CHAPTER

5

Locking SAS Data Objects

Introduction

The SAS/SHARE lock manager facility enables multiple clients to share the same SAS file at the same time. The lock manager must evaluate each incoming client request for access to SAS data objects against a set of complex locking rules. It grants or denies each new incoming request for access to the data object while monitoring the status of all other client activities. The lock manager grants access to the wanted data object by locking the data object, thus keeping out all other requests for the data object until the operation has completed or the lock has been cleared explicitly.

Note: The term *operation* refers to any SAS procedure, statement, or command. △

Audience

This chapter is intended primarily for applications developers but may also be of interest to end-users.

About the SAS Data Hierarchy and Locking

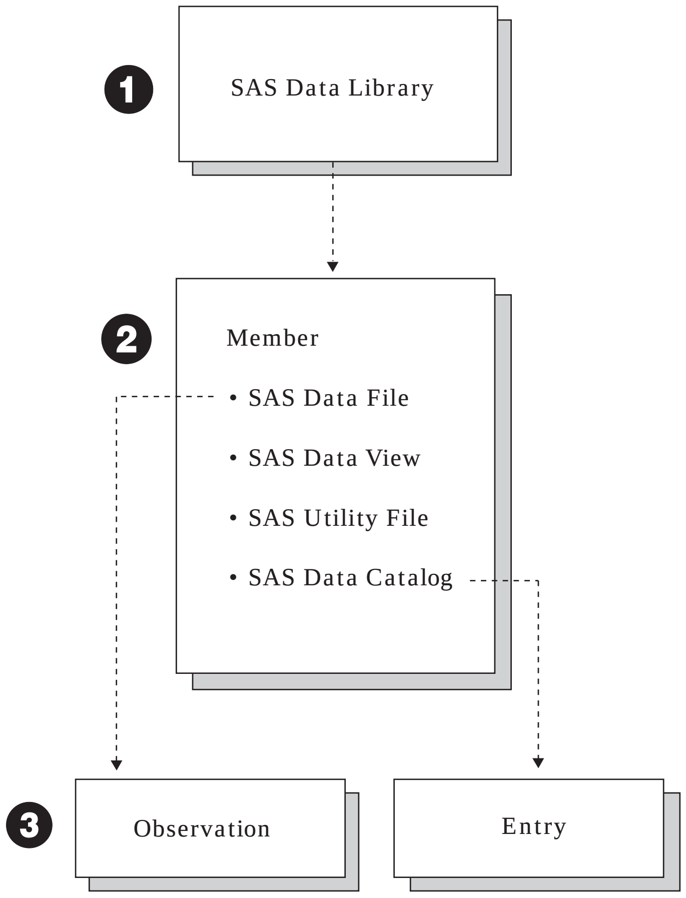
Knowing about the SAS data object hierarchy and locking concepts will help you to understand the behavior of explicit locks that you set by using the LOCK statement or a LOCK command and the conditions under which implicit locks are set automatically. Also explained are the effects of locking on other operations.

The SAS Data Hierarchy

When you perform a SAS operation, the SAS/SHARE server controls what data object is locked and how the data object is locked. This allows you to access the data objects and denies access to those data objects by other users for the duration of the operation.

Figure 5.1 on page 60 shows the SAS data object type hierarchy.

Figure 5.1 Hierarchy of SAS data object Types



❶ SAS data library
a collection of one or more SAS files that are recognized by SAS. Each file is a member of the library.

❷ Member
a file in a SAS data library that may be one of the following types:

- SAS data file
- SAS data view
- SAS utility file
- SAS data catalog.

SAS data file
a SAS data set that contains both the data values and the descriptor information. SAS data files are of member type DATA.

SAS data view
a SAS data file in which the descriptor information and observations are obtained from other files. SAS data views store only the information that is required to retrieve data values or descriptor information. SAS data views are of member type VIEW.

SAS utility file
a SAS file that stores information that is private to a component of SAS. Examples include SAS/ACCESS descriptors, MDDB (Multi-Dimensional Data Base) files, and DMDB (Data Mining Data Base) files.

