48 userid;
end;
return;
```

```
h:
put @4 'NAME '
    @15 'CREATED'
    @27 'CHANGED'
    @39 'TIME'
    @48 ' ID ';
put;
return;
run;
```

The following output shows the results.

Output 16.12 Producing Directory Records

```
                              The SAS System
    NAME       CREATED     CHANGED     TIME      ID

    OUT1601    1999-02-20  1999-02-20  10:50    USERID
    OUT1602    1999-02-20  1999-02-20  10:54    USERID
    OUT1603    1999-02-20  1999-02-20  10:59    USERID
```

Example of Generating Control Cards for IEBCOPY

This example first produces control statements for the IBM utility program, IEBCOPY. Then IEBCOPY executes, copying selected members.

```
//IEBPDS JOB (0,ROOM),'USERID',
//  NOTIFY=
/*JOBPARM FETCH
//   EXEC SAS
//IN DD DSN=XXX.SUBLIB,DISP=SHR
//OUT DD DSN=&&TEMP,SPACE=(CYL,(1,2)),
//       DISP=(,PASS),UNIT=DISK
//SYSIN DD *
   proc source indd=in outdd=out nodata  noprint;
   select hc:;
   select lm:;
   select sasextrn;
   first ' COPY INDD=IN,OUTDD=NEWPDS';
   before '  SELECT MEMBER=XXXXXXXX -----------'
      17;
   before '       S      M=XXXXXXXX ***ALIAS***'
      17 ALIAS;
//S1     EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//IN     DD DSN=XXX.SUBLIB,DISP=SHR
//NEWPDS DD DSN=&&NEW,SPACE=(CYL,(20,10,20)),
//          UNIT=DISK
//SYSUT1 DD UNIT=DISK,SPACE=(CYL,(2,3))
//SYSUT2 DD UNIT=DISK,SPACE=(CYL,(2,3))
//SYSUT3 DD UNIT=DISK,SPACE=(CYL,(2,3))
//SYSIN  DD DSN=&&TEMP,DISP=(OLD,DELETE)
```

Output 16.13 on page 273 shows what is written to the SAS log after PROC SOURCE is run. Output 16.14 on page 273 shows the IEBCOPY output.

Output 16.13 Producing Control Statements for the IEBCOPY Utility

```
1             proc source indd=in outdd=out nodata  noprint;
2             select hc:;
3             select lm:;
4             select sasextrn;
5             first ' COPY INDD=IN,OUTDD=NEWPDS';
6             before '  SELECT MEMBER=XXXXXXXX -----------' 17;
7             before '       S      M=XXXXXXXX ***ALIAS***' 17 ALIAS;

NOTE: INDD=IN data set is :
      Dsname=USERID.DATASET,
      Unit=3380,Volume=XXXXXX,Disp=SHR,Blksize=6160,
      Lrecl=80,Recfm=FB.

NOTE: OUTDD=OUT data set is :
      Dsname=SYS96052.T131013.RA000.IEBPDS.TEMP,
      Unit=3390,Volume=,Disp=NEW,Blksize=27920,
      Lrecl=80,Recfm=FB.

           9      Members defined in source library.
           0      Aliases defined in source library.
           6      Members selected.
           0      Records read from source library.
```

Output 16.14 IEBCOPY Output: Selected Members Copied

