CHAPTER

14

The DATASETS Procedure

Overview

The DATASETS procedure is a utility procedure that manages your SAS files. With PROC DATASETS, you can

- copy SAS files from one SAS library to another
- rename SAS files
- repair SAS files
- delete SAS files
- list the SAS files that are contained in a SAS library
- list the attributes of a SAS data set, information such as the date the data were last modified, whether the data are compressed, whether the data are indexed, and so on
- manipulate passwords on SAS files
- append SAS data sets
- modify attributes of SAS data sets and variables within the data sets
- create and delete indexes on SAS data sets
- create and delete integrity constraints on SAS data sets.

The following PROC DATASETS step

- copies all data sets from the CONTROL library to the HEALTH library
- lists the contents of the HEALTH library
- deletes the SYNDROME data set from the HEALTH library
- changes the name of the PRENAT data set to INFANT.

The SAS log is shown in Output 14.1 on page 331.

```
libname control 'SAS-data-library-1';
libname health 'SAS-data-library-2';


proc datasets memtype=data;
   copy in=control out=health;
```

```
run;

proc datasets library=health details;
   delete syndrome;
   change prenat=infant;
run;
quit;
```

Output 14.1 Log of PROC DATASETS Activity (UNIX Environment)

```
16  proc datasets library=health details;
                                              -----Directory-----

                             Libref:            HEALTH
                             Engine:            V7
                             Filefmt:           7
                             Physical Name:     external-file
                             File Name:         external-file
                             Inode Number:      718930053
                             Access Permission: rwxr-xr-x
                             Owner Name:        UNIX-userid
                             File Size (bytes): 2048

                                     Obs, Entries                                 File
                 #  Name      Memtype   or Indexes   Vars  Label                   size  Last modified
                ---------------------------------------------------------------------------------------------
                 1  ALL       DATA          23        17                          17408  Thu Aug 28 15:53:10 1997
                 2  BODYFAT   DATA           1         2                          12288  Thu Aug 28 15:53:10 1997
                 3  CONFOUND  DATA           8         4                          12288  Thu Aug 28 15:53:10 1997
                 4  CORONARY  DATA          39         4                          12288  Thu Aug 28 15:53:10 1997
                 5  DRUG1     DATA           6         2   JAN95 Data             12288  Thu Aug 28 15:53:10 1997
                 6  DRUG2     DATA          13         2   MAY95 Data             12288  Thu Aug 28 15:53:11 1997
                 7  DRUG3     DATA          11         2   JUL95 Data             12288  Thu Aug 28 15:53:11 1997
                 8  DRUG4     DATA           7         2   JAN92 Data             12288  Thu Aug 28 15:53:11 1997
                 9  DRUG5     DATA           1         2   JUL92 Data             12288  Thu Aug 28 15:53:11 1997
                10  GROUP     DATA         148        11                          32768  Thu Aug 28 15:53:12 1997
                11  MLSCL     DATA          32         4   Multiple Sclerosis Data 12288  Thu Aug 28 15:53:12 1997
                12  NAMES     DATA           7         4                          12288  Thu Aug 28 15:53:12 1997
                13  OXYGEN    DATA          31         7                          13312  Thu Aug 28 15:53:13 1997
                14  PERSONL   DATA         148        11                          32768  Thu Aug 28 15:53:13 1997
                15  PHARM     DATA           6         3   Sugar Study            12288  Thu Aug 28 15:53:13 1997
                16  POINTS    DATA           6         6                          12288  Thu Aug 28 15:53:13 1997
                17  PRENAT    DATA         149         6                          23552  Thu Aug 28 15:53:14 1997
                18  RESULTS   DATA          10         5                          12288  Thu Aug 28 15:53:14 1997
                19  SLEEP     DATA         108         6                          16384  Thu Aug 28 15:53:14 1997
                20  SYNDROME  DATA          46         8                          16384  Thu Aug 28 15:53:15 1997
                21  TENSION   DATA           4         3                          12288  Thu Aug 28 15:53:15 1997
                22  TEST2     DATA          15         5                          12288  Thu Aug 28 15:53:15 1997
                23  TRAIN     DATA           7         2                          12288  Thu Aug 28 15:53:15 1997
                24  VISION    DATA          16         3                          12288  Thu Aug 28 15:53:16 1997
                25  WEIGHT    DATA          83        13   California Results     26624  Thu Aug 28 15:53:16 1997
                26  WGHT      DATA          83        13   California Results     26624  Thu Aug 28 15:53:16 1997
17     delete syndrome;
18     change prenat=infant;
19  run;
NOTE: Deleting HEALTH.SYNDROME (memtype=DATA).
NOTE: Changing the name HEALTH.PRENAT to HEALTH.INFANT (memtype=DATA).
20  quit;
```

Notes

- Although the DATASETS procedure can perform some operations on catalogs, generally the CATALOG procedure is the best utility to use for managing catalogs. For documentation of PROC CATALOG, see Chapter 6, “The CATALOG Procedure,” on page 155. See Chapter 3, "Statements with the Same Function in Multiple Procedures," for details. You can also use any global statements as well. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," for a list.
  .

- The term *member* often appears as a synonym for *SAS file*. In addition, if you are unfamiliar with SAS files and SAS libraries, see the chapter on SAS files in *SAS Language Reference: Concepts*.
- PROC DATASETS cannot work with sequential data libraries.

Procedure Syntax

Tip: Supports RUN-group processing.

Tip: Supports the Output Delivery System. (See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures" for information on the Output Delivery System.)

Reminder: You can use any global statements as well. See Chapter 3, "Statements with the Same Function in Multiple Procedures," for details. You can also use any global statements as well. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," for a list.

PROC DATASETS <*option(s)*>;

AGE *current-name related-SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*>>;

APPEND BASE=<*libref.*>*SAS-data-set*
<APPENDVER=V6>
<DATA=<*libref.*>*SAS-data-set*>
<FORCE>;

AUDIT *SAS-file-name* <*SAS-password*>;
INITIATE;
<**LOG** <BEFORE_IMAGE=YES|NO>
<DATA_IMAGE=YES|NO>
<ERROR_IMAGE=YES|NO>;>
<**USER_VAR**=*variable-1* <*... variable-n*>;>
<**SUSPEND|RESUME|TERMINATE**;>

CHANGE *old-name-1=new-name-1*
<*…old-name-n=new-name-n* >
</ <ALTER=*alter-password*>
<CONSTRAINT=YES|NO>
<GENNUM=ALL|*integer*>
<INDEX=YES|NO>
<MEMTYPE=*mtype*>>;

CONTENTS<*option(s)*>;

COPY OUT=*libref-1*
<CLONE|NOCLONE>
<IN=*libref-2*>
<MEMTYPE=(*mtype(s)*)>
<MOVE <ALTER=*alter-password*>>;
EXCLUDE *SAS-file(s)* < / MEMTYPE=*mtype*>;
SELECT *SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE= *mtype*>>;

DELETE *SAS-file(s)*
</ <ALTER=*alter-password*>

<GENNUM=ALL|HIST|REVERT|*integer*>
<MEMTYPE=*mtype*>>;

EXCHANGE *name-1=other-name-1*
<*…name-n=other-name-n*>
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*> >;

MODIFY *SAS-file* <(*file-option(s)*)>
</ <GENNUM=ALL|HIST|REVERT|*integer*>
<MEMTYPE=*mtype*>>;
FORMAT *variable-list-1* <*format-1*>
<*…variable-list-n* <*format-n*>>;
IC CREATE <*constraint-name*>=*constraint*
<NOT NULL | CHECK(WHERE-*clause*)
| PRIMARY KEY | UNIQUE
| FOREIGN KEY(*variable*)
REFERENCES *SAS-data-set*>
<MESSAGE='message-string'>;
IC DELETE *constraint-name(s)*| _ALL_;
IC REACTIVATE *foreign-key-name* REFERENCES *libref*;
INDEX CENTILES *index(s)*
</ <REFRESH>
<UPDATECENTILES=
ALWAYS|NEVER|*integer*> >;
INDEX CREATE *index-specification(s)*
</ <NOMISS>
<UNIQUE>
<UPDATECENTILES=
ALWAYS|NEVER|*integer*>>;
INDEX DELETE *index(s)* | _ALL_;
INFORMAT *variable-list-1* <*informat-1*>
<*…variable-list-n* <*informat-n*>>;
LABEL *variable-1*=<*'label-1'*|' '>
<*…variable-n*=<*'label-n'*|' ' >>;
RENAME *old-name-1=new-name-1*
<*…old-name-n=new-name-n*>;

REPAIR *SAS-file(s)*
</ <ALTER=*alter-password*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

SAVE *SAS-file(s)* </ MEMTYPE=*mtype*>;

PROC DATASETS Statement

PROC DATASETS <*option(s)*>;

To do this	Use this option
Specify the procedure input library	LIBRARY=
Provide alter access to any alter-protected SAS file in the SAS data library	ALTER=
Include information in the output about the number of observations, number of variables, and data set labels	DETAILS|NODETAILS
Force a RUN group to execute even when there are errors	FORCE
Force an append operation	FORCE
Delete SAS files	KILL
Restrict processing to a certain type of SAS file	MEMTYPE=
Suppress the printing of the directory	NOLIST
Suppress error processing	NOWARN
Provide read, write, or alter access	PW=
Provide read access	READ=

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files in the SAS data library.

See also: "Using Passwords with the DATASETS Procedure" on page 377

DETAILS|NODETAILS
determines whether the output includes the following columns:

`Obs or Entries`
gives the number of observations for SAS files of type DATA and VIEW and the number of entries for type CATALOG. If the SAS System cannot determine the number of observations in a SAS data set, the value in this column is a period (.). The value for type CATALOG is the total number of entries. For other types, this column is blank.

`Vars`
gives the number of variables for types DATA and VIEW. If the SAS System cannot determine the number of variables in the SAS data set, the value in this column is a period (.). For other types, this column is blank.

`Label`
contains the label associated with the SAS data set. This column prints a label only for the type DATA.

The DETAILS option requires read access to all read-protected SAS files in the SAS data library. If you do not supply the read password, the directory listing contains missing values for the columns produced by the DETAILS option.

Default: DETAILS system option setting

Tip: If you are using the SAS windowing environment and specify the DETAILS option for a library that contains read-protected SAS files, a requestor window prompts you for each read password that you do not specify in the PROC DATASETS statement. Therefore, you may want to assign the same read password to all SAS files in the same SAS data library.

Featured in: Example 1 on page 391

FORCE
performs two separate actions:

- forces a RUN group to execute even if errors are present in one or more statements in the RUN group. See "Procedure Execution" on page 375 for a discussion of RUN-group processing and error handling.
- forces all APPEND statements to concatenate two data sets even when the variables in the data sets are not exactly the same. The APPEND statement drops the extra variables and issues a warning message. Without the FORCE option, the procedure issues an error message and stops processing if you try to perform an append operation with two SAS data sets whose variables are not exactly the same. Refer to "APPEND Statement" on page 339 for more information on the FORCE option.

KILL
deletes *all* SAS files in the SAS data library that are available for processing. The MEMTYPE= option subsets the member types that the statement deletes.

CAUTION:
The KILL option deletes the SAS files immediately after you submit the statement. △

LIBRARY=*libref*
names the library that the procedure processes. This library is the *procedure input library*.

Aliases: DDNAME=, DD=, LIB=

Default: WORK or USER. See "Temporary and Permanent SAS Data Sets" on page 16 for more information on the WORK and USER libraries.

Restriction: A SAS library that is accessed via a sequential engine (such as a tape format engine) cannot be specified as the value of the LIBRARY= option.

Featured in: Example 1 on page 391

MEMTYPE=(*mtype(s)*)
restricts processing to one or more member types and restricts the listing of the data library directory to SAS files of the specified member types. For example, the following PROC DATASETS statement limits processing to SAS data sets in the default data library and limits the directory listing in the SAS log to SAS files of member type DATA:

```
proc datasets memtype=data;
```

Aliases: MTYPE=, MT=

Default: ALL

See also: "Restricting Member Types Available for Processing" on page 378

NODETAILS
See the description of DETAILS on page 335.

NOLIST
suppresses in the SAS log the printing of the directory of the SAS files that are available for processing.

Featured in: Example 3 on page 397

NOWARN
suppresses the error processing that occurs when a SAS file that is specified in a SAVE, CHANGE, EXCHANGE, or COPY statement or listed as the first SAS file in an AGE statement is not in the procedure input library. When an error occurs and the NOWARN option is in effect, PROC DATASETS continues processing that RUN

group. If NOWARN is not in effect, PROC DATASETS stops processing that RUN group.

PW= ***password***
provides the password for any protected SAS files in the SAS data library. PW= can act as an alias for READ=, WRITE=, or ALTER=.

See also: "Using Passwords with the DATASETS Procedure" on page 377

READ=*read-password*
provides the read-password for any read-protected SAS files in the SAS data library.

See also: "Using Passwords with the DATASETS Procedure" on page 377

AGE Statement

Renames a group of related SAS files in a library.

Featured in: Example 6 on page 404

AGE *current-name related-SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*>>;

Required Arguments

current-name
is a SAS file that the procedure renames. *current-name* receives the name of the first name in *related-SAS-file(s)*.

related-SAS-file(s)
is one or more SAS files in the SAS data library.

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files named in the AGE statement. Because an AGE statement renames and deletes SAS files, you need alter access to use the AGE statement. You can use the ALTER= option in parentheses after the name of each SAS file or after a forward slash.

See also: "Using Passwords with the DATASETS Procedure" on page 377

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*). All the SAS files you name in the AGE statement must be the same member type.

Aliases: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is DATA.

See also: "Restricting Member Types Available for Processing" on page 378

Details

- The AGE statement renames *current-name* to the name of the first name in *related-SAS-file(s)*, renames the first name in *related-SAS-file(s)* to the second name in *related-SAS-file(s)*, and so on until it changes the name of the next-to-last SAS file in *related-SAS-file(s)* to the last name in *related-SAS-file(s)*. The AGE statement then deletes the last file in *related-SAS-file(s)*.
- If the first SAS file named in the AGE statement does not exist in the SAS data library, PROC DATASETS stops processing the RUN group containing the AGE statement and issues an error message. The AGE statement does not age any of the *related-SAS-file(s)*. To override this behavior, use the NOWARN option in the PROC DATASETS statement.

  If one of the *related-SAS-file(s)* does not exist, the procedure prints a warning message to the SAS log but continues to age the SAS files that it can.
- If you age a data set that has an index, the index continues to correspond to the data set.
- You can age only entire generation groups. For example, if A and B are generation groups, then the following statement deletes generation group B and ages (renames) generation group A to the name B:

  ```
  age a b;
  ```

  For example, suppose the generation group A has 3 historical generations and the generation group B has 2 historical generations. Then aging A to B has this effect:

Old Name	GENNUM	New Name	GENNUM
A		B	
A	1	B	1
A	2	B	2
A	3	B	3
B		is deleted	

B	1	is deleted
B	2	is deleted

APPEND Statement

Adds the observations from one SAS data set to the end of another SAS data set.