SAS catalog
a SAS file that stores many different kinds of information in smaller units called entries. Some catalog entries contain system information such as key definitions. Other catalog entries contain application information such as window definitions, help windows, formats, informats, macros, or graphics output.

❸ Observation
often referred to as a record, the horizontal component of a SAS data file. An observation is a collection of data values associated with a single entity, such as a customer or a state. Each observation contains one data value for each variable in the data file.

❸Entry
a unit of information that is stored in a SAS catalog.

How SAS Data Objects Are Accessed and Used

The type of lock that a server sets on a member or an observation is affected by how the operation accesses and uses the SAS data object type. The ways to access a data object are:

input	to read data
update	to change the values of variables
output	to add new variables with values
utility	to change the header information of the file.

Each SAS operation has a default behavior for each object that is accessed and the way that the object is accessed. For example, given that the server engine permits observation locking and the observation is not already locked, the server can open and lock an observation in a data set. If the engine does not support observation locking, the engine locks the member (above the observation) instead.

The lowest hierarchical level at which data can be locked varies according to the engine that is used to access the data:

- The Version 7 engine (V7) and the Version 8 (default) engine (V8) allow locking at the library, member, and observation level.
- The V7TAPE, the V8TAPE, and other sequential engines do not support locking below the member level.
- Any engine that does not support access to SAS catalogs does not support locking at any level.
- View engine default-locking behavior is based on how the view is created; for example, by using the DATA step, by using the SQL procedure, or by using the ACCESS procedure, which is available through SAS/ACCESS. Likewise, the specific SAS/ACCESS engine that is used depends on the DBMS. See your SAS/ACCESS documentation for information about view engine default-locking behavior.

Table 5.1 on page 62 shows the combinations of objects that are locked, how objects are locked, and the effects on other client operations.

Table 5.1 Effects of Object Locking on Other Client Operations

What Object Is Locked	**How the Object Is Locked**		
	Input	**Update**	**Output**
Member	Other operations can read the data set but cannot open it for update or output.	No other operations can access the data set.	No other operations can access the data set.
Observation	Other operations can read or update the data set but cannot open it for output.	Other operations can read or update the data set but cannot open it for output.	No other operations can access the data set.

Types of Locks

There are two ways that a data object can be locked:

explicit lock
: one that you set with the LOCK statement or the LOCK command for exclusive access to the data object type. The behavior of the LOCK statement or the LOCK command is restricted by the server engine, the object that is being accessed, and the way that the object is accessed (for example, a data set is locked for writing). For details about using the LOCK statement or the LOCK command, see “Locking Objects Explicitly (LOCK Statement)” on page 63 and “Locking Explicitly in a SAS Window (LOCK Command)” on page 67.

implicit lock
: one that is automatically set on the data object type as required by the SAS operation. The operation has default locking requirements that are affected by two factors: the data object that is being accessed and the way that the object is accessed. For example, the DATA step with a MODIFY statement accesses an observation for update by default.

Regardless of the type of lock that is attempted, in order to lock a selected data object, the server must lock preceding levels of the hierarchy, as needed. This type of lock is also referred to as an *implicit lock*.

When you specify a data object in a LOCK statement, you set an *explicit lock* on that object. If you lock a lower-level object without explicitly locking the higher level or levels, SAS locks the higher level (or levels) automatically.

For example, when you explicitly lock a SAS data set (lower-level lock) but not the SAS data library (higher-level lock) that contains it, the data library is locked implicitly. An implicit lock gives you and other users shared access to the data library, even though you have exclusive access to the locked data set.

Locking Objects Explicitly (LOCK Statement)

A classic case in which explicit locks protect data while it is being updated is a multi-step SAS program. For example, consider a nightly update process that consists of a DATA step that removes observations that are no longer useful and then runs PROC SORT to sort the file and PROC DATASETS to re-build the file's indexes. No other users can be allowed to access the file between any of these steps because the SORT and DATASETS procedures will fail if they cannot acquire exclusive access to the file.