```
                 IEBCOPY MESSAGES AND CONTROL STATEMENTS
 COPY INDD=IN,OUTDD=NEWPDS
  SELECT MEMBER=HCMEM1   -----------
  SELECT MEMBER=HCMEM2   -----------
  SELECT MEMBER=HCMEM3   -----------
  SELECT MEMBER=LMMEM1   -----------
  SELECT MEMBER=LMMEM2   -----------
  SELECT MEMBER=SASEXTRN -----------
          .
          .
          .
IEB167I FOLLOWING MEMBER(S) COPIED FROM INPUT DATA SET REFERENCED BY IN
IEB154I HCMEM1   HAS BEEN SUCCESSFULLY COPIED
IEB154I HCMEM2   HAS BEEN SUCCESSFULLY COPIED
IEB154I HCMEM3   HAS BEEN SUCCESSFULLY COPIED
IEB154I LMMEM1   HAS BEEN SUCCESSFULLY COPIED
IEB154I LMMEM2   HAS BEEN SUCCESSFULLY COPIED
IEB154I SASEXTRN HAS BEEN SUCCESSFULLY COPIED
IEB144I THERE ARE 239 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY NEWPDS
IEB149I THERE ARE 8 UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
IEB147I END OF JOB - 0 WAS HIGHEST SEVERITY CODE
```

References

□ IBM's *DFSMS/MVS: Utilities*

TAPECOPY

Copies an entire tape volume (tape or cartridge), or files from one or several tape volumes, to one output tape volume

OS/390 specifics: all

Syntax

PROC TAPECOPY *options* ;

INVOL *options*;

FILES *file-numbers* ;

Details

PROC TAPECOPY always begins writing at the beginning of the output tape volume; any files that previously existed on the output tape are destroyed.

Note: PROC TAPECOPY copies to a *single* output tape volume. △

The TAPECOPY procedure can copy either standard labeled or nonlabeled tapes or cartridges. You can specify, within limits, whether the output tape is standard labeled (SL) or nonlabeled (NL). You cannot create an SL tape using an NL input tape because TAPECOPY cannot manufacture tape labels. Also, if LABEL=(,SL) was specified in a DD statement for an output tape volume, you cannot change that tape into a nonlabeled tape. PROC TAPECOPY does allow you to write over an existing volume label on a standard labeled tape if you specify LABEL=(,BLP) in the DD statement. (The BLP value indicates bypass label processing.)

The JCL DD statement parameter LABEL=(,BLP) must be authorized specifically by each computing installation. If your installation allows the BLP specification, then ANSI-labeled, nonstandard labeled, and standard user-labeled tapes can be treated as nonlabeled tape volumes. If the BLP specification is not authorized at your installation, then LABEL=(,BLP) is treated as LABEL=(,NL). PROC TAPECOPY will work as you expect if your tape is in fact nonlabeled; otherwise, the operating environment does not allow TAPECOPY to use the tape, thus preserving the label.

Throughout this description, references to specifying LABEL=(,BLP) assume that LABEL=(,BLP) is a valid specification at your installation.

CAUTION:

Record lengths cannot exceed 32K bytes. PROC TAPECOPY copies up to 32K bytes of data per record, even if the length of the record exceeds 32K. No error message is generated. △

Input Tape DD Statement Requirements In the DD statement that describes an input tape, you need to specify the UNIT, VOL=SER, DISP parameters, and usually either the LABEL or DSN parameter.

VOL=SER gives the volume serial of the first input tape. You can omit VOL=SER if the UNIT parameter specifies deferred mounting–for example, UNIT=(*tape*,,DEFER). If you specify deferred mounting, remember to use the INVOL= option in the PROC TAPECOPY statement or in an INVOL statement to specify the volume serial of the input tape. For details , see the information on the INVOL= option or “INVOL Statement” on page 277.

For a nonlabeled input tape, you must specify either LABEL=(,NL) or LABEL=(,BLP) in the DD statement. If you are unsure whether the input tape volume is labeled or nonlabeled, specify LABEL=(,BLP) in the input tape DD statement, if your installation allows it.

For a standard labeled input tape at an installation that does not allow LABEL=(,BLP), specify LABEL=(,SL) and the DSN parameter, giving the DSNAME of the first data set on the tape.

Output Tape DD Statement Requirements In the DD statement that describes the output tape, you usually need to specify only the UNIT, VOL=SER, and DISP parameters, and possibly the LABEL or DSN parameters.

VOL=SER gives the volume serial of the output tape. You can omit VOL=SER if the UNIT parameter specifies deferred mounting–for example, UNIT=(*tape*,,DEFER). If you specify deferred mounting, use the OUTVOL= option in the PROC TAPECOPY statement to specify the volume serial of the output tape. For details, see the information on the OUTVOL= option below.

You should usually specify DISP=(NEW,KEEP) for the output tape in the DD statement. At some installations it may be necessary to specify DISP=(OLD,KEEP) along with the DSN parameter, giving the DSNAME of the first data set on the tape volume. The LABEL parameter should give the tape's label type as it is before the TAPECOPY procedure is executed, regardless of its label type after the copying operation.