Reminder: You can use data set options with the BASE= and DATA= options. See Chapter 3, "Statements with the Same Function in Multiple Procedures," for details. You can also use any global statements as well. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," for a list.

Restriction: The BASE= data set must be a member of a SAS library that supports UPDATE access. The TAPE engine and the XPORT engine are examples of engines that do not support UPDATE access.

Restriction: If the BASE= data set is accessed through a SAS server and if no other user has the data set open at the time the APPEND statement begins processing, the BASE= data set defaults to CNTLLEV=MEMBER. When this happens, no other user can update the file while the data set is processed.

Featured in: Example 5 on page 403

APPEND BASE=<*libref.*>*SAS-data-set* <*options*> ;

Required Arguments

BASE=<*libref.*> *SAS-data-set*
names the data set to which you want to add observations.

libref
specifies the library that contains the SAS data set. If you omit *libref*, the default is the libref for the procedure input library. If you are using PROC APPEND, the default for *libref* is either WORK or USER.

SAS-data-set
names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it creates a new data set in the library. In other words, you can use the APPEND statement to create a data set by specifying a new data set name in the BASE= argument.

The BASE= data set is the current SAS data set after all append operations regardless of whether you are creating a new data set or appending to an existing data set.

Alias: OUT=

Featured in: Example 5 on page 403

Options

APPENDVER=V6
uses the Version 6 behavior for appending observations to the BASE= data set. You must specify V6.

See also: "Appending to an Indexed Data Set" on page 341

DATA=<*libref.*> *SAS-data-set*
names the SAS data set containing observations that you want to append to the end of the SAS data set specified in the BASE= argument.

libref
specifies the library that contains the SAS data set. If you omit *libref*, the default is the libref for the procedure input library. The DATA= data set can be from any SAS data library, but you must use the two-level name if the data set resides in a library other than the procedure input library.

SAS-data-set
names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it stops processing.

Alias: NEW=

Default: the most recently created SAS data set, from any SAS data library

Featured in: Example 5 on page 403

FORCE
forces the APPEND statement to concatenate data sets when the DATA= data set contains variables that either

- are not in the BASE= data set
- do not have the same type as the variables in the BASE= data set
- are longer than the variables in the BASE= data set.

See also: "Appending to Data Sets with Different Variables" on page 342 and "Appending to Data Sets That Contain Variables with Different Attributes" on page 342

Featured in: Example 5 on page 403

Restricting the Observations That Are Appended

You can use the WHERE= data set option with the DATA= data set to restrict the observations that are appended. Likewise, you can use the WHERE statement to restrict the observations from the DATA= data set. The WHERE statement has no affect on the BASE= data set. If you use the WHERE= data set option with the BASE= data set, WHERE= has no affect.

Choosing between the SET Statement and the APPEND Statement

If you use the SET statement in a DATA step to concatenate two data sets, the SAS System must process all the observations in both data sets to create a new one. The APPEND statement bypasses the processing of data in the original data set and adds new observations directly to the end of the original data set. Using the APPEND statement can be more efficient than using a SET statement if

- the BASE= data set is large
- all variables in the BASE= data set have the same length and type as the variables in the DATA= data set and if all variables exist in both data sets.

  Note: You can use the CONTENTS statement to see the variable lengths and types. △

The APPEND statement is especially useful if you frequently add observations to a SAS data set (for example, in production programs that are constantly appending data to a journal-type data set).

Appending Password-Protected SAS Data Sets

In order to use the APPEND statement, you need read access to the DATA= data set and write access to the BASE= data set. To gain access, use the READ= and WRITE= data set options in the APPEND statement the way you would use them in any other SAS statement, in parentheses immediately after the data set name. When you are appending password-protected data sets, remember the following guidelines:

- If you do not give the read password for the DATA= data set in the APPEND statement, by default the procedure looks for the read password for the DATA= data set in the PROC DATASETS statement. However, the procedure does not look for the write password for the BASE= data set in the PROC DATASETS statement. Therefore, you must specify the write password for the BASE= data set in the APPEND statement.
- If the BASE= data set is read-protected only, you must specify its read password in the APPEND statement.

Appending to a Compressed Data Set

You can concatenate compressed SAS data sets. Either or both of the BASE= and DATA= data sets can be compressed. If the BASE= data set allows the reuse of space from deleted observations, the APPEND statement may insert the observations into the middle of the BASE= data set to make use of available space.

For information on the COMPRESS= and REUSE= data set and system options, see *SAS Language: Reference*.

Appending to an Indexed Data Set

Beginning with Version 7, the behavior of appending to an indexed data set has changed to improve performance.

- In Version 6, when you appended to an indexed data set, the index was updated for each added observation. Index updates tend to be random; therefore, disk I/O could have been high.
- Currently, SAS does not update the index until all observations are added to the data set. After the append, SAS internally sorts the observations and inserts the data into the index in sequential order, which reduces most of the disk I/O and results in a faster append method.

The current method is used by default when the following requirements are met; otherwise, the Version 6 method is used:

- The BASE= data set is open for member-level locking.
- The BASE= data set does not contain referential integrity constraints.
- The BASE= data set is not accessed using the Cross Environment Data Access (CEDA) facility.
- The BASE= data set is not using a WHERE= data set option.

To display information in the SAS log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

Either a message displays if the fast-append method is in use or a message or messages display as to why the fast-append method is not in use.

The current append method initially adds observations to the BASE= data set regardless of the restrictions that are determined by the index. For example, a variable that has an index that was created with the UNIQUE option does not have its values

validated for uniqueness until the index is updated. Then, if a nonunique value is detected, the offending observation is deleted from the data set. This means that after observations are appended, some of them may subsequently be deleted.

For a simple example, consider that the BASE= data set has ten observations numbered from 1 to 10 with a UNIQUE index for the variable ID. You append a data set that contains five observations numbered from 1 to 5, and observations 3 and 4 both contain the same value for ID. The following occurs

1. After the observations are appended, the BASE= data set contains 15 observations numbered from 1 to 15.
2. SAS updates the index for ID, validates the values, and determines that observations 13 and 14 contain the same value for ID.
3. SAS deletes one of the observations from the BASE= data set, resulting in 14 observations that are numbered from 1 to 15. For example, observation 13 is deleted. Note that you cannot predict which observation will be deleted, because the internal sort may place either observation first. (In Version 6, you could predict that observation 13 would be added and observation 14 would be rejected.)

If you do not want the current behavior (which could result in deleted observations) or if you want to be able to predict which observations are appended, request the Version 6 append method by specifying the APPENDVER=V6 option:

```
proc data sets;
     append base=a data=b appendver=v6;
run;
```

Note: In Version 6, deleting the index and then recreating it after the append could improve performance. The current method may eliminate the need to do that. However, the performance depends on the nature of your data. △

Appending to Data Sets with Different Variables

If the DATA= data set contains variables that are not in the BASE= data set, use the FORCE option in the APPEND statement to force the concatenation of the two data sets. The APPEND statement drops the extra variables and issues a warning message.

If the BASE= data set contains a variable that is not in the DATA= data set, the APPEND statement concatenates the data sets, but the observations from the DATA= data set have a missing value for the variable that was not present in the DATA= data set. The FORCE option is not necessary in this case.

Appending to Data Sets That Contain Variables with Different Attributes

If a variable has different attributes in the BASE= data set than it does in the DATA= data set, the attributes in the BASE= data set prevail.

If the length of a variable is longer in the DATA= data set than in the BASE= data set, or if the same variable is a character variable in one data set and a numeric variable in the other, use the FORCE option. Using FORCE has these consequences:

- The length of the variables in the BASE= data set takes precedence. The SAS System truncates values from the DATA= data set to fit them into the length that is specified in the BASE= data set.
- The type of the variables in the BASE= data set takes precedence. The APPEND statement replaces values of the wrong type (all values for the variable in the DATA= data set) with missing values.

Appending Data Sets That Contain Integrity Constraints

If the DATA= data set contains integrity constraints and the BASE= data set does not exist, the APPEND statement copies both general and referential integrity constraints. If the BASE= data set exists, the APPEND action copies only observations.

Appending with Generation Groups

You can use the GENNUM= data set option to append to a specific generation file. Here are examples:

SAS Statements	Result

`proc datasets;`
`  append base=a`
`  data=b(gennum=2);`
	appends B (GENNUM=2) to A

`proc datasets;`
`  append base=a(gennum=2)`
`  data=b(gennum=2);`
	appends B (GENNUM=2) to A(GENNUM=2)

Using the APPEND Procedure Instead of the APPEND Statement

The only difference between the APPEND procedure and the APPEND statement in PROC DATASETS, is the default for *libref* in the BASE= and DATA= arguments. For PROC APPEND, the default is either WORK or USER. For the APPEND statement, the default is the libref of the procedure input library.

System Failures

If a system failure or some other type of interruption occurs while the procedure is executing, the append operation may not be successful; it is possible that not all, perhaps none, of the observations will be added to the BASE= data set. In addition, the BASE= data set may suffer damage. The APPEND operation performs an update in place, which means that it does not make a copy of the original data set before it begins to append observations.

AUDIT Statement

Initiates and controls event logging to an audit file.

AUDIT *SAS-file-name* <*SAS-password*>;
  INITIATE;
    <**LOG**< BEFORE_IMAGE=YES|NO>
      <DATA_IMAGE=YES|NO> <ERROR_IMAGE=YES|NO>;>
    <**USER_VAR**=*variable-1* <*... variable-n*>;>
  <**SUSPEND|RESUME|TERMINATE**;>

Required Arguments and Statements

SAS-file-name
specifies the SAS data file in the procedure input library that you want to audit.

INITIATE
creates an audit file that has the same name as the SAS data file and a data set type of AUDIT. In the initial release of the Audit Trail, the audit file documents additions, deletions, and updates to the SAS data file.

Options

SAS-password
specifies the password for the SAS data file, if one exists.

LOG
specifies the audit settings. The audit settings are

BEFORE_IMAGE=YES|NO
controls the storage of before-update record images.

DATA_IMAGE=YES|NO
controls the storage of after-update record images.

ERROR_IMAGE=YES|NO
controls the storage of unsuccessful after-update record images.

USER_VAR=*variable-1 < ... variable-n>*
defines optional variables to be logged in the audit file with each update to an observation. The syntax for defining variable formats is

USER_VAR=*variable-name-1* <$> <*length*> <LABEL='*variable-label*' >
<... *variable-name-n* <$> <*length*> <LABEL='*variable-label*' > >

where

variable-name
is a name for the variable.

$
indicates that the variable is a character variable.

length
specifies the length of the variable. If a length is not specified, the default is 8.

LABEL='*variable-label*'
specifies a label for the variable.

SUSPEND
suspends event logging to the audit file, but does not delete the audit file.

RESUME
resumes event logging to the audit file, if it was suspended.

TERMINATE
terminates event logging and deletes the audit file.

Creating an Audit File

The following example code shows you how to create the audit file MYLIB.MYFILE.AUDIT to log updates to the data file MYLIB.MYFILE.DATA, storing all available record images:

```
proc datasets library=MyLib;
   audit MyFile alter=MyPassword;
   initiate;
```

```
run;
```

The following example code creates the same audit file but stores only error record images:

```
proc datasets library=MyLib;
   audit MyFile alter=MyPassword;
   initiate;
   log data_image=NO before_image=NO;
run;
```

CHANGE Statement

Renames one or more SAS files in the same SAS data library.

Featured in: Example 1 on page 391

CHANGE*old-name-1=new-name-1*
<…*old-name-n=new-name-n* >
</ <ALTER=*alter-password*>
<GENNUM=ALL|*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

old-name=new-name
changes the name of a SAS file in the input data library. *old-name* must be the name of an existing SAS file in the input data library.

Featured in: Example 1 on page 391

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files named in the CHANGE statement. Because a CHANGE statement changes the names of SAS files, you need alter access to use the CHANGE statement. You can use the ALTER= option in parentheses after *new-name* or after a forward slash.

See also: "Using Passwords with the DATASETS Procedure" on page 377

GENNUM=ALL|*integer*
restricts processing to a single generation file or to the entire generation group. Valid values for GENNUM= are

ALL
refers to the base name and all generation files of a generation group.

postive integer
refers to an explicit generation file.

negative integer
refers to a relative generation file.
The following statements change the name of A(GENNUM=3) to B:

```
proc datasets;
   change A=B / gennum=3;

proc datasets;
   change A(gennum=3)=B;
```

The following CHANGE statement produces an error:

```
proc datasets;
   change A(gennum=3)=B(gennum=3);
```

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

See also: "Restricting Member Types Available for Processing" on page 378

Details

- The CHANGE statement changes names by the order that the *old-names* occur in the directory listing, not in the order that you list the changes in the CHANGE statement.
- If the *old-name* SAS file does not exist in the SAS data library, PROC DATASETS stops processing the RUN group containing the CHANGE statement and issues an error message. To override this behavior, use the NOWARN option in the PROC DATASETS statement.
- If you change the name of a data set that has an index, the index continues to correspond to the data set.
- The CHANGE statement can reference only a specific generation file or the entire generation group. Therefore, you cannot use HIST or REVERT with the GENNUM= option.

CONTENTS Statement

Describes the contents of one or more SAS data sets and prints the directory of the SAS data library.

Reminder: You can use data set options with the BASE= and DATA= options. See Chapter 3, "Statements with the Same Function in Multiple Procedures," for details. You can also use any global statements as well. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," for a list.

Featured in: Example 4 on page 399

CONTENTS<*option(s)*>;

To do this	Use this option
Specify the input data set	DATA=
Specify the name for an output data set	OUT=
Specify the name of an output data set to contain information about indexes and constraints	OUT2=
Include information in the output about the number of observations, number of variables, and data set labels	DETAILS|NODETAILS
Print a list of the SAS files in the SAS data library	DIRECTORY
Print the length of a variable's informat or format	FMTLEN
Restrict processing to one or more types of SAS files	MEMTYPE=
Suppress the printing of individual files	NODS
Suppress the printing of the output	NOPRINT
Print a list of the variables by their position in the data set	VARNUM
Print abbreviated output	SHORT
Print centiles information for indexed variables	CENTILES

Options

CENTILES
prints centiles information for indexed variables.