An explicit lock provides the needed protection. Before the first DATA step, execute a LOCK statement to acquire exclusive access to the file. If exclusive access cannot be obtained, the LOCK statement return code (&SYSLCKRC) indicates that, and the update program can re-schedule the update for a later time, or it can signal an operator or an action that its programmer thinks is appropriate. If the LOCK statement is successful, a user who attempts to access the file before the corresponding LOCK CLEAR statement executes (after the end of the PROC DATASETS step) will be denied access, and the batch update will proceed uninterrupted.

You can use the LOCK statement to obtain an explicit lock on the following data objects:

data library

data set

catalog

catalog entry.

When you use a LOCK statement, you have exclusive access to the data object. No other clients can read or write to a data object that you have locked by using this statement. You cannot lock a data object that another client has open.

When you use a LOCK statement to lock a data object, you can open that data object as often as you want and in any mode that you want. For example, you can create, replace, update, or read the object, as long as your PROC or DATA step does not conflict with what is allowed by the engine that the server uses to access the data object. You must first access a SAS data library through a server before you can lock that library or any data object in it.

The syntax for the LOCK statement follows:

LOCK *libref*<*.member-name*<*.member-type*
  | *.entry-name.entry-type*>> <LIST | CLEAR>;

The LOCK statement takes the following arguments:

libref
: is a valid SAS name that serves as a symbolic link to a SAS data library.

member-name
: is a valid SAS name that specifies a member of the referenced SAS data library.

member-type
: is the type of the SAS file to be locked. Valid values are DATA, VIEW, and CATALOG. The default value is DATA.

  If *member-type* is omitted or is specified as the value DATA or VIEW, two locks are obtained: one on *libref.member-name*.DATA and the other on *libref.member-name*.VIEW.

entry-name
: is the name of the catalog entry to be locked.

entry-type
: is the type of the catalog entry to be locked.

LIST
: writes to the SAS log whether the specified data object is locked and by whom. This argument is optional.

CLEAR
: releases a lock on the specified data object that was acquired in your SAS session by use of the LOCK statement. This argument is optional.

  For details about releasing locks, see "Clearing an Explicit Lock" on page 65.

Locking a SAS Data Library

This statement locks the SAS data library that is referenced by MYLIB.

```
lock mylib;
```

Locking the library prevents other users from reading, updating, or deleting existing SAS files in the library or from creating new SAS files in the library. The lock also prevents other users from obtaining a list of files in the library. It does not prevent users from issuing LIBNAME statements to access the library, but it does prevent them from using SAS files in the library while it is locked.

Locking a SAS Data Set

These statements lock the SAS data set FUEL that is referenced by MYLIB. These three statements are equivalent to each other.

```
lock mylib.fuel;
lock mylib.fuel.data;
lock mylib.fuel.view;
```

Locking a SAS data set (a SAS data file or a SAS data view) prevents other users from creating, reading, updating, deleting, or renaming the SAS data file. Locking a SAS data view prevents other users from creating, reading, deleting, renaming, or interpreting the view.

Since Release 6.06 of SAS software, a SAS data set can be either a SAS data file (member type DATA) or a SAS data view (member type VIEW). In most SAS programs, it does not matter whether the data comes from a SAS data file or a data view.

Because of this transparency in users' SAS programs, it is important to lock both the SAS data file and the SAS data view by the same name at the same time. When you execute the LOCK statement on one of these data sets, it automatically locks both of them. In the preceding example, the server locks the SAS data file MYLIB.FUEL.DATA and the SAS data view MYLIB.FUEL.VIEW. For more information about SAS data sets, see *SAS Language Reference: Concepts*.

CAUTION:

The LOCK statement does not affect the source data of a data view. The LOCK statement does not prevent a SAS data view's underlying SAS file (or files) from being read or updated by a SAS library engine or by a SAS view engine when a different view is interpreted in the *server* SAS session. △

Locking a SAS Catalog

This statement locks the member MYCAT in the library SCLLIB. MYCAT is a SAS catalog, as indicated by the member type CATALOG.

```
lock scllib.mycat.catalog;
```

Locking a member of type CATALOG prevents other users from creating, deleting, or renaming the catalog, or listing the entries in the catalog. It also prevents other users from creating, reading, updating, deleting, or renaming any of the entries in the catalog.