Output The TAPECOPY procedure writes to the SAS log a listing of the input and output tape characteristics plus a summary of the files that were copied.

PROC TAPECOPY Statement

PROC TAPECOPY *options* ;

The following options can appear in the PROC TAPECOPY statement. The options are listed alphabetically.

COPYVOLSER

specifies that the output tape should have a standard label with the same volume serial as the first input tape. COPYVOLSER is effective only when both of the following conditions are true:

- the output tape volume is to be standard labeled–that is, LABEL=SL
- the output tape DD statement specifies LABEL=(,NL) or LABEL=(,BLP).

Both of these conditions must be true because the PROC TAPECOPY statement LABEL= option specifies whether the output tape is standard labeled or nonlabeled *after* the copy operation. The output tape volume's DD statement LABEL= parameter specifies what the output tape's label status is *before* the copy operation.

If you specify COPYVOLSER and these conditions are not true, PROC TAPECOPY stops processing.

DEN=*density*

specifies the density of the output tape. (The DEN= option should not be specified for cartridge tapes.) If the DEN= option appears in the PROC TAPECOPY statement, it overrides any DCB=DEN specification in the DD statement for the output tape volume. If you do not specify a density in the PROC TAPECOPY statement or in the DD statement, the operating environment writes the tape at its default density. The default density is usually the highest density at which the unit allocated to the output tape volume can record.

Valid density values follow:

Tape Density Value	Tape Volume Type
DEN=2	800 bpi
DEN=800	
DEN=3	1600 bpi
DEN=1600	
DEN=4	6250 bpi
DEN=6250	

INDD=*DDname*
specifies the DDname that is referenced in the JCL DD statement for the first input tape volume. The default INDD= option value is VOLIN.

INVOL=*volume-serial*
specifies the volume serial of the first input tape when deferred mounting is specified in the DD statement for the first input tape. The INVOL= option specification overrides the volume serial, if any, that was specified in the DD statement for the tape.
Specify the INVOL= option only if you are using deferred mounting.

LABEL=SL | NL
specifies whether the output tape volume is to be standard labeled (LABEL=SL) or nonlabeled (LABEL=NL).

Note: Be careful not to confuse the LABEL= option in the PROC TAPECOPY statement with the DD statement parameter LABEL=(,*specification*). The PROC TAPECOPY statement LABEL= option specifies whether the output tape is standard labeled or nonlabeled *after* the copy operation. The output tape volume's DD statement LABEL= parameter specifies what the output tape's label status is *before* the copy operation. △

The DD statement for nonlabeled output tapes must specify either LABEL=(,NL) or LABEL=(,BLP). If the output tape has an existing label (before the copy operation) and the output tape is to be nonlabeled (after the copy operation), then the DD statement must specify LABEL=(,BLP).

The default LABEL= option value is NL when multiple input volumes are used and when the DD statements for any of them specify LABEL=(,NL). If there are multiple input tapes and LABEL=(,NL) is not specified for any of them, and if the first input tape volume is actually standard labeled, then the default LABEL= option value is SL. This is true even if the DD statement specifies LABEL=(,BLP) for the first tape; in this case, PROC TAPECOPY reads the tape volume's first record to determine the actual label type.

NEWVOLSER=*new-volume-serial*
specifies a new volume serial for the output tape. NEWVOLSER is effective only if the output tape is standard labeled. If the output tape has an existing label, then the DD statement for the output tape must specify LABEL=(,BLP); otherwise, PROC TAPECOPY stops processing and does not write over the label.

NOFSNRESEQ | NFR
specifies that file sequence numbers in the file labels should not be resequenced when a standard labeled output tape volume is being produced. PROC TAPECOPY usually resequences these numbers and updates the label in order to reflect both the ordinal position of the file on the output tape as it is copied and the actual density at which the output tape is written.

NOLIST
: tells SAS not to write the tape characteristics and the summary of copied files to the SAS log. Even when you specify NOLIST, the SAS log contains a brief summary of PROC TAPECOPY's action; this summary is usually enough to verify proper functioning of PROC TAPECOPY if you are familiar with the contents of the input tape(s).

NORER
: tells SAS not to specify the "reduced error recovery for tape devices" feature of the operating environment for each input tape volume. When NORER is specified, some tapes of marginal quality can be read successfully by PROC TAPECOPY because the error recovery procedures are more extensive.