DATA=*SAS-file-specification*
specifies an entire library or a specific SAS data set within a library. *SAS-file-specification* can take one of the following forms:

<libref.>SAS-data-set
names one SAS data set to process. The default for *libref* is the libref of the procedure input library. For example, to obtain the contents of the SAS data set HTWT from the procedure input library, use the following CONTENTS statement:

```
contents data=HtWt;
```

To obtain the contents of a specific generation file, use the following CONTENTS statement:

```
contents data=HtWt(gennum=3);
```

*<libref.>*_ALL_
gives you information about all SAS data sets having the type or types specified by the MEMTYPE= option. *libref* refers to the SAS data library. The default for *libref* is the libref of the procedure input library.

- If you are using the _ALL_ keyword, you need read access to all read-protected SAS data sets in the SAS data library.
- DATA=_ALL_ automatically prints a listing of SAS files in the library.

Default: most recently created data set in your job or session, from any SAS data library

Tip: If you specify a read-protected data set in the DATA= option but do not give the read password, by default the procedure looks in the PROC DATASETS

statement for the read password. However, if you do not specify the DATA= option and the default data set (last one created in the session) is read protected, the procedure does not look in the PROC DATASETS statement for the read password.

Featured in: Example 4 on page 399

DETAILS|NODETAILS
includes three additional columns in the directory.

Default: first defaults to DETAILS option in PROC DATASETS statement, then to the DETAILS system option setting.

See also: description of these additional columns in "Options" in "PROC DATASETS Statement" on page 334

DIRECTORY
prints a list of all SAS files in the specified SAS data library.

FMTLEN
prints the length of the informat or format. If you do not specify a length for the informat or format when you associate it with a variable, the length does not appear in the output of the CONTENTS statement unless you use the FMTLEN option. The length also appears in the FORMATL or INFORML variable in the output data set.

MEMTYPE=(*mtype(s)*)
restricts processing to one or more member types. The CONTENTS statement produces output only for member types DATA, VIEW, and ALL, which includes DATA and VIEW.

MEMTYPE= in the CONTENTS statement differs from MEMTYPE= in most of the other statements in the DATASETS procedure in the following ways:

- A slash does not precede the option.
- You cannot enclose the MEMTYPE= option in parentheses to limit its effect to only the SAS file immediately preceding it.

Specifying the MEMTYPE= option in the PROC DATASETS statement affects the CONTENTS statement only if you specify the _ALL_ keyword in the DATA= option. For example, the following statements produce the contents of only the SAS data sets with member type DATA:

```
proc datasets memtype=data;
   contents data=_all_;
run;
```

Aliases: MT=, MTYPE=

Default: DATA

NODS
suppresses printing the contents of individual files when you specify _ALL_ in the DATA= option. The CONTENTS statement prints only the SAS data library directory. You cannot use the NODS option when you specify only one SAS data set in the DATA= option.

NODETAILS
See the description of DETAILS on page 348.

NOPRINT
suppresses printing the output of the CONTENTS statement.

OUT=*SAS-data-set*
names an output SAS data set.

Tip: OUT= does not suppress the printed output from the statement. If you want to suppress the printed output, you must use the NOPRINT option.

See also: "The OUT= Data Set" on page 385 for a description of the variables in the OUT= data set.

OUT2=*SAS-data-set*
names an output data set to contain information about indexes and integrity constraints.

See also: "The OUT2= Data Set" on page 389 for a description of the variables in the OUT2= data set.

SHORT
prints only the list of variable names, the index information, and the sort information for the SAS data set.

VARNUM
prints a second list of the variable names in the order of their logical position in the data set. By default, the CONTENTS statement lists the variables alphabetically. The physical position of the variable in the data set is engine-dependent.

Using the CONTENTS Procedure Instead of the CONTENTS Statement

The only difference between the CONTENTS procedure and the CONTENTS statement in PROC DATASETS is the default for *libref* in the DATA= option. For PROC CONTENTS, the default is either WORK or USER. For the CONTENTS statement, the default is the libref of the procedure input library.

COPY Statement

Copies all or some of the SAS files in a SAS data library.

Featured in: Example 1 on page 391

COPY OUT=*libref-1*
<CLONE|NOCLONE>
<CONSTRAINT=YES|NO>
<IN=*libref-2*>
<INDEX=YES|NO>
<MEMTYPE=(*mtype(s)*)>
<MOVE <ALTER=*alter-password*>> ;

Required Arguments

OUT=*libref*
names the SAS data library to copy SAS files to.

Aliases: OUTLIB= and OUTDD=

Featured in: Example 1 on page 391

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files that you are moving from one data library to another. Because the MOVE option deletes the SAS file from the original data library, you need alter access to move the SAS file.

See also: "Using Passwords with the DATASETS Procedure" on page 377

CLONE|NOCLONE
specifies whether to copy the following data set attributes:

- size of input/output buffers
- whether the data set is compressed
- whether free space is reused.

You specify these attributes with either data set options or SAS system options:

- BUFSIZE= value for the size of the input/output buffers
- COMPRESS= value for whether the data set is compressed
- REUSE= value for whether free space is reused.

For the BUFSIZE= attribute, Table 14.1 on page 350 summarizes how the COPY statement works:

Table 14.1 CLONE and the BUFSIZE= Attribute

If you use...	the COPY statement...
CLONE	uses the BUFSIZE= value from the input data set for the output data set.
NOCLONE	uses the current setting of the SAS system option BUFSIZE= for the output data set.
neither	determines the type of access method, sequential or random, used by the engine for the input data set and the engine for the output data set. If both engines use the same type of access, the COPY statement uses the BUFSIZE= value from the input data set for the output data set. If the engines do not use the same type of access, the COPY statement uses the setting of SAS system option BUFSIZE= for the output data set.

For the COMPRESS= and REUSE= attributes, Table 14.2 on page 350 summarizes how the COPY statement works:

Table 14.2 CLONE and the COMPRESS= and REUSE= Attributes

If you use...	the COPY statement...
CLONE	uses the values from the input data set for the output data set. If the engine for the input data set does not support the COMPRESS= or REUSE= attribute, the COPY statement uses the current setting of the corresponding SAS system option.
NOCLONE	uses the current setting of the SAS system options COMPRESS= or REUSE= for the output data set.
neither	defaults to CLONE.

CONSTRAINT=YES|NO
specifies whether to copy all integrity constraints when copying a data set.

Default: NO

IN=*libref*
names the SAS data library containing SAS files to copy.

Aliases: INLIB= and INDD=

Default: the libref of the procedure input library

INDEX=YES|NO
specifies whether to copy all indexes for a data set when copying the data set to another SAS data library.

Default: YES

MEMTYPE=(*mtype(s)*)
restricts processing to one or more member types.

Aliases: MT=, MTYPE=

Default: If you omit MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

See also:

"Specifying Member Types When Copying or Moving SAS Files" on page 351

"Member Types" on page 379

Featured in: Example 1 on page 391

MOVE
moves SAS files from the input data library (named with the IN= option) to the output data library (named with the OUT= option) and deletes the original files from the input data library.

Restriction: The MOVE option can be used to delete a member of a SAS library only if the IN= engine supports the deletion of tables. A tape format engine does not support table deletion. If you use a tape format engine, SAS suppresses the MOVE operation and prints a warning.

Featured in: Example 1 on page 391

NOCLONE
See the description of CLONE on page 350.

Copying an Entire Library

To copy an entire SAS data library, simply specify an input data library and an output data library. For example, the following statements copy all the SAS files in the SOURCE data library into the DEST data library:

```
proc datasets library=source;
   copy out=dest;
run;
```

Copying Selected SAS Files

To copy selected SAS files, use a SELECT or EXCLUDE statement. For more discussion of using the COPY statement with a SELECT or an EXCLUDE statement, see "Specifying Member Types When Copying or Moving SAS Files" on page 351 and see Example 1 on page 391 for an example.

Specifying Member Types When Copying or Moving SAS Files

The MEMTYPE= option in the COPY statement differs from the MEMTYPE= option in other statements in the procedure in several ways:

- A slash does not precede the option.
- You cannot enclose the MEMTYPE= option in parentheses to limit its effect to the member immediately preceding it.
- The SELECT and EXCLUDE statements and the IN= option (in the COPY statement) affect the behavior of the MEMTYPE= option in the COPY statement according to the following rules:

  1. MEMTYPE= in a SELECT or EXCLUDE statement takes precedence over the MEMTYPE= option in the COPY statement. The following statements copy only VISION.CATALOG and NUTR.DATA from the default data library to the DEST data library; the MEMTYPE= value in the first SELECT statement overrides the MEMTYPE= value in the COPY statement.

```
proc datasets;
    copy out=dest memtype=data;
       select vision(memtype=catalog)
              nutr;
 run;
```

  2. If you do not use the IN= option, or you use it to specify the library that happens to be the procedure input library, the value of the MEMTYPE= option in the PROC DATASETS statement limits the types of SAS files that are available for processing. The procedure uses the order of precedence described in rule 1 to further subset the types available for copying. The following statements do not copy any members from the default data library to the DEST data library; instead, the procedure issues an error message because the MEMTYPE= value specified in the SELECT statement is not one of the values of the MEMTYPE= option in the PROC DATASETS statement.

```
      /* This step fails! */
 proc datasets memtype=(data program);
    copy out=dest;
       select apples / memtype=catalog;
 run;
```

  3. If you specify an input data library in the IN= option other than the procedure input library, the MEMTYPE= option in the PROC DATASETS statement has no affect on the copy operation. Because no subsetting has yet occurred, the procedure uses the order of precedence described in rule 1 to subset the types available for copying. The following statements successfully copy BODYFAT.DATA to the DEST data library because the SOURCE library specified in the IN= option in the COPY statement is not affected by the MEMTYPE= option in the PROC DATASETS statement.

```
proc datasets library=work
              memtype=catalog;
    copy in=source out=dest;
       select bodyfat / memtype=data;
run;
```

Copying Password-Protected SAS Files

You can copy a password-protected SAS file without specifying the password. However, because the password continues to correspond to the SAS file, you must know the password in order to access and manipulate the SAS file after you copy it.

Copying Data Sets with Long Variable Names

If the VALIDVARNAME=V6 option is set and the data set has long variable names, the long variable names are truncated, unique variables names are generated, and the copy succeeds. If VALIDVARNAME=ANY or V7, the copy fails with an error if the OUT= engine does not support long variable names.

When a variable name is truncated, the variable name is shortened to eight bytes. If this name has already been defined in the dataset, the name is shortened and a digit is added, starting with the number 2. The process of truncation and adding a digit continues until the variable name is unique. For example, a variable named LONGVARNAME becomes LONGVARN, provided that a variable with that names does not already exist in the data set. In that case, the variable names becomes LONGVAR2.

CAUTION:

Truncated variable names can collide with names already defined in the input data set. This is possible when the variable name that is already defined is exactly eight bytes long and ends in a digit. In that case, the truncated name is defined in the output data set and the name from the input data set is changed. For example,

```
options validvarname=v7;
data test;
   lonvar10='aLongVariableName';
   retain longvar1-longvar5 0;
run;
options validvarname=v6;
proc copy in=work out=sasuser;
   select test;
run;
```

In this example, LONGVAR10 is truncated to LONVAR1 and placed in the output data set. Next, the original LONGVAR1 is copied. Its name is no longer unique and so it is renamed LONGVAR2. The other variables in the input data set are also renamed according to the renaming algorithm. △

Using the COPY Procedure Instead of the COPY Statement

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. The two differences are

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If omitted, the default value is the libref of the procedure input library.
- PROC DATASETS cannot work with libraries that allow only sequential data access.

Copying Generation Groups

You can use the COPY statement to copy generation groups. However, you cannot copy individual generation files.

Transporting SAS Data Sets between Hosts

Typically, you use PROC COPY to transport SAS data sets between hosts. See Chapter 11, "The COPY Procedure," on page 269 for more information and an example.

DELETE Statement

Deletes SAS files from a SAS data library.

Featured in: Example 1 on page 391

DELETE*SAS-file(s)*
</ <ALTER=*alter-password*>
<GENNUM=ALL|HIST|REVERT|*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

SAS-file(s)
specifies one or more SAS files that you want to delete.

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files that you want to delete. You can use the ALTER= option in parentheses after the name of each SAS file or after a forward slash.

See also: "Using Passwords with the DATASETS Procedure" on page 377

GENNUM=ALL|HIST|REVERT|*integer*
restricts processing to the specified generation files. Valid values for GENNUM= are

ALL
refers to the base name and all generations.

HIST
refers to all generations (excludes the base name).

REVERT
refers to deleting the base name and changing the most current generation file, if it exists, to the base name.

postive integer
refers to an explicit generation file.

negative integer
refers to a relative generation file.

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MT=, MTYPE=

Default: DATA

See also: "Restricting Member Types Available for Processing" on page 378

Featured in: Example 1 on page 391

Details

- The SAS System immediately deletes SAS files when the RUN group executes. You do not have an opportunity to verify the delete operation before it begins.
- If you attempt to delete a SAS file that does not exist in the procedure input library, PROC DATASETS issues a message and continues processing.
- When you use the DELETE statement to delete a data set that has indexes associated with it, the statement also deletes the indexes.
- You cannot use the DELETE statement to delete a data file that has a foreign key integrity constraint or a primary key with foreign key references. For data files that have foreign keys, you must remove the foreign keys before you delete the data file. For data files that have primary keys with foreign key references, you must remove the foreign keys that reference the primary key before you delete the data file.

Working with Generation Groups

When you are working with generation groups, you can use the DELETE statement to

- delete the base name and all generations
- delete the base name and rename the youngest generation to the base name
- delete a specific generation
- delete a relative generation
- delete all generations and leave the base name.

Delete the Base Name and All Generations

- The following statements delete the base name and all generations where the base name is A:

```
proc datasets;
   delete A(gennum=all);

proc datasets;
   delete A / gennum=all;

proc datasets gennum=all;
   delete A;
```

- The following statements delete the base name and all generations where the base name begins with the letter A:

```
proc datasets;
   delete A:(gennum=all);

proc datasets;
   delete A: / gennum=all;

proc datasets gennum=all;
   delete A:;
```

Delete the Base Name and Rename Youngest Generation to the Base Name

- The following statements delete the base name and rename the youngest generation to the base name, where the base name is A:

```
proc datasets;
   delete A(gennum=revert);
```

```
proc datasets;
   delete A / gennum=revert;

proc datasets gennum=revert;
   delete A;
```

- The following statements delete the base name and rename the youngest generation to the base name, where the base name begins with the letter "A":

```
proc datasets;
   delete A:(gennum=revert);

proc datasets;
   delete A: / gennum=revert;

proc datasets gennum=revert;
   delete A:;
```

Delete a Specific Generation

- The following statements delete the first generation that uses an absolute number:

```
proc datasets;
   delete A(gennum=1);

proc datasets;
   delete A / gennum=1;

proc datasets gennum=1;
   delete A;
```

- The following statements delete a specific generation, where the base name begins with the letter A:

```
proc datasets;
   delete A:(gennum=1);

proc datasets;
   delete A: / gennum=1;

proc datasets gennum=1;
   delete A:;
```

Delete a Relative Generation

- The following statements delete the youngest version, where the base name is A, using a relative number:

```
proc datasets;
   delete A(gennum=-1);

proc datasets;
   delete A / gennum=-1;

proc datasets gennum=-1;
   delete A;
```

- The following statements delete a relative generation, where the base name begins with the letter A:

```
proc datasets;
   delete A:(gennum=-1);

proc datasets;
   delete A: / gennum=-1;

proc datasets gennum=-1;
   delete A:;
```

Delete all Generations and Leave the Base Name

- The following statements delete all generations and leave the base name, where the base name is A:

```
proc datasets;
   delete A(gennum=hist);

proc datasets;
   delete A / gennum=hist;

proc datasets gennum=hist;
   delete A;
```

- The following statements delete all generations and leave the base name, where the base name begins with the letter A:

```
proc datasets;
   delete A:(gennum=hist);

proc datasets;
   delete A: / gennum=hist;

proc datasets gennum=hist;
   delete A:;
```

EXCHANGE Statement

Exchanges the names of two SAS files in a SAS data library.

Featured in: Example 1 on page 391

EXCHANGE *name-1=other-name-1*
<…*name-n=other-name-n*>
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*> >;

Required Arguments

name=other-name
exchanges the names of SAS files in the procedure input library. Both *name* and *other-name* must already exist in the procedure input library.

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files whose names you want to exchange. You can use the ALTER= option in parentheses after the name of each SAS file or after a forward slash.

See also: "Using Passwords with the DATASETS Procedure" on page 377

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*). You can only exchange the names of SAS files of the same type.

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is ALL.

See also: "Restricting Member Types Available for Processing" on page 378

Details

- When you exchange more than one pair of names in one EXCHANGE statement, PROC DATASETS performs the exchanges in the order that the names of the SAS files occur in the directory listing, not in the order that you list the exchanges in the EXCHANGE statement.
- If the *name* SAS file does not exist in the SAS data library, PROC DATASETS stops processing the RUN group that contains the EXCHANGE statement and issues an error message. To override this behavior, specify the NOWARN option in the PROC DATASETS statement.
- The EXCHANGE statement also exchanges the associated indexes so that they correspond with the new name.
- The EXCHANGE statement only allows two existing generation groups to exchange names. You cannot exchange a specific generation number with either an existing base name or another generation number.

EXCLUDE Statement

Excludes SAS files from copying.

Restriction: Must follow a COPY statement

Restriction: Cannot appear in the same COPY step with a SELECT statement

Featured in: Example 1 on page 391

EXCLUDE *SAS-file(s)* </ MEMTYPE=*mtype*>;

Required Arguments

SAS-file(s)
specifies one or more SAS files to exclude from the copy operation. All the SAS files you name in the EXCLUDE statement must be in the library that is specified in the

IN= option in the COPY statement. If the SAS files are generation groups, the EXCLUDE statement allows only selection of the base names.

Options

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the COPY statement, or in the EXCLUDE statement, the default is MEMTYPE=ALL.

See also: "Restricting Member Types Available for Processing" on page 378 and "Specifying Member Types When Copying or Moving SAS Files" on page 351

Excluding Many Like-Named Files

You can use shortcuts for listing many SAS files in the EXCLUDE statement. For more information, see "Shortcuts for Specifying Lists of Variable Names" on page 58.

FORMAT Statement

Permanently assigns, changes, and removes variable formats in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 397

FORMAT *variable-list-1 <format-1>*
<…variable-list-n <format-n>>;

Required Arguments

variable-list
specifies one or more variables whose format you want to assign, change, or remove. If you want to disassociate a format with a variable, list the variable last in the list with no format following. For example:

```
format x1-x3 4.1 time hhmm2.2 age;
```

Options

format
specifies a format to apply to the variable or variables listed before it. If you do not specify a format, the FORMAT statement removes any format associated with the variables in *variable-list*.

Note: You can use shortcut methods for specifying variables, such as the keywords _NUMERIC, _CHARACTER_, and _ALL_. See "Shortcuts for Specifying Lists of Variable Names" on page 58 for more information. △

IC CREATE Statement

Creates an integrity constraint.

Restriction: Must be in a MODIFY RUN group

IC CREATE <*constraint-name*>=*constraint* <MESSAGE='message-string'>;

Required Arguments

<constraint-name=>
is a name for the constraint. The name must be a valid SAS name. When you do not supply a constraint name, a default name is generated. This default constraint name has the following form

Default name	Constraint type
NMxxxx	Not Null
UNxxxx	Unique
CKxxxx	Check
PKxxxx	Primary key
FKxxxx	Foreign key

where *xxxx* is a counter beginning at 0001.

Note: The names PRIMARY, FOREIGN, MESSAGE, UNIQUE, DISTINCT, CHECK, and NOT cannot be used as values for *constraint-name*. △

constraint
is the type of constraint. Valid values are

NOT NULL(*variable*)
specifies that *variable* does not contain a SAS missing value, including special missing values.

UNIQUE(*variables*)
specifies that the values of *variables* must be unique. This constraint is identical to DISTINCT.

DISTINCT(*variables*)
specifies that the values of *variables* must be unique. This constraint is identical to UNIQUE.

CHECK(WHERE-*clause*)
specifies validity checking with respect to lists and ranges. This is accomplished with a WHERE clause.

PRIMARY KEY(*variable*)
specifies a primary key variable, that is, a variable that does not contain missing values and whose values are unique.

FOREIGN KEY(*variables*) REFERENCES *table-name* <ON DELETE *referential-action*> <ON UPDATE *referential-action*>
specifies a foreign key, that is, a set of variables whose values are linked to the values of the primary key variable in another data set. The referential actions are performed when updates are made to the values of a primary key variable that is referenced by a foreign key. For a RESTRICT referential action,

a delete operation
deletes the primary key row, but only if no foreign key values matches the deleted value.

an update operation
updates the primary key value, but only if no foreign keys match the current value to be updated.

For a SET NULL referential action,

a delete operation
deletes the primary key row and sets the corresponding foreign key values to NULL.

an update operation
modifies the primary key value and sets all matching foreign key values to NULL.

MESSAGE='message-string'
"message-string" is the text of an error message that is written to the log when the data fail the constraint. For example,

```
ic create not null(socsec)
   message='Invalid Social Security number';
```

Length: The maximum length of the message is 250 characters.

The following examples show how to create integrity constraints:

```
ic create a = not null(x);
ic create Unique_D = unique(d);
ic create Distinct_DE = distinct(d e);
ic create E_less_D = check(where (e < d or d = 99));
ic create primkey = primary key(a);
ic create not null (x);
```

IC DELETE Statement

Deletes an integrity constraint.

Restriction: Must be in a MODIFY RUN group

IC DELETE *constraint-name(s)* | _ALL_;

Required Arguments

constraint-name(s)
names one or more constraints to delete. For example, to delete the constraints Unique_D and Unique_E, use this statement:

```
ic delete Unique_D Unique_E;
```

ALL
deletes all constraints.

IC REACTIVATE Statement

Reactivates a foreign key integrity constraint that is inactive.

Restriction: Must be in a MODIFY RUN group

IC REACTIVATE *foreign-key-name* REFERENCES *libref*;

Required Arguments

foreign-key-name
is the name of the foreign key to reactivate.

libref
refers to the SAS library containing the data set that contains the primary key that is referenced by the foreign key.

For example, suppose that you have a foreign key, FKEY, defined in a data set named MYLIB.MYOWN, and suppose that FKEY is linked to a primary key in the data set MAINLIB.MAIN. If the integrity constraint becomes inactive because of a processing error, you can reactivate the integrity constraint by using the following code:

```
proc datasets library=mylib;
   modify myown;
   ic reactivate fkey references mainlib;
run;
```

INDEX CENTILES

Updates centiles information for indexed variables.

Restriction: Must be in a MODIFY RUN group

INDEX CENTILES *index(s)*
</ <REFRESH>
<UPDATECENTILES= ALWAYS | NEVER | *integer*>>;

Required Arguments

index(s)
names one or more indexes.

Options

REFRESH
updates the centiles immediately, regardless of the value of UPDATECENTILES.

UPDATECENTILES=ALWAYS|NEVER|*integer*
specifies when the centiles are to be updated. It is not practical to update centiles after every data set update. Therefore, you can specify as the value of UPDATECENTILES the percent of the data values that can be changed before the centiles for the indexed variables are updated.

Valid values for UPDATECENTILES are

ALWAYS|0
Centiles are updated when the data set is closed if any changes have been made to the data set index.

NEVER|101
Centiles are not updated.

integer
The percent of values for the indexed variable that can be updated before the centiles are refreshed.

Alias: UPDCEN

Default 5 (percent)

INDEX CREATE Statement

Creates simple or composite indexes in the SAS data set specified in the MODIFY statement.

Restriction: Must be in a MODIFY RUN group

See also: "SAS Files" in *SAS Language: Reference*

Featured in: Example 3 on page 397

INDEX CREATE *index-specification(s)*
</ <NOMISS><UNIQUE>
<UPDATECENTILES= ALWAYS|NEVER|*integer*>>;

Required Arguments

index-specification(s)
can be one or both of the following forms:

variable
creates a simple index on the variable you specify.

index=(variables)
creates a composite index. The name you use for *index* is the name of the composite index. It must be a SAS name and cannot be the same as any variable name or any other composite index name. You must specify at least two variables.

Options

NOMISS
excludes from the index all observations with missing values for all index variables.

When you create an index with the NOMISS option, the SAS System uses the index only for WHERE processing and only when missing values fail to satisfy the WHERE clause. For example, if you use this WHERE statement

```
where dept ne '01';
```

the SAS System does not use the index because missing values satisfy the WHERE clause. Refer to *SAS Language Reference: Concepts*.

Note: BY-group processing ignores indexes that are created with the NOMISS option. △

Featured in: Example 3 on page 397

UNIQUE
specifies that the combination of values of the index variables must be unique. If you specify UNIQUE and multiple observations have the same values for the index variables, the index is not created.

Featured in: Example 3 on page 397

UPDATECENTILES=ALWAYS|NEVER|*integer*
specifies when the centiles are to be updated. It is not practical to update centiles after every data set update. Therefore, you can specify as the value of UPDATECENTILES the percent of the data values that can be changed before the centiles for the indexed variables are updated.

Valid values for UPDATECENTILES are

ALWAYS|0
Centiles are updated when the data set is closed if any changes have been made to the data set index.

NEVER|101
Centiles are not updated.

integer
The percent of values for the indexed variable that can be updated before the centiles are refreshed.

Alias: UPDCEN

Default 5 (percent)

How Indexes Are Affected by Changes to SAS Data Sets

Indexes are separate files in SAS data libraries, but in general, they are treated as an extension of the data set. Therefore, most data management tasks you perform on a data set also affect associated indexes, and the indexes continue to correspond to the data set. For example, if you change the name of a data set, any associated indexes continue to correspond to the data set.

INDEX DELETE Statement

Deletes one or more indexes associated with the data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

INDEX DELETE *index(s)* | _ALL_;

Required Arguments

index(s)
names one or more indexes to delete. The indexes must be on variables in the data set that is named in the preceding MODIFY statement. You can delete simple and composite indexes.

ALL
deletes all indexes, except for indexes that are owned by an integrity constraint.
The SAS System uses indexes in the following situations:

- when a user adds a primary key value. SAS must ensure that there are no duplicate values in the data set.
- when a user adds a foreign key value. SAS must ensure that a matching primary key value exists.
- when a user updates or deletes a primary key value. SAS must locate all of the matching foreign key values.
- when a user specifies unique and primary key integrity constraints. SAS must ensure that duplicate values for the specified variables do not exist in the data set.

When an index is created, it is marked as "owned" by the user, by an integrity constraint, or by both. If an index is owned by both a user and an integrity constraint, the index is not deleted until both an IC DELETE statement and an INDEX DELETE statement are processed.

Note: You can use the CONTENTS statement to produce a list of all indexes for a data set. △

INFORMAT Statement

Permanently assigns, changes, and removes variable informats in the data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 397

INFORMAT *variable-list-1 <informat-1>*
<…variable-list-n <informat-n>>;

Required Arguments

variable-list
specifies one or more variables whose informats you want to assign, change, or remove. If you want to disassociate an informat with a variable, list the variable last in the list with no informat following. For example:

```
informat a b 2. x1-x3 4.1 c;
```

Options

informat
specifies an informat for the variables immediately preceding it in the statement. If you do not specify an informat, the INFORMAT statement removes any existing informats for the variables in *variable-list*.

Note: You can use shortcut methods for specifying variables, such as the keywords _NUMERIC, _CHARACTER_, and _ALL_. See "Shortcuts for Specifying Lists of Variable Names" on page 58 for more information. △

LABEL Statement

Assigns, changes, and removes variable labels in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 397

LABEL *variable-1*=<*'label-1'*|*' '*>
<…*variable-n*=<*'label-n'*|*' '* >>;

Required Arguments

***variable*=<*'label'*>**
assigns a label to a variable. If a single quote appears in the label, write it as two single quotes in the LABEL statement. Specifying *variable*= or *variable*=' 'removes the current label.

Range: 1-40 characters

MODIFY Statement

Changes the attributes of SAS files and, through the use of subordinate statements, the attributes of variables in those SAS files.

Featured in: Example 3 on page 397

MODIFY *SAS-file* <(*file-option(s)*)>
</ <GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

To do this	Use this option
Restrict processing to a certain type of SAS file	MEMTYPE=
Specify attributes	
Assign or change a data set label	LABEL=
Assign or change a special data set type	TYPE=
Specify how the data are currently sorted	SORTEDBY=
Modify passwords	
Modify an alter password	ALTER=
Modify a read, write, or alter password	PW=
Modify a read password	READ=
Modify a write password	WRITE=
Modify generation groups	
Modify the maximum number of generations for a generation group	GENMAX=
Modify a historical version	GENNUM=

Required Arguments

SAS-file
specifies a SAS file in the procedure input library.

Options

ALTER=*password-modification*
assigns, changes, or removes an alter password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password* / *new-password*
- / *new-password*
- *old-password* /
- /

See also: “Manipulating Passwords” on page 369

GENMAX=*integer*
sets the maximum number of files in a generation group.

Range: 0 to 999

Default: 0

GENNUM=*integer*
restricts processing to the specified generation file. Valid values for GENNUM= are

postive integer
refers to an explicit generation file.

negative integer
refers to a relative generation file.