OUTDD=*DDname*
: specifies the DDname that is referenced in the JCL DD statement for the output tape. The default OUTDD= option value is VOLOUT.

OUTVOL=*volume-serial*
: specifies the volume serial of the output tape when deferred mounting is specified in the DD statement for the output tape. The OUTVOL= option specification overrides the volume serial, if any, that was specified in the DD statement for the tape.

  Specify the OUTVOL= option only if you are using deferred mounting.

INVOL Statement

INVOL *options* ;

The INVOL statement defines an input tape volume from which some or all files are to be copied to the output tape volume. The INVOL statement is not necessary if you are using only one input tape nor for the first of several input tapes. (Use the INDD= and INVOL= options of the PROC TAPECOPY statement instead.) When you are using several input tapes, use an INVOL statement for each tape after the first input tape.

The following options can appear in the INVOL statement. The options are listed alphabetically.

DSN | DSNAME='*physical-filename*'
: specifies the data set name of the first file on the current input tape. You must use this option when both of the following conditions are true:

  - ☐ The data set name specified in the DD statement is incorrect or missing.
  - ☐ LABEL=(,SL) is specified (or implied by default) in the input tape volume DD statement.

  You typically use this option when one of the following conditions is true:

  - ☐ The DD statement for the input tape specifies deferred mounting.
  - ☐ You are reusing a DD statement (and tape drive); that is, when the fileref is the same but you want another standard labeled tape volume on the same unit. LABEL=(,SL) should be specified or implied by default, and the data set name cannot be the same as that on the previous tape that was used with this fileref.

INDD=*DDname*
: specifies the DDname that is referenced in the JCL DD statement for the current input tape. The default INDD= option value is the DDname that is already in effect for the previous input tape volume, as specified in the PROC TAPECOPY statement or in the last INVOL statement.

INVOL=*volume-serial*
: specifies the volume serial of the current input tape. Use the INVOL= option when the JCL DD statement for the input tape specifies deferred mounting (as described

in “PROC TAPECOPY Statement” on page 275), or when you are reusing a DD statement (and tape drive); that is, when the DDname is the same, but you want a different tape volume on the same unit.

NL
: specifies that the input tape is nonlabeled; if LABEL=(,SL) or LABEL=(,BLP) has been specified in the DD statement for the input tape and the tape is actually standard labeled, specifying the NL option causes the tape to be treated as if it were nonlabeled. In this case, any file numbers that are specified in FILES statements must be physical file numbers, not logical file numbers.

NORER
: tells SAS not to specify the "reduced error recovery for tape devices" feature of the operating environment for the input tape volume. When this option is specified, some tapes of marginal quality can be read successfully by PROC TAPECOPY because the error recovery procedures are more extensive. If NORER is specified in the PROC TAPECOPY statement, then NORER is in effect for all input tape volumes and INVOL statements.

SL
: specifies that the input tape is standard labeled. If you specify LABEL=(,BLP) in the DD statement for the input tape and specify SL in the INVOL statement, PROC TAPECOPY verifies that the tape is standard labeled. Do not specify SL unless the tape is actually standard labeled.

Note: If you do not specify NL or SL in the INVOL statement, the actual input tape label type determines whether PROC TAPECOPY treats the tape as nonlabeled or standard labeled, even when LABEL=(,BLP) is specified in the DD statement. △

FILES Statement

FILES *file-numbers*;

When you want to copy particular files from an input tape, use the FILES statement to specify which files you want to copy. Use as many FILES statements as you want. Give the physical file numbers for nonlabeled tapes or for labeled tapes that are being treated as nonlabeled. Give the logical file numbers for standard labeled tapes that are not being treated as nonlabeled, even when the output tape volume is to be nonlabeled (LABEL=NL). FILE is an alias for the FILES statement.

If you are using only one input tape, the FILES statement(s) can directly follow the PROC TAPECOPY statement. When you use several input tape volumes, follow each INVOL statement with the associated FILES statement or statements.

Specifying Individual Files File numbers in a FILES statement can be specified in any order. For example, you might want to copy file 5 and then file 2 and then file 1, as in the following example:

```
proc tapecopy;
   files 5 2;
   files 1;
run;
```

Specifying a Range You can specify a range of files by putting a dash between two file numbers, as in the following example:

```
proc tapecopy;
   files 1-7;
```

```
run;
```

In a range, the second number must be greater than the first. The keyword EOV (end of volume) can be used as the last file in a range. PROC TAPECOPY copies all files on the input tape until the end of the volume (in most cases, a double tapemark). On a nonlabeled tape, you can copy files from the input tape beyond the double tapemark by specifying the physical file number, counting tapemarks as usual. If another double tapemark exists on the input tape volume, you can then specify EOV in another range.

Examples

Example 1: Copying Standard Labeled to Standard Labeled The following job copies a standard labeled tape (volume serial XXXXXX) to another standard labeled tape (volume serial YYYYYY).

```
//jobname  JOB account,name
//  EXEC SAS
//VOLIN DD UNIT=TAPE,DISP=OLD,
//  VOL=SER=XXXXXX,LABEL=(,SL),
//  DSN=first-dsname-on-tape
//VOLOUT DD UNIT=TAPE,DISP=(,KEEP),
//  VOL=SER=YYYYYY,LABEL=(,SL)
//SYSIN DD *
   proc tapecopy;
   run;
/*
//
```

After PROC TAPECOPY executes, the output tape volume is labeled YYYYYY.

If LABEL=(,BLP) had been specified in the input tape DD statement (VOLIN), then it would not have been necessary to use the DSN= option. Because some installations do not permit the BLP label type specification, and because no volume label checking is performed when it is specified, it is recommended that you specify (or allow to default) LABEL=(,SL).

The specification of LABEL=(,SL) in the output tape DD statement (VOLOUT) causes the operating environment to check the volume label when a tape volume is mounted on the tape drive. The operating environment ensures that a tape with volume serial YYYYYY is mounted. However, if the tape with external volume label YYYYYY were, in fact, internally labeled something other than YYYYYY, PROC TAPECOPY would fail. In this case, you would have to specify LABEL=(,BLP) or else give the actual internal volume serial in the output tape DD statement. If the output tape is not labeled internally, you can specify LABEL=(,NL) or LABEL=(,BLP).

Example 2: Copying Standard Labeled to Nonlabeled The next job copies a standard labeled tape with volume serial TAPEIN to a nonlabeled tape, FCSTP1. After the job is executed, the output tape volume is still a nonlabeled tape, presumably with only an external volume label of FCSTP1. You must specify LABEL=NL in the PROC TAPECOPY statement; otherwise, the procedure defaults to LABEL=SL because the first (and only) input tape volume is standard labeled.