LABEL=*’data-set-label’ | ”*
assigns, changes, or removes a data set label for the SAS data set named in the MODIFY statement. If a single quote appears in the label, write it as two single quotes. LABEL= or LABEL=’ ’removes the current label.

Range: 1-40 characters

Featured in: Example 3 on page 397

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MTYPE= and MT=

Default: If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the MODIFY statement, the default is MEMTYPE=DATAVIEW.

See also: “Restricting Member Types Available for Processing” on page 378

PW=*password-modification*
assigns, changes, or removes a read, write, or alter password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password* / *new-password*
- / *new-password*
- *old-password* /
- /

See also: “Manipulating Passwords” on page 369

READ=*password-modification*
assigns, changes, or removes a read password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password* / *new-password*
- / *new-password*
- *old-password* /
- /

See also: “Manipulating Passwords” on page 369

Featured in: Example 3 on page 397

SORTEDBY=*sort-information*
specifies how the data are currently sorted. SAS stores the sort information with the file but does not verify that the data are sorted the way you indicate.
sort-information can be one of the following:

by-clause </ collate-name>
indicates how the data are currently sorted. Values for *by-clause* are the variables and options you can use in a BY statement in a PROC SORT step. *collate-name* names the collating sequence used for the sort. By default, the collating sequence is that of your host operating environment.

NULL
removes any existing sort information.

Featured in: Example 3 on page 397

TYPE=*special-type*
assigns or changes the special data set type of a SAS data set.
SAS does *not* verify

- the SAS data set type you specify in the TYPE= option (except to check if it has a length of eight or fewer characters).
- that the SAS data set's structure is appropriate for the type you have designated.

Note: Do not confuse the TYPE= option with the MEMTYPE= option. The TYPE= option specifies a type of special SAS data set. The MEMTYPE= option specifies one or more types of SAS files in a SAS data library. △

Tip: Most SAS data sets have no special type. However, certain SAS procedures, like the CORR procedure, can create a number of special SAS data sets. In addition, SAS/STAT software and SAS/EIS software support special data set types.

WRITE=*password-modification*
assigns, changes, or removes a write password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password* / *new-password*
- / *new-password*
- *old-password* /
- /

See also: "Manipulating Passwords" on page 369

Manipulating Passwords

In order to assign, change, or remove a password, you must specify the password for the highest level of protection that currently exists on that file.

Assigning Passwords

```
/* assign a password to an unprotected
   file */
modify colors (pw=green);


/* assigns an alter password to an already
   read-protected SAS data set */
 modify colors (read=green alter=red);
```

Changing Passwords

```
/* changes the write password from
   YELLOW to BROWN */
modify cars (write=yellow/brown);


/* use alter access to change the unknown
   read password to BLUE */
modify colors (read=/blue alter=red);
```

Removing Passwords

```
/* removes the alter password RED from
   STATES */
modify states (alter=red/);


/* Use alter access to remove the read
   password */
modify zoology (read=green/ alter=red);


/* Use PW= as an alias for either WRITE=
   or ALTER= to remove the unknown read
   password */
modify biology (read=/ pw=red);
```

Working with Generation Groups

Changing the Number of Generations

```
/* change the number of generations on A to 99 */
modify A(genmax=99);
```

Removing Passwords

```
/* removes the alter password RED from
   STATES(GENNUM=2) */
modify states (alter=red/) / gennum=2;
```

RENAME Statement

Renames variables in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 397

RENAME *old-name-1=new-name-1*
<*…old-name-n=new-name-n*>;

Required Arguments

old-name=new-name
changes the name of a variable in the data set specified in the MODIFY statement. *old-name* must be a variable that already exists in the data set. *new-name*, which must be a SAS name, cannot be the name of a variable that already exists in the data set or the name of an index.

Details

- If *old-name* does not exist in the SAS data set or *new-name* already exists, PROC DATASETS stops processing the RUN group containing the RENAME statement and issues an error message.
- When you use the RENAME statement to change the name of a variable for which there is a simple index, the statement also renames the index.
- If the variable that you are renaming is used in a composite index, the composite index automatically references the new variable name. However, if you attempt to rename a variable to a name that has already been used for a composite index, you receive an error message.

REPAIR Statement

Attempts to restore damaged SAS data sets or catalogs to a usable condition.

REPAIR *SAS-file(s)*
</ <ALTER=*alter-password*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

SAS-file(s)
specifies one or more SAS data sets or catalogs in the procedure input library.

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files that are named in the REPAIR statement. You can use the ALTER= option in parentheses after the name of each SAS file or after a forward slash.

See also: "Using Passwords with the DATASETS Procedure" on page 377

GENNUM=*integer*
restricts processing to the specified generation files. Valid values for GENNUM= are

postive integer
refers to an explicit generation file.

negative integer
refers to a relative generation file.

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MT=, MTYPE=

Default: If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the REPAIR statement, the default is MEMTYPE=ALL.

See also: "Restricting Member Types Available for Processing" on page 378

Details

The most common situations that require the REPAIR statement are described here:

- A system failure occurs while you are updating a SAS data set or catalog.
- The device on which a SAS data set or an associated index resides is damaged. In this case, you can restore the damaged data set or index from a backup device, but the data set and index no longer match.
- The disk that stores the SAS data set or catalog becomes full before the file is completely written to disk. You may need to free some disk space. PROC DATASETS requires free space when repairing SAS data sets with indexes and when repairing SAS catalogs.
- An I/O error occurs while you are writing a SAS data set or catalog entry.

When you use the REPAIR statement for SAS data sets, it re-creates all indexes for the data set. It also attempts to restore the data set to a usable condition, but the restored data set may not include the last several updates that occurred before the system failed. You cannot use the REPAIR statement to re-create indexes that were destroyed by using the FORCE option in a PROC SORT step.

When you use the REPAIR statement for a catalog, you receive a message stating whether the REPAIR statement restored the entry. If the entire catalog is potentially damaged, the REPAIR statement attempts to restore all the entries in the catalog. If only a single entry is potentially damaged, for example when a single entry is being updated and a disk-full condition occurs, on most systems only the entry that is open when the problem occurs is potentially damaged. In this case, the REPAIR statement attempts to repair only that entry. Some entries within the restored catalog may not include the last updates that occurred before a system crash or an I/O error. The REPAIR statement issues warning messages for entries that may have truncated data.

If the REPAIR operation is not successful, try to restore the SAS data set or catalog from your system's backup files.

The REPAIR statement can reference only a specific file to be repaired. Therefore, when you are referring to generation files, ALL, HIST, and REVERT cannot be used with the GENNUM= option.

SAVE Statement

Deletes all the SAS files in a library except the ones listed in the SAVE statement.

Featured in: Example 2 on page 395

SAVE *SAS-file(s)* </ MEMTYPE=*mtype*>;

Required Arguments

SAS-file(s)
specifies one or more SAS files that you do not want to delete from the SAS data library.

Options

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MTYPE= and MT=

Default: If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the SAVE statement, the default is MEMTYPE=ALL.

See also: "Restricting Member Types Available for Processing" on page 378

Featured in: Example 2 on page 395

Details

- If one of the SAS files in *SAS-file* does not exist in the procedure input library, PROC DATASETS stops processing the RUN group containing the SAVE statement and issues an error message. To override this behavior, specify the NOWARN option in the PROC DATASETS statement.
- When the SAVE statement deletes SAS data sets, it also deletes any indexes associated with those data sets.

  CAUTION:
  SAS immediately deletes libraries and library members when you submit a RUN group. You are not asked to verify the delete operation before it begins. Because the SAVE statement deletes many SAS files in one operation, be sure that you understand how the MEMTYPE= option affects which types of SAS files are saved and which types are deleted. △

- When you use the SAVE statement with generation groups, the SAVE statement treats the base name and all generations as a unit. You cannot save a specific generation file.

SELECT Statement

Selects SAS files for copying.

Restriction: Must follow a COPY statement

Restriction: Cannot appear with an EXCLUDE statement in the same COPY step

Featured in: Example 1 on page 391

SELECT *SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE= *mtype*>>;

Required Arguments

SAS-file(s)
specifies one or more SAS files that you want to copy. All the SAS files you name must be in the data library that is referenced by the libref named in the IN= option in the COPY statement. If the SAS files are generation groups, the SELECT statement allows only selection of the base names.

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files that you are moving from one data library to another. Because you are moving, and thus deleting, a SAS file from a SAS data library, you need alter access. You can use the ALTER= option in parentheses after the name of each SAS file or after a forward slash.

See also: "Using Passwords with the DATASETS Procedure" on page 377

MEMTYPE=*mtype*
restricts processing to one member type (*mtype*).

Aliases: MTYPE= and MT=

Default: If you do not specify the MEMTYPE= option in the PROC DATASETS statement, in the COPY statement, or in the SELECT statement, the default is MEMTYPE=ALL.

See also: "Specifying Member Types When Copying or Moving SAS Files" on page 351 and "Restricting Member Types Available for Processing" on page 378

Featured in: Example 1 on page 391

Selecting Many Like-Named Files

You can use shortcuts for listing many SAS files in the SELECT statement. For more information, see "Shortcuts for Specifying Lists of Variable Names" on page 58.

Concepts

Procedure Execution

When you start the DATASETS procedure, you specify the procedure input library in the PROC DATASETS statement. If you omit a procedure input library, the procedure processes the current default SAS data library (usually the WORK data library). To specify a new procedure input library, start the DATASETS procedure again.

Statements execute in the order they are written. For example, if you want to see the contents of a data set, copy a data set, and then compare the contents of the second data set with the first, the statements that perform those tasks must appear in that order (that is, CONTENTS, COPY, CONTENTS).

RUN-Group Processing

PROC DATASETS supports RUN-group processing. RUN-group processing enables you to submit RUN groups without ending the procedure.

The DATASETS procedure supports four types of RUN groups. Each RUN group is defined by the statements that compose it and by what causes it to execute.

Some statements in PROC DATASETS act as *implied* RUN statements because they cause the RUN group preceding them to execute.

The following list discusses what statements compose a RUN group and what causes each RUN group to execute:

- The PROC DATASETS statement always executes immediately. No other statement is necessary to cause the PROC DATASETS statement to execute. Therefore, the PROC DATASETS statement alone is a RUN group.
- The MODIFY statement, and any of its subordinate statements, form a RUN group. These RUN groups always execute immediately. No other statement is necessary to cause a MODIFY RUN group to execute.
- The APPEND, CONTENTS, and COPY statements (including EXCLUDE and SELECT, if present), form their own separate RUN groups. Every APPEND statement forms a single-statement RUN group; every CONTENTS statement forms a single-statement RUN group; and every COPY step forms a RUN group. Any other statement in the procedure, except those that are subordinate to either the COPY or MODIFY statement, causes the RUN group to execute.
- One or more of the following statements form a RUN group:

AGE	EXCHANGE
CHANGE	REPAIR
DELETE	SAVE

If any of these statements appear in sequence in the PROC step, the sequence forms a RUN group. For example, if a REPAIR statement appears immediately after a SAVE statement, the REPAIR statement does not force the SAVE statement to execute; it becomes part of the same RUN group. To execute the RUN group, submit one of the following statements:

PROC DATASETS	MODIFY
APPEND	QUIT
CONTENTS	RUN
COPY	another DATA or PROC step

The SAS System reads the program statements that are associated with one task until it reaches a RUN statement or an implied RUN statement. It executes all of the preceding statements immediately, then continues reading until it reaches another RUN statement or implied RUN statement. To execute the last task, you must use a RUN statement or a statement that stops the procedure.

The following PROC DATASETS step contains five RUN groups:

```
libname dest 'SAS-data-library';
   /* RUN group */
proc datasets;
      /* RUN group */
   change nutr=fatg;
   delete bldtest;
   exchange xray=chest;
      /* RUN group */
   copy out=dest;
      select report;
      /* RUN group */
   modify bp;
      label dias='Taken at Noon';
      rename weight=bodyfat;
      /* RUN group */
   append base=tissue data=newtiss;
quit;
```

Note: If you are running in interactive line mode, you can receive messages that statements have already executed before you submit a RUN statement. Plan your tasks carefully if you are using this environment for running PROC DATASETS. △

Error Handling

Generally, if an error occurs in a statement, the RUN group containing the error does not execute. RUN groups preceding or following the one containing the error execute normally. The MODIFY RUN group is an exception. If a syntax error occurs in a statement subordinate to the MODIFY statement, only the statement containing the error fails. The other statements in the RUN group execute.

Note that if the first word of the statement (the statement name) is in error and the procedure cannot recognize it, the procedure treats the statement as part of the preceding RUN group.

Password Errors

If there is an error involving an incorrect or omitted password in a statement, the error only affects the statement containing the error. The other statements in the RUN group execute.

Forcing a RUN Group with Errors to Execute

The FORCE option in the PROC DATASETS statement forces execution of the RUN group even if one or more of the statements contain errors. Only the statements that are error-free execute.

Ending the Procedure

To stop the DATASETS procedure, you must issue a QUIT statement, a RUN CANCEL statement, a new PROC statement, or a DATA statement. Submitting a QUIT statement executes any statements that have not executed. Submitting a RUN CANCEL statement cancels any statements that have not executed.

Using Passwords with the DATASETS Procedure

Several statements in PROC DATASETS support options that manipulate passwords on SAS files. These options, ALTER=, PW=, READ=, and WRITE=, are also data set options.* If you do not know how passwords affect SAS files, refer to *SAS Language Reference: Concepts*.

When you are working with password-protected SAS files in the AGE, CHANGE, DELETE, EXCHANGE, REPAIR, or SELECT statement, you can specify password options in the PROC DATASETS statement or in a subordinate statement. The SAS System searches for passwords in the following order:

1. in parentheses after the name of the SAS file in a subordinate statement. When used in parentheses, the option only refers to the name immediately preceding the option. If you are working with more than one SAS file in a data library and each SAS file has a different password, you must specify password options in parentheses after individual names.

   In the following statement, the ALTER= option provides the password RED for the SAS file BONES only:

   ```
   delete xplant bones(alter=red);
   ```

2. after a forward slash (/) in a subordinate statement. When you use a password option following a slash, the option refers to all SAS files named in the statement unless the same option appears in parentheses after the name of a SAS file. This method is convenient when you are working with more than one SAS file and they all have the same password.

   In the following statement, the ALTER= option in parentheses provides the password RED for the SAS file CHEST, and the ALTER= option after the slash provides the password BLUE for the SAS file VIRUS:

   ```
   delete chest(alter=red) virus /
          alter=blue;
   ```

3. in the PROC DATASETS statement. Specifying the password in the PROC DATASETS statement can be useful if all the SAS files you are working with in the library have the same password.

   In the following PROC DATASETS step, the PW= option provides the password RED for the SAS files INSULIN and ABNEG:

   ```
   proc datasets pw=red;
      delete insulin;
   ```

* In the APPEND and CONTENTS statements, you use these options just as you use any SAS data set option, in parentheses after the SAS data set name.

```
   contents data=abneg;
run;
```

Note: For the password for a SAS file in a SELECT statement, the SAS System looks in the COPY statement before it looks in the PROC DATASETS statement. △

Restricting Member Types Available for Processing

In the PROC DATASETS Statement

If you name a member type or several member types in the PROC DATASETS statement, in most subsequent statements (the CONTENTS and COPY statements are exceptions to this rule) you can name only a subset of the list of member types included in the PROC DATASETS statement. The directory listing that the PROC DATASETS statement writes to the SAS log includes only those SAS files of the type specified in the MEMTYPE= option.

In Subordinate Statements

Use the MEMTYPE= option in the following subordinate statements to limit the member types that are available for processing:

AGE

CHANGE

DELETE

EXCHANGE

EXCLUDE

REPAIR

SAVE

SELECT

Note: The MEMTYPE= option works slightly differently for the CONTENTS and COPY statements. Refer to "CONTENTS Statement" on page 346 and "COPY Statement" on page 349 for more information. △

The procedure searches for MEMTYPE= in the following order:

1. in parentheses immediately after the name of a SAS file. When used in parentheses, the MEMTYPE= option refers only to the SAS file immediately preceding the option. For example, the following statement deletes HOUSE.DATA, LOT.CATALOG, and SALES.DATA because the default member type for the DELETE statement is DATA. (Refer to Table 14.3 on page 379 for the default types for each statement.)

   ```
   delete house lot(memtype=catalog) sales;
   ```

2. after a slash (/) at the end of the statement. When used following a slash, the MEMTYPE= option refers to all SAS files named in the statement unless the option appears in parentheses after the name of a SAS file. For example, the following statement deletes LOTPIX.CATALOG, REGIONS.DATA, and APPL.CATALOG:

```
delete lotpix regions(memtype=data) appl
       / memtype=catalog;
```

3 in the PROC DATASETS statement. For example, this PROC DATASETS step deletes APPL.CATALOG:

```
proc datasets memtype=catalog;
   delete appl;
run;
```

Note: When you use the EXCLUDE and SELECT statements, the procedure looks in the COPY statement for the MEMTYPE= option before it looks in the PROC DATASETS statement. For more information, see "Specifying Member Types When Copying or Moving SAS Files" on page 351. △

4 for the default value. If you do not specify a MEMTYPE= option in the subordinate statement or in the PROC DATASETS statement, the default value for the subordinate statement determines the member type available for processing.

Member Types

The following list gives the possible values for the MEMTYPE= option:

ACCESS
access descriptor files (created by SAS/ACCESS software)

ALL
all member types

CATALOG
SAS catalogs

DATA
SAS data sets

FDB
financial database

MDDB
multidimensional database

PROGRAM
stored compiled SAS programs

VIEW
SAS data views

Table 14.3 on page 379 shows the member types that you can use in each statement:

Table 14.3 Subordinate Statements and Appropriate Member Types

Statement	Appropriate member types	Default member type
AGE	ACCESS, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	DATA
CHANGE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL

Statement	Appropriate member types	Default member type
CONTENTS	ALL, DATA, VIEW	DATA[1]
COPY	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
DELETE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	DATA
EXCHANGE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
EXCLUDE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
MODIFY	ACCESS, DATA, VIEW	DATA
REPAIR	ALL, CATALOG, DATA	ALL[2]
SAVE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
SELECT	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL

1 When DATA=_ALL_ in the CONTENTS statement, the default is ALL. ALL includes only DATA and VIEW.
2 ALL includes only DATA and CATALOG.

Results

Directory Listing to the SAS Log

The PROC DATASETS statement lists the SAS files in the procedure input library unless the NOLIST option is specified. If you specify the MEMTYPE= option, only specified types are listed. If you specify the DETAILS option, PROC DATASETS prints three additional columns of information: **`Obs or Entries`**, **`Vars`**, and **`Label`**.

Directory Listing as SAS Output

The CONTENTS statement lists the directory of the procedure input library if you use the DIRECTORY option or use DATA=_ALL_.

If you want only a directory, use the NODS option and the _ALL_ keyword in the DATA= option. The NODS option suppresses the description of the SAS data sets; only the directory appears in the output.

Note: The CONTENTS statement does not put a directory in an output data set. If you try to create an output data set using the NODS option, you receive an empty output data set. Use the SQL procedure to create a SAS data set that contains information about a SAS data library. See “DICTIONARY tables” on page 1062 for more information. △

Procedure Output

The only statement in PROC DATASETS that produces procedure output is the CONTENTS statement. This section shows the output from the CONTENTS statement

for the GROUP data set, including the modifications made to the GROUP data set in Example 3 on page 397.

Only the items in the output that require explanation are discussed.

Data Set Attributes

Here are descriptions of selected fields in Output 14.2 on page 381:

Member Type
names the type of library member (DATA or VIEW).

Protection
indicates whether the SAS data set is READ, WRITE, or ALTER protected.

Data Set Type
names the special data set type (such as CORR, COV, SSPC, EST, or FACTOR), if any.

Deleted Observations
is the number of observations marked for deletion. These observations are not included in the total number of observations, shown in the **Observations** field.

Compressed
indicates whether the data set is compressed. If the data set is compressed, the output includes an additional item, **Reuse Space** (with a value of YES or NO), that indicates whether to reuse space that is made available when observations are deleted.

Sorted
indicates whether the data set is sorted. If you sort the data set with PROC SORT, PROC SQL, or specify sort information with the SORTEDBY= data set option, a value of YES appears here, and there is an additional section to the output. See "Sort Information" on page 384 for details.

Output 14.2 Data Set Attributes Section

The Contents of the GROUP Data Set

DATASETS PROCEDURE

Data Set Name:	HEALTH.GROUP	Observations:	148
Member Type:	DATA	Variables:	11
Engine:	V7	Indexes:	1
Created:	10:42 Thursday, August 28, 1997	Observation Length:	96
Last Modified:	11:13 Thursday, August 28, 1997	Deleted Observations:	0
Protection:	READ	Compressed:	NO
Data Set Type:		Sorted:	YES
Label:	Test Subjects		

Engine and Operating Environment-dependent Information

The CONTENTS statement produces operating environment-specific and engine-specific information. This information differs depending on the operating environment. The example in Output 14.3 on page 381 is from the UNIX environment.

Output 14.3 Engine and Operating Environment Dependent Information Section of CONTENTS Output

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

                  Data Set Page Size:          8192

                  Number of Data Set Pages:    4

                  File Format:                 7

                  First Data Page:             1

                  Max Obs per Page:            84

                  Obs in First Data Page:      62

                  Index File Page Size:        4096

                  Number of Index File Pages: 2

                  Number of Data Set Repairs: 0

                  File Name:                   external-file
                  Release Created:             7.00.000

                  Host Created:                HP-UX

                  Inode Number:                718939608

                  Access Permission:           rw-r--r--

                  Owner Name:                  UNIX-userid

                  File Size (bytes):           40960
```

Alphabetic List of Variables and Attributes

Here are descriptions of selected columns in Output 14.4 on page 382:

#
indicates the position of each variable in the data set.

Variable
is the name of each variable. By default, variables appear alphabetically.

Note: Variable names are sorted such that X1, X2, and X10 appear in that order and not in the true collating sequence of X1, X10, and X2. Variable names that contain an underscore and digits may appear in a nonstandard sort order. For example, P25 and P75 appear before P2_5. △

Type
specifies the type of variable, character or numeric.

Pos
specifies the starting position of each variable in the observation.

Note: If none of the variables in the SAS data set has a format, informat, or label associated with it, the column for that attribute does not appear. △

Output 14.4 Variable Attributes Section

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

#	Variable	Type	Len	Pos	Format	Informat	Label
9	BIRTH	Num	8	8	DATE7.	DATE7.	
4	CITY	Char	15	58	$.	$.	
3	FNAME	Char	15	43	$.	$.	
10	HIRED	Num	8	16	DATE7.	DATE7.	
11	HPHONE	Char	12	79	$.	$.	
1	IDNUM	Char	4	24	$.	$.	
7	JOBCODE	Char	3	76	$.	$.	
2	LNAME	Char	15	28	$.	$.	
8	SALARY	Num	8	0	COMMA8.		current salary excluding bonus
6	SEX	Char	1	75	$.	$.	
5	STATE	Char	2	73	$.	$.	

Alphabetic List of Indexes and Attributes

The section shown in Output 14.5 on page 383 appears only if the data set has indexes associated with it.

#
indicates the number of each index. The CONTENTS statement numbers the indexes sequentially as they are defined.

Index
displays the name of each index. For simple indexes, the name of the index is the same as a variable in the data set.

Unique Option
indicates whether the index must have unique values. If the column contains YES, the combination of values of the index variables is unique for each observation.

Nomiss Option
indicates whether the index excludes missing values for all index variables. If the column contains YES, the index does not contain observations with missing values for all index variables.

# of Unique Values
gives the number of unique values in the index.

Variables
names the variables in a composite index.

Output 14.5 Index Attributes Section

```
-----Alphabetic List of Indexes and Attributes-----

                                      # of
                Unique    Nomiss    Unique
#    Index      Option    Option    Values   Variables
-------------------------------------------------------
1    vital      YES       YES          148    BIRTH SALARY
```

Sort Information

The section shown in Output 14.6 on page 384 appears only if the **Sorted** field has a value of YES.

Sortedby
indicates how the data are currently sorted. This field contains either the variables and options you use in the BY statement in PROC SORT, the column name in PROC SQL, or the values you specify in the SORTEDBY= option.

Validated
indicates whether PROC SORT or PROC SQL sorted the data. If PROC SORT or PROC SQL sorted the data set, the value is YES. If you assigned the sort information with the SORTEDBY= data set option, the value is NO.

Character Set
is the character set used to sort the data. The value for this field can be ASCII, EBCDIC, or PASCII.

Collating Sequence
is the collating sequence used to sort the data set. This field does not appear if you do not specify a specific collating sequence that is different from the character set. (not shown)

Sort Option
indicates whether PROC SORT used the NODUPKEY or NODUPREC option when sorting the data set. This field does not appear if you did not use one of these options in a PROC SORT statement. (not shown)

Output 14.6 Sort Information Section