```
//jobname  JOB account,name
//  EXEC SAS
//VOLIN DD UNIT=TAPE,DISP=OLD,VOL=SER=TAPEIN,
//         LABEL=(,BLP)
//VOLOUT DD UNIT=TAPE,DISP=(,KEEP),VOL=SER=FCSTP1,
//         LABEL=(,NL)
```

```
//SYSIN DD *
   proc tapecopy label=nl;
   run;
/*
//
```

Example 3: Copying Nonlabeled to Nonlabeled The following job copies a nonlabeled tape with volume serial QDR123 to a nonlabeled, 1600 bpi tape, SLXATK:

```
//jobname  JOB account,name
//  EXEC SAS
//INTAPE DD UNIT=TAPE,DISP=OLD,VOL=SER=QDR123,
//          LABEL=(,NL)
//OUTTAPE DD UNIT=2927-3,DISP=(,KEEP),
//           VOL=SER=SLXATK,LABEL=(,NL)
//SYSIN DD *
   proc tapecopy indd=intape outdd=outtape
        den=1600;
   run;
/*
//
```

Example 4: Copying Multiple Files from One Input Tape This next job copies the first seven files from the standard labeled input tape U02746 plus four files from the standard labeled input tape T13794 to an initially nonlabeled output tape with volume serial MINI01. After the procedure is executed, the output tape is standard labeled and has a volume serial of U02746, as specified by the COPYVOLSER option.

```
//jobname  JOB account,name
//  EXEC SAS
//TAPI1 DD DISP=SHR,UNIT=TAPE,
//   VOL=SER=U02746,LABEL=(,SL),
//   DSN=first-file-dsname
//TAPI2 DD UNIT=(TAPE,,DEFER)
//OUTDDN DD DISP=(,KEEP),UNIT=TAPE,VOL=SER=MINI01,
//   LABEL=(,NL)
//SYSIN DD *
   proc tapecopy outdd=outddn indd=tapi1
        copyvolser;
      files 3 2 1;
      invol indd=tapi2 invol=t13794
         dsn='first-dsname-on-this-tape ';
      file 3;
      invol indd=tapi1;
      files 5-7 4;
      invol indd=tapi2;
      files 2 4 1;
   run;
/*
//
```

Example 5: Copying Multiple Files from Multiple Input Tapes The next job copies several files from several input tape volumes to one output tape volume:

```
//REARRNGE JOB account,name
//  EXEC SAS
```

```
//DEN2IN DD UNIT=(2927-4,,DEFER),LABEL=(,BLP)
//DEN3IN DD UNIT=(2927-3,,DEFER),LABEL=(,SL)
//TAPE1 DD UNIT=TAPE,DISP=SHR,VOL=SER=XR8475,
//   LABEL=(,BLP)
//TAPE2 DD UNIT=TAPE,DISP=OLD,VOL=SER=BKT023,
//  DSN=first-file-dsname
//OUTPUT DD UNIT=(3400-5,,DEFER),DISP=(,KEEP)
//SYSIN DD *
   proc tapecopy label=sl den=6250 nolist
        outdd=output outvol=histpe;
      invol indd=den2in invol=ptftp0;
      files 2-4 8-eov 7 6;
      invol indd=tape1;
      files 5 7 9-eov;
      invol indd=tape2;
      files 4 5 1;
      invol indd=den3in invol=s03768
         dsn='xrt.bkt120.g0081v00';
      files 1-6 22-34;
      invol invol=so3760 dsn='t.bkt120.g0023v00';
      files 4 5 6 9;
      invol indd=tape2;
      files 7-eov;
   run;
/*
//
```

TAPELABEL

Writes the label information of an IBM standard-labeled tape volume to the SAS procedure output file

OS/390 specifics: all

Syntax

PROC TAPELABEL <*options*>;

Details

The procedure writes information from the tape label, including the data set name, DCB information, and data set history, to the SAS procedure output file.

Each tape volume must have a DDname allocated for it before that volume can be read by the TAPELABEL procedure. Multiple tape volumes can be read in one PROC TAPELABEL statement, using a list of DDnames in the DDNAME= option, as shown below. At some installations, you may need to specify the data set name of the first file on the tape volume as the first entry in your list of DDnames. This is necessary if you cannot use LABEL=(,BLP), which is restricted at many sites.

PROC TAPELABEL Statement

PROC TAPELABEL <*options*>;

The following options can be specified in the PROC TAPELABEL statement:

DCBDEVT=128
: enables PROC TAPELABEL to process Fujitsu F6470 tape cartridges.

DDNAME=(*DDname-1...DDname-n*)
: specifies the *DDname* of the tape volume that you want to process. More than one DDname can be specified, with blanks spaces delimiting the list. If you specify only one DDname, you can omit the parentheses.

  If DDNAME= is omitted, the default DDname is TAPE.

DUMP
: sends to output the first 80 bytes in the first 10 blocks of each data set on the tape.

NOTRAP813
: tells the TAPELABEL procedure not to trap 813-04 abends. When you use LABEL=(,SL) to access an IBM standard labeled tape, this option prevents you from reading the tape unless you specify the data set name of the first file on the tape volume.

PAGE
: begins the output for each tape volume on a new page.

Output

For each file on a tape volume, TAPELABEL generates the following information:

- FILE NUMBER, the file sequence number
- DSNAME, the data set name
- RECFM, the record format
- LRECL, the logical record length
- BLKSIZE, the block size
- BLOCK COUNT, the number of blocks in the file (from the trailer label)
- EST FEET, the *estimated* length of the file in feet (assumes all blocks=BLKSIZE)
- CREATED, the file creation date
- EXPIRES, the file expiration date
- CREATED BY JOB NAME STEPNAME, the job and step names of the job that created the file
- TRTCH, the track recording technique
- DEN, the file recording density code
- PSWD, the file protection indicator
- UHL, the number of user header labels
- UTL, the number of user trailer labels.

TAPELABEL also lists the sum of the estimated file lengths.

Note: On an IBM standard tape label, only 17 characters are available for the data set name. If a longer name is specified in the JCL when the data set is created, only the rightmost 17 characters are used. PROC TAPELABEL displays what is stored in the tape's header label. Some tape management systems catalog data sets by the full name specified in the JCL and therefore require you to specify the full name when you access the data set. △

Example

The following job generates the label information for all files on the MVSV8 tape volume allocated to the DDname OURTAPE:

```
//jobname JOB acct,name
/*JOBPARM FETCH
//TLABEL   EXEC SAS
//OURTAPE  DD   UNIT=TAPE,DISP=OLD,VOL=SER=MVSV8
//SYSIN DD *
   proc tapelabel ddname=ourtape;
   run;
/*
//
```

Output 16.15 on page 283 shows the results.

Output 16.15 Output from the TAPELABEL Procedure

```
                                                    The SAS System

                                             TAPE LIST FOR DDNAME -  OURTAPE
 CONTENTS OF TAPE VOLUME -  OS390T                                                                    OWNER -

  FILE                                           BLOCK   EST       CUM                         CREATED BY
  NUMBER DSNAME          RECFM  LRECL  BLKSIZE  COUNT    FEET    FEET CREATED   EXPIRES  JOB NAME STEPNAME TRTCH DEN PSWD UHL UTL

    1  SAS.SASROOT        FB      80     6160    175      3.6     3.6 12MAR1999 0000000  E70S701 /GO              5   NO   0   0
    2  SAS.V186.@P@BA$H   FB   6144      6144     77      1.6     5.2 12MAR1999 0000000  E70S701 /GO              5   NO   0   0
    3  SAS.V186.EMO1CLR   U        0     6164    633     12.9    18.0 12MAR1999 0000000  E70S701 /GO              5   NO   0   0
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Companion for the OS/390 Environment, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Companion for the OS/390® Environment, Verison 8

1–58025–523–X

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999