```
                    -----Sort Information-----

                       Sortedby:      LNAME
                       Validated:     NO
                       Character Set: ASCII
```

Output Data Sets

The CONTENTS statement is the only statement in the DATASETS procedure that generates output data sets.

The OUT= Data Set

The OUT= option in the CONTENTS statement creates an output data set. Each variable in each DATA= data set has one observation in the OUT= data set. These are the variables in the output data set:

CHARSET
the character set used to sort the data set. The value is ASCII, EBCDIC, or PASCII. A blank appears if the data set does not have sort information stored with it.

COLLATE
the collating sequence used to sort the data set. A blank appears if the sort information for the input data set does not include a collating sequence.

COMPRESS
indicates whether the data set is compressed.

CRDATE
date the data set was created.

DELOBS
number of observations marked for deletion in the data set. (Observations can be marked for deletion but not actually deleted when you use the FSEDIT procedure of SAS/FSP software.)

ENCRYPT
indicates whether the data set is encrypted.

ENGINE
name of the method used to read from and write to the data set.

FORMAT
variable format. The value of FORMAT is a blank if you do not associate a format with the variable.

FORMATD
number of decimals you specify when you associate the format with the variable. The value of FORMATD is 0 if you do not specify decimals in the format.

FORMATL
format length. If you specify a length for the format when you associate the format with a variable, the length you specify is the value of FORMATL. If you do not specify a length for the format when you associate the format with a variable, the value of FORMATL is the default length of the format if you use the FMTLEN option and 0 if you do not use the FMTLEN option.

GENMAX
maximum number of files (generations) for the generation group.

GENNEXT
the next generation number for a generation group.

GENNUM
the generation number.

IDXCOUNT
number of indexes for the data set.

IDXUSAGE
use of the variable in indexes. Possible values are

NONE
the variable is not part of an index.

SIMPLE
the variable has a simple index. No other variables are included in the index.

COMPOSITE
the variable is part of a composite index.

BOTH
the variable has a simple index and is part of a composite index.

INFORMAT
variable informat. The value is a blank if you do not associate an informat with the variable.

INFORMD
number of decimals you specify when you associate the informat with the variable. The value is 0 if you do not specify decimals when you associate the informat with the variable.

INFORML
informat length. If you specify a length for the informat when you associate the informat with a variable, the length you specify is the value of INFORML. If you do not specify a length for the informat when you associate the informat with a variable, the value of INFORML is the default length of the informat if you use the FMTLEN option and 0 if you do not use the FMTLEN option.

JUST
justification (0=left, 1=right).

LABEL
variable label (blank if none given).

LENGTH
variable length.

LIBNAME
libref used for the data library.

MEMLABEL
label for this SAS data set (blank if no label).

MEMNAME
SAS data set that contains the variable.

MEMTYPE
library member type (DATA or VIEW).

MODATE
date the data set was last modified.

NAME
variable name.

NOBS
number of observations in the data set.

NODUPKEY
indicates whether the NODUPKEY option was used in a PROC SORT statement to sort the input data set.

NODUPREC
indicates whether the NODUPREC option was used in a PROC SORT statement to sort the input data set.

NPOS
physical position of the first character of the variable in the data set.

POINTOBS
indicates if the data set can be addressed by observation.

PROTECT
the first letter of the level of protection. The value for PROTECT is one or more of the following:

A	indicates the data set is alter-protected.
R	indicates the data set is read-protected.
W	indicates the data set is write-protected.

REUSE
indicates whether the space made available when observations are deleted from a compressed data set should be reused. If the data set is not compressed, the REUSE variable has a value of NO.

SORTED
the value depends on the sorting characteristics of the input data set. Possible values are

. (period)	for not sorted.
0	for sorted but not validated.
1	for sorted and validated.

SORTEDBY
the value depends on that variable's role in the sort. Possible values are

. (period)
if the variable was not used to sort the input data set.

n
where *n* is an integer that denotes the position of that variable in the sort. A negative value of *n* indicates that the data set is sorted by the descending order of that variable.

TYPE
type of the variable (1=numeric, 2=character).

TYPEMEM
special data set type (blank if no TYPE= value is specified).

VARNUM
variable number in the data set. Variables are numbered in the order they appear.

The output data set is sorted by the variables LIBNAME and MEMNAME.

Note: The variable names are sorted so that the values X1, X2, and X10 are listed in that order, not in the true collating sequence of X1, X10, X2. Therefore, if you want to use a BY statement on MEMNAME in subsequent steps, run a PROC SORT step on the output data set first or use the NOTSORTED option in the BY statement. △

Output 14.7 on page 387 is an example of an output data set created from the GROUP data set, which is shown in Example 4 on page 399 and in "Procedure Output" on page 380.

Output 14.7 The Data Set Health.Grpout

An Example of an Output Data Set 1

OBS	LIBNAME	MEMNAME	MEMLABEL	TYPEMEM	NAME	TYPE	LENGTH	VARNUM
1	HEALTH	GROUP	Test Subjects		BIRTH	1	8	9
2	HEALTH	GROUP	Test Subjects		CITY	2	15	4
3	HEALTH	GROUP	Test Subjects		FNAME	2	15	3
4	HEALTH	GROUP	Test Subjects		HIRED	1	8	10
5	HEALTH	GROUP	Test Subjects		HPHONE	2	12	11
6	HEALTH	GROUP	Test Subjects		IDNUM	2	4	1
7	HEALTH	GROUP	Test Subjects		JOBCODE	2	3	7
8	HEALTH	GROUP	Test Subjects		LNAME	2	15	2
9	HEALTH	GROUP	Test Subjects		SALARY	1	8	8
10	HEALTH	GROUP	Test Subjects		SEX	2	1	6
11	HEALTH	GROUP	Test Subjects		STATE	2	2	5

OBS	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML
1		DATE	7	0	DATE	7
2		$	0	0	$	0
3		$	0	0	$	0
4		DATE	7	0	DATE	7
5		$	0	0	$	0
6		$	0	0	$	0
7		$	0	0	$	0
8		$	0	0	$	0
9	current salary excluding bonus	COMMA	8	0		0
10		$	0	0	$	0
11		$	0	0	$	0

An Example of an Output Data Set 2

OBS	INFORMD	JUST	NPOS	NOBS	ENGINE	CRDATE	MODATE	DELOBS
1	0	0	8	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
2	0	0	58	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
3	0	0	43	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
4	0	0	16	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
5	0	0	79	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
6	0	0	24	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
7	0	0	76	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
8	0	0	28	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
9	0	0	0	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
10	0	0	75	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0
11	0	0	73	148	V7	28AUG97:10:42:26	28AUG97:11:13:46	0

OBS	IDXUSAGE	MEMTYPE	IDXCOUNT	PROTECT	FLAGS	COMPRESS	REUSE	SORTED	SORTEDBY
1	COMPOSITE	DATA	1	R--	---	NO	NO	0	.
2	NONE	DATA	1	R--	---	NO	NO	0	.
3	NONE	DATA	1	R--	---	NO	NO	0	.
4	NONE	DATA	1	R--	---	NO	NO	0	.
5	NONE	DATA	1	R--	---	NO	NO	0	.
6	NONE	DATA	1	R--	---	NO	NO	0	.
7	NONE	DATA	1	R--	---	NO	NO	0	.
8	NONE	DATA	1	R--	---	NO	NO	0	1
9	COMPOSITE	DATA	1	R--	---	NO	NO	0	.
10	NONE	DATA	1	R--	---	NO	NO	0	.
11	NONE	DATA	1	R--	---	NO	NO	0	.

An Example of an Output Data Set 3

OBS	CHARSET	COLLATE	NODUPKEY	NODUPREC	ENCRYPT	POINTOBS	GENMAX	GENNUM	GENNEXT
1	ASCII		NO	NO	NO	YES	0	.	0
2	ASCII		NO	NO	NO	YES	0	.	0
3	ASCII		NO	NO	NO	YES	0	.	0
4	ASCII		NO	NO	NO	YES	0	.	0
5	ASCII		NO	NO	NO	YES	0	.	0
6	ASCII		NO	NO	NO	YES	0	.	0
7	ASCII		NO	NO	NO	YES	0	.	0
8	ASCII		NO	NO	NO	YES	0	.	0
9	ASCII		NO	NO	NO	YES	0	.	0
10	ASCII		NO	NO	NO	YES	0	.	0
11	ASCII		NO	NO	NO	YES	0	.	0

The OUT2= Data Set

The OUT2= option in the CONTENTS statement creates an output data set that contains information about indexes and integrity constraints. These are the variables in the output data set:

NUMVARS
the number of variables involved in the index or integrity constraint.

NAME
the name of the index or integrity constraint.

TYPE
the type. For an index, the value is "Index" while for an integrity constraint, the value is the type of integrity constraint (Not Null, Check, Primary Key, etc.).

RECREATE
the SAS statement necessary to recreate the index or integrity constraint.

MG
the value of MESSAGE=, if it is used, in the IC CREATE statement.

INACTIVE
contains YES if the integrity constraint is inactive.

ONDELETE
for a foreign key integrity constraint, contains RESTRICT or SET NULL if applicable (the ON DELETE option in the IC CREATE statement).

ONUPDATE
for a foreign key integrity constraint, contains RESTRICT or SET NULL if applicable (the ON UPDATE option in the IC CREATE statement).

REFERENCE
for a foreign key integrity constraint, contains the name of the referenced data set.

WHERE
for a check integrity constraint, contains the WHERE statement.

UNIQUE
contains YES if the UNIQUE option is defined for the index.

NOMISS
contains YES if the NOMISS option is defined for the index.

IC_BUILD
contains YES if the index was built by creating an integrity constraint.

IC_OWN
contains YES if the index is owned by the integrity constraint.

NUMVALS
the number of distinct values in the index (displayed for centiles).

UPERCMX
the percentage of the index update that triggers a refresh (displayed for centiles).

UPERC
the percentage of the index that has been updated since the last refresh (displayed for centiles).

Examples

Example 1: Manipulating SAS Files

Procedure features:

PROC DATASETS statement options:
- DETAILS
- LIBRARY=

CHANGE statement

COPY statement options:
- MEMTYPE
- MOVE
- OUT=

DELETE statement option:
- MEMTYPE=

EXCHANGE statement

EXCLUDE statement

SELECT statement option:
- MEMTYPE=

This example
- changes the names of SAS files
- copies SAS files between SAS data libraries
- deletes SAS files
- selects SAS files to copy
- exchanges the names of SAS files
- excludes SAS files from a copy operation.

Program

The SAS system option SOURCE writes the programming statements to the SAS log.

```
options pagesize=40 linesize=132 nodate pageno=1 source;


libname dest1 'SAS-data-library-1';
libname dest2 'SAS-data-library-2';
libname health 'SAS-data-library-3';
```

LIBRARY= specifies the procedure input library. DETAILS prints three additional columns in the directory: **Obs or Entries**, **Vars**, and **Label**. All member types are available for processing because the MEMTYPE= option does not appear in the PROC DATASETS statement.

```
proc datasets library=health details;
```

The DELETE statement deletes the TENSION data set and the A2 catalog. MT=CATALOG only applies to A2 and is necessary because the default member type for the DELETE statement is DATA. The CHANGE statement changes the name of the A1 catalog to POSTDRUG. The EXCHANGE statement exchanges the names of the WEIGHT and BODYFAT data sets. MEMTYPE= is not necessary in the CHANGE or EXCHANGE statement because the default is MEMTYPE=ALL for each statement.

```
    delete tension a2(mt=catalog);
    change a1=postdrug;
    exchange weight=bodyfat;
```

MEMTYPE=VIEW restricts processing to SAS data views. MOVE specifies that all SAS data views named in the SELECT statements in this step be deleted from the HEALTH data library and moved to the DEST1 data library.

```
    copy out=dest1 move memtype=view;
```

The SELECT statement specifies that the SAS data view SPDATA be moved from the HEALTH data library to the DEST1 data library.

```
       select spdata;
```

The SELECT statement specifies that the catalogs ETEST1 through ETEST5 be moved from the HEALTH data library to the DEST1 data library. MEMTYPE=CATALOG overrides the MEMTYPE=VIEW option in the COPY statement.

```
       select etest1-etest5 / memtype=catalog;
```

The EXCLUDE statement excludes from the COPY operation all SAS files that begin with the letter D and the other SAS files listed. All remaining SAS files in the HEALTH data library are copied to the DEST2 data library.

```
    copy out=dest2;
       exclude d: mlscl oxygen source test2 vision weight;
quit;
```

SAS Log

(UNIX Environment)

```
      cpu time            0.00 seconds

1
2   libname dest1
3   'SAS-data-library';
NOTE: Libref DEST1 was successfully assigned as follows:
      Engine:        V8
      Physical Name: UNIX-path
4   libname dest2
5   'SAS-data-library';
NOTE: Libref DEST2 was successfully assigned as follows:
      Engine:        V8
      Physical Name: UNIX-path
6   libname health
7   'SAS-data-library ';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V8
      Physical Name: UNIX-path

17   proc datasets library=health details;
                                                      -----Directory-----

                    Libref:            HEALTH
                    Engine:            V8
                    Physical Name:     UNIX-path
                    File Name:         UNIX-path
                    Inode Number:      inode-number
                    Access Permission: rwxr-xr-x
                    Owner Name:        UNIX-user
                    File Size (bytes): 4096
```

#	Name	Memtype	Obs, Entries or Indexes	Vars	Label	File size	Last modified
1	A1	CATALOG	23			69632	09FEB1999:10:42:10
2	A2	CATALOG	1			24576	09FEB1999:10:42:11
3	ALL	DATA	23	17		17408	09FEB1999:10:42:11
4	BODYFAT	DATA	1	2		12288	09FEB1999:10:42:11
5	CONFOUND	DATA	8	4		12288	09FEB1999:10:42:11
6	CORONARY	DATA	39	4		12288	09FEB1999:10:42:11
7	DRUG1	DATA	6	2	JAN95 Data	12288	09FEB1999:10:42:11
8	DRUG2	DATA	13	2	MAY95 Data	12288	09FEB1999:10:42:11
9	DRUG3	DATA	11	2	JUL95 Data	12288	09FEB1999:10:42:11
10	DRUG4	DATA	7	2	JAN92 Data	12288	09FEB1999:10:42:11
11	DRUG5	DATA	1	2	JUL92 Data	12288	09FEB1999:10:42:11
12	ETEST1	CATALOG	1			24576	09FEB1999:10:42:11
13	ETEST2	CATALOG	1			24576	09FEB1999:10:42:12
14	ETEST3	CATALOG	1			24576	09FEB1999:10:42:12
15	ETEST4	CATALOG	1			24576	09FEB1999:10:42:12
16	ETEST5	CATALOG	1			24576	09FEB1999:10:42:12
17	ETESTS	CATALOG	1			24576	09FEB1999:10:42:12
18	FORMATS	CATALOG	6			24576	09FEB1999:10:42:12
19	GROUP	DATA	148	11		32768	09FEB1999:10:42:13
20	GRPOUT	DATA	11	40		24576	29JAN1999:13:38:22
21	INFANT	DATA	149	6		23552	18JAN1999:14:08:42
22	MLSCL	DATA	32	4	Multiple Sclerosis Data	12288	09FEB1999:10:42:13
23	NAMES	DATA	7	4		12288	09FEB1999:10:42:13
24	OXYGEN	DATA	31	7		13312	09FEB1999:10:42:13
25	PERSONL	DATA	148	11		32768	09FEB1999:10:42:13
26	PHARM	DATA	6	3	Sugar Study	12288	09FEB1999:10:42:13
27	POINTS	DATA	6	6		12288	09FEB1999:10:42:13
28	PRENAT	DATA	149	6		23552	09FEB1999:10:42:13
29	RESULTS	DATA	10	5		12288	09FEB1999:10:42:13
30	SLEEP	DATA	108	6		16384	09FEB1999:10:42:13
31	SOURCE	PROGRAM				16384	09FEB1999:10:42:13
32	SPDATA	VIEW	.	2		12288	09FEB1999:10:42:14
33	SUR	DATA	3	6		12288	18JAN1999:14:08:44
34	SYNDROME	DATA	46	8		16384	09FEB1999:10:42:14
35	TENSION	DATA	4	3		12288	09FEB1999:10:42:14
36	TEST2	DATA	15	5		12288	09FEB1999:10:42:14
37	TRAIN	DATA	7	2		12288	09FEB1999:10:42:14
38	VISION	DATA	16	3		12288	09FEB1999:10:42:14
39	WEIGHT	DATA	83	13	California Results	26624	09FEB1999:10:42:14
40	WGHT	DATA	83	13	California Results	26624	09FEB1999:10:42:14

```
19      delete tension a2(mt=catalog);
20      change a1=postdrug;
21      exchange weight=bodyfat;
NOTE: Deleting HEALTH.TENSION (memtype=DATA).
NOTE: Deleting HEALTH.A2 (memtype=CATALOG).
NOTE: Changing the name HEALTH.A1 to HEALTH.POSTDRUG (memtype=CATALOG).
NOTE: Exchanging the names HEALTH.WEIGHT and HEALTH.BODYFAT (memtype=DATA).
22      copy out=dest1 move memtype=view;
23
24         select spdata;
25
26         select etest1-etest5 / memtype=catalog;
27
NOTE: Moving HEALTH.SPDATA to DEST1.SPDATA (memtype=VIEW).
NOTE: Moving HEALTH.ETEST1 to DEST1.ETEST1 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST2 to DEST1.ETEST2 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST3 to DEST1.ETEST3 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST4 to DEST1.ETEST4 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST5 to DEST1.ETEST5 (memtype=CATALOG).
28      copy out=dest2;
29         exclude d: mlscl oxygen source test2 vision weight;
30   quit;
NOTE: Copying HEALTH.ALL to DEST2.ALL (memtype=DATA).
NOTE: The data set DEST2.ALL has 23 observations and 17 variables.
NOTE: Copying HEALTH.BODYFAT to DEST2.BODYFAT (memtype=DATA).
NOTE: The data set DEST2.BODYFAT has 83 observations and 13 variables.
NOTE: Copying HEALTH.CONFOUND to DEST2.CONFOUND (memtype=DATA).
NOTE: The data set DEST2.CONFOUND has 8 observations and 4 variables.
NOTE: Copying HEALTH.CORONARY to DEST2.CORONARY (memtype=DATA).
NOTE: The data set DEST2.CORONARY has 39 observations and 4 variables.
NOTE: Copying HEALTH.ETESTS to DEST2.ETESTS (memtype=CATALOG).
NOTE: Copying HEALTH.FORMATS to DEST2.FORMATS (memtype=CATALOG).
NOTE: Copying HEALTH.GROUP to DEST2.GROUP (memtype=DATA).
NOTE: The data set DEST2.GROUP has 148 observations and 11 variables.
NOTE: Copying HEALTH.GRPOUT to DEST2.GRPOUT (memtype=DATA).
NOTE: The data set DEST2.GRPOUT has 11 observations and 40 variables.
NOTE: Copying HEALTH.INFANT to DEST2.INFANT (memtype=DATA).
NOTE: The data set DEST2.INFANT has 149 observations and 6 variables.
NOTE: Copying HEALTH.NAMES to DEST2.NAMES (memtype=DATA).
NOTE: The data set DEST2.NAMES has 7 observations and 4 variables.
NOTE: Copying HEALTH.PERSONL to DEST2.PERSONL (memtype=DATA).
NOTE: The data set DEST2.PERSONL has 148 observations and 11 variables.
NOTE: Copying HEALTH.PHARM to DEST2.PHARM (memtype=DATA).
NOTE: The data set DEST2.PHARM has 6 observations and 3 variables.
NOTE: Copying HEALTH.POINTS to DEST2.POINTS (memtype=DATA).
NOTE: The data set DEST2.POINTS has 6 observations and 6 variables.
NOTE: Copying HEALTH.POSTDRUG to DEST2.POSTDRUG (memtype=CATALOG).
NOTE: Copying HEALTH.PRENAT to DEST2.PRENAT (memtype=DATA).
NOTE: The data set DEST2.PRENAT has 149 observations and 6 variables.
NOTE: Copying HEALTH.RESULTS to DEST2.RESULTS (memtype=DATA).
NOTE: The data set DEST2.RESULTS has 10 observations and 5 variables.
NOTE: Copying HEALTH.SLEEP to DEST2.SLEEP (memtype=DATA).
NOTE: The data set DEST2.SLEEP has 108 observations and 6 variables.
NOTE: Copying HEALTH.SUR to DEST2.SUR (memtype=DATA).
NOTE: The data set DEST2.SUR has 3 observations and 6 variables.
NOTE: Copying HEALTH.SYNDROME to DEST2.SYNDROME (memtype=DATA).
NOTE: The data set DEST2.SYNDROME has 46 observations and 8 variables.
NOTE: Copying HEALTH.TRAIN to DEST2.TRAIN (memtype=DATA).
NOTE: The data set DEST2.TRAIN has 7 observations and 2 variables.
NOTE: Copying HEALTH.WGHT to DEST2.WGHT (memtype=DATA).
NOTE: The data set DEST2.WGHT has 83 observations and 13 variables.
```

Example 2: Saving SAS Files from Deletion

Procedure features:

SAVE statement option:

MEMTYPE=

This example uses the SAVE statement to save some SAS files from deletion and to delete other SAS files.

Program

The SAS system option SOURCE writes the programming statements to the SAS log.

```
options pagesize=40 linesize=80 nodate pageno=1 source;


libname elder 'SAS-data-library';
```

LIBRARY= specifies the procedure input library.

```
    proc datasets lib=elder;
```

The SAVE statement saves the data sets CHRONIC, AGING, and CLINICS and deletes all other SAS files (of all types) in the ELDER library. MEMTYPE=DATA is necessary because the ELDER library has a catalog named CLINICS and a data set named CLINICS.

```
       save chronic aging clinics / memtype=data;
    run;
```

SAS Log

(UNIX Environment)

```
8   options pagesize=40 linesize=80 nodate pageno=1 source;
9
10      proc datasets lib=elder;
                              -----Directory-----

Libref:            ELDER
Engine:            V8
Physical Name:     UNIX-pathname
File Name:         UNIX-pathname
Inode Number:      Inode-name
Access Permission: rwxr-xr-x
Owner Name:        UNIX-userid
File Size (bytes): 2048


                                    File
               #  Name      Memtype    size  Last modified
               -------------------------------------------------
               1  AGING     DATA      12288   29JAN1999:13:29:33
               2  ALCOHOL   DATA      12288   29JAN1999:13:29:33
               3  BACKPAIN  DATA      12288   29JAN1999:13:29:33
               4  CHRONIC   DATA      12288   29JAN1999:13:29:33
               5  CLINICS   CATALOG   24576   29JAN1999:13:29:33
               6  CLINICS   DATA      12288   29JAN1999:13:29:33
               7  DISEASE   DATA      12288   29JAN1999:13:29:33
               8  GROWTH    DATA      12288   29JAN1999:13:29:33
               9  HOSPITAL  CATALOG   24576   29JAN1999:13:29:33
11
12         save chronic aging clinics / memtype=data;
13      run;
NOTE: Saving ELDER.CHRONIC (memtype=DATA).
NOTE: Saving ELDER.AGING (memtype=DATA).
NOTE: Saving ELDER.CLINICS (memtype=DATA).
NOTE: Deleting ELDER.ALCOHOL (memtype=DATA).
NOTE: Deleting ELDER.BACKPAIN (memtype=DATA).
NOTE: Deleting ELDER.CLINICS (memtype=CATALOG).
NOTE: Deleting ELDER.DISEASE (memtype=DATA).
NOTE: Deleting ELDER.GROWTH (memtype=DATA).
NOTE: Deleting ELDER.HOSPITAL (memtype=CATALOG).
```

Example 3: Modifying SAS Data Sets

Procedure features:

PROC DATASETS statement option:

NOLIST

FORMAT statement

INDEX CREATE statement options:

NOMISS

UNIQUE

INFORMAT statement

LABEL statement

MODIFY statement options:

LABEL=

READ=

SORTEDBY=

RENAME statement

This example modifies two SAS data sets using the MODIFY statement and statements subordinate to it. Example 4 on page 399 shows the modifications to the GROUP data set.

Tasks include

- modifying SAS files
- labeling a SAS data set
- adding a READ password to a SAS data set
- indicating how a SAS data set is currently sorted
- creating an index for a SAS data set
- assigning informats and formats to variables in a SAS data set
- renaming variables in a SAS data set
- labeling variables in a SAS data set.

Program

The SAS system option SOURCE writes the programming statements to the SAS log.

```
options pagesize=40 linesize=80 nodate pageno=1 source;


libname health 'SAS-data-library';
```

LIBRARY= specifies HEALTH as the procedure input library. NOLIST suppresses the directory listing for the HEALTH data library.

```
proc datasets library=health nolist;
```

LABEL= adds a data set label to the data set GROUP. READ= assigns GREEN as the read password. The password appears as Xs in the SAS log. SAS issues a warning message if you specify a level of password protection on a SAS file that does not include alter protection. SORTEDBY= specifies how the data are sorted.

```
    modify group (label='Test Subjects' read=green sortedby=lname);
```

The INDEX CREATE statement creates the composite index VITAL on the variables BIRTH and SALARY for the GROUP data set. NOMISS excludes all observations that have missing values for BIRTH and SALARY from the index. UNIQUE specifies that the index is created only if each observation has a unique combination of values for BIRTH and SALARY.

```
       index create vital=(birth salary) / nomiss unique;
```

The INFORMAT and FORMAT statements assign an informat and format, respectively, to the BIRTH variable.

```
      informat birth date7.;
      format birth date7.;
```

The LABEL statement assigns a label to the variable SALARY.

```
      label salary='current salary excluding bonus';
```

The MODIFY statement names the data set to modify. The RENAME statement renames the variable OXYGEN to INTAKE. The LABEL statement assigns a label to the variable INTAKE.

```
    modify oxygen;
       rename oxygen=intake;
       label intake='Intake Measurement';
quit;
```

SAS Log

```
1   options pagesize=40 linesize=80 nodate pageno=1 source;
2
3   proc datasets library=health nolist;
4
5      modify group (label='Test Subjects' read=XXXXX sortedby=lname);
WARNING: The file HEALTH.GROUP.DATA is not ALTER protected.  It could be
         deleted or replaced without knowing the password.
6
7         index create vital=(birth salary) / nomiss unique;
NOTE: Composite index vital has been defined.
8
9         informat birth date7.;
10         format birth date7.;
11         label salary='current salary excluding bonus';
12
13      modify oxygen;
14        rename oxygen=intake;
NOTE: Renaming variable oxygen to intake.
15        label intake='Intake Measurement';
16  quit;
```

Example 4: Describing a SAS Data Set

Procedure features:

CONTENTS statement option:

DATA=

Other features:

SAS data set option:

READ=

This example shows the output from the CONTENTS statement for the GROUP data set. The output shows the modifications made to the GROUP data set in Example 3 on page 397.

Program

```
options pagesize=40 linesize=132 nodate pageno=1;


libname health 'SAS-data-library';
```

LIBRARY= specifies HEALTH as the procedure input library. NOLIST suppresses the directory listing for the HEALTH data library.

```
proc datasets library=health nolist;
```

DATA= specifies GROUP as the data set to describe. READ= gives read access to the GROUP data set. OUT= creates the output data set GRPOUT, which appears in "The OUT= Data Set" on page 385.

```
   contents data=group(read=green) out=grpout;
   title  'The Contents of the GROUP Data Set';
run;
```

Output

(UNIX Environment)

```
                                The Contents of the GROUP Data Set                                1

                                     The DATASETS Procedure

                Data Set Name: HEALTH.GROUP                       Observations:          148
                Member Type:   DATA                               Variables:             11
                Engine:        V8                                 Indexes:               1
                Created:       13:24 Friday, January 29, 1999     Observation Length:    96
                Last Modified: 13:34 Friday, January 29, 1999     Deleted Observations:  0
                Protection:    READ                               Compressed:            NO
                Data Set Type:                                    Sorted:                YES
                Label:         Test Subjects


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

Data Set Page Size:         8192
Number of Data Set Pages:   4
First Data Page:            1
Max Obs per Page:           84
Obs in First Data Page:     62
Index File Page Size:       4096
Number of Index File Pages: 2
Number of Data Set Repairs: 0
File Name:                  Unix-pathname
Release Created:            8.00.00B
Host Created:               HP-UX
Inode Number:               Inode-number
Access Permission:          rw-r--r--
Owner Name:                 UNIX-userid
File Size (bytes):          40960


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

            #    Variable    Type    Len    Pos    Format     Informat    Label
           ----------------------------------------------------------------------------------------
            9    BIRTH       Num       8      8    DATE7.     DATE7.
            4    CITY        Char     15     58    $.         $.
            3    FNAME       Char     15     43    $.         $.
```

The Contents of the GROUP Data Set 2

The DATASETS Procedure

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

#	Variable	Type	Len	Pos	Format	Informat	Label
10	HIRED	Num	8	16	DATE7.	DATE7.	
11	HPHONE	Char	12	79	$.	$.	
1	IDNUM	Char	4	24	$.	$.	
7	JOBCODE	Char	3	76	$.	$.	
2	LNAME	Char	15	28	$.	$.	
8	SALARY	Num	8	0	COMMA8.		current salary excluding bonus
6	SEX	Char	1	75	$.	$.	
5	STATE	Char	2	73	$.	$.	

-----Alphabetic List of Indexes and Attributes-----

#	Index	Unique Option	Nomiss Option	# of Unique Values	Variables
1	vital	YES	YES	148	BIRTH SALARY

-----Sort Information-----

Sortedby: LNAME
Validated: NO
Character Set: ASCII

Example 5: Concatenating Two SAS Data Sets

Procedure features:

APPEND statement options:

BASE=
DATA=
FORCE=

This example appends one data set to the end of another data set.

Input Data Sets

The BASE= data set, EXP.RESULTS.

The EXP.RESULTS Data Set 1

ID	TREAT	INITWT	WT3MOS	AGE
1	Other	166.28	146.98	35
2	Other	214.42	210.22	54
3	Other	172.46	159.42	33
5	Other	175.41	160.66	37
6	Other	173.13	169.40	20
7	Other	181.25	170.94	30
10	Other	239.83	214.48	48
11	Other	175.32	162.66	51
12	Other	227.01	211.06	29
13	Other	274.82	251.82	31

The DATA= data set, EXP.SUR, contains the variable WT6MOS, but the EXP.RESULTS data set does not.

The EXP.SUR Data Set 2

id	treat	initwt	wt3mos	wt6mos	age
14	surgery	203.60	169.78	143.88	38
17	surgery	171.52	150.33	123.18	42
18	surgery	207.46	155.22	.	41

Program

```
options pagesize=40 linesize=64 nodate pageno=1;
```

```
libname exp 'SAS-data-library';
```

LIBRARY= specifies EXP as the procedure input library. NOLIST suppresses the directory listing for the EXP library. The APPEND statement appends the DATA= data set, EXP.SUR, to the BASE= data set, EXP.RESULTS. FORCE causes the APPEND statement to carry out the append operation even though EXP.SUR has a variable that EXP.RESULTS does not. APPEND does not add the WT6MOS variable to EXP.RESULTS.

```
proc datasets library=exp nolist;
   append base=exp.results data=exp.sur force;
run;
```

PROC PRINT prints the data set.

```
proc print data=exp.results noobs;
   title 'The EXP.RESULTS Data Set';
run;
```

Output

Output 14.8

The EXP.RESULTS Data Set 2

ID	TREAT	INITWT	WT3MOS	AGE
1	Other	166.28	146.98	35
2	Other	214.42	210.22	54
3	Other	172.46	159.42	33
5	Other	175.41	160.66	37
6	Other	173.13	169.40	20
7	Other	181.25	170.94	30
10	Other	239.83	214.48	48
11	Other	175.32	162.66	51
12	Other	227.01	211.06	29
13	Other	274.82	251.82	31
14	surgery	203.60	169.78	38
17	surgery	171.52	150.33	42
18	surgery	207.46	155.22	41

Example 6: Aging SAS Data Sets

Procedure features:
AGE statement

This example shows how the AGE statement ages SAS files.

Program

The SAS system option SOURCE writes the programming statements to the SAS log.

```
options pagesize=40 linesize=80 nodate pageno=1 source;


libname daily 'SAS-data-library';
```

LIBRARY= specifies DAILY as the procedure input library. NOLIST suppresses the directory listing for DAILY.

```
    proc datasets library=daily nolist;
```

The AGE statement deletes the last SAS file in the list, DAY7, and then ages (or renames) DAY6 to DAY7, DAY5 to DAY6, and so on, until it ages TODAY to DAY1.

```
       age today day1-day7;
    run;
```

SAS Log

```
6  options pagesize=40 linesize=80 nodate pageno=1 source;
7
8     proc datasets library=daily nolist;
9
10        age today day1-day7;
11     run;
NOTE: Deleting DAILY.DAY7 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY6 to DAILY.DAY7 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY5 to DAILY.DAY6 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY4 to DAILY.DAY5 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY3 to DAILY.DAY4 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY2 to DAILY.DAY3 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY1 to DAILY.DAY2 (memtype=DATA).
NOTE: Ageing the name DAILY.TODAY to DAILY.DAY1 (memtype=DATA).
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Procedures Guide, Version 8,* Cary, NC: SAS Institute Inc., 1999. 1729 pp.

SAS® Procedures Guide, Version 8

ISBN 1–58025–482–9

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

1st printing, October 1999