While your SAS catalog or catalogs are locked, you can update an application that uses many different catalog entries. For example, you can execute LOCK statements to ensure exclusive access to the catalogs that contain your application's entries. By doing this, you can be sure that no other users are executing your application while you are in the middle of updating its entries. After you have updated all the entries and tested your application, you clear the lock by using the LOCK statement and the CLEAR argument. This allows other users to gain access to your catalogs and to execute your application. For information about the CLEAR argument, see "Clearing an Explicit Lock" on page 65.

Locking a Catalog Entry

This statement locks the catalog entry JOHNCBT of type CMAP in the catalog SCLLIB.MYCAT.

```
lock scllib.mycat.johncbt.cmap;
```

Locking an entry in a catalog prevents other users from creating, reading, updating, or deleting that entry.

Clearing an Explicit Lock

How you clear an explicit lock depends on the level in the data object hierarchy at which the lock was obtained. You can use one of three methods to clear locks:

- Explicitly lock and unlock each data object that you access.
- Explicitly lock lower-level data objects and unlock the higher-level data objects, which implicitly unlocks its lower-level objects.
- Explicitly lock a higher-level data object that contains multiple lower-level data objects that you want to access. Consequently, you can clear the single higher-level lock after you have finished accessing the lower-level objects.

Explanations of these methods follow.

Explicitly Locking and Unlocking Each Data Object

You obtain an explicit lock on a specific data object and you clear each lock individually. Examples follow:

```
lock educlib.mycat.choice1.menu;
lock educlib.mycat.choice2.menu;
/* Update the two catalog entries */
/* as needed.                      */
lock educlib.mycat.choice1.menu clear;
lock educlib.mycat.choice2.menu clear;
```

The first LOCK statement sets implicit locks on the SAS data library EDUCLIB and on the SAS catalog EDUCLIB.MYCAT. It then sets an explicit lock on the catalog entry EDUCLIB.MYCAT.CHOICE1.MENU. Because the user already has implicit locks on the catalog and library, the second LOCK statement does not set additional implicit locks before it sets an explicit lock on the catalog entry EDUCLIB.MYCAT.CHOICE2.MENU.

The first LOCK statement that contains the CLEAR argument releases the explicit lock on the catalog entry CHOICE1.MENU, but it does not clear the implicit locks because an entry in the catalog is still locked. The second LOCK statement that contains the CLEAR argument releases the explicit lock on the catalog entry CHOICE2.MENU. Because no catalog entries remain locked, the CLEAR argument releases the implicit lock on the SAS catalog EDUCLIB.MYCAT. Also, because no members of the library are locked, it clears the implicit lock on the SAS library EDUCLIB.

Unlocking One Higher-Level Data Object for Multiple Lower-Level Objects

You set explicit locks on data objects at low levels. Clear a higher-level implicit lock to cause all of the lower-level explicit locks to be cleared automatically. Examples follow:

```
lock educlib.mycat.choice1.menu;
lock educlib.mycat.choice2.menu;
/* Update the two catalog entries */
/* as needed.                      */
lock educlib.mycat clear;
```

The first LOCK statement sets implicit locks on the SAS data library EDUCLIB and on the SAS catalog EDUCLIB.MYCAT. It then sets an explicit lock on the catalog entry EDUCLIB.MYCAT.CHOICE1.MENU. Because the user already has implicit locks on the catalog and the library, the second LOCK statement does not set additional implicit locks before it sets an explicit lock on the catalog entry EDUCLIB.MYCAT.CHOICE2.MENU.

The LOCK statement that contains the CLEAR argument releases the explicit locks on both catalog entries and clears the implicit lock on the SAS catalog. Because no members of the library remain locked, it also clears the implicit lock on the SAS library.

Locking One Higher-Level Data Object for Access to Multiple Lower-Level Data Objects and Clearing the Single Higher-Level Lock

To update several lower-level data objects without individually locking each one when all these data objects fall under a single higher-level data object, you can lock the higher-level data object to prevent access by other users to all of the data objects that are included under the higher-level data object.

However, you may need to clear the lock on the higher-level data object before you are finished with your work. For example, a co-worker wants to work on other lower-level data objects under the same higher-level data object. In this case, you can acquire explicit locks on the lower-level data objects that you need, and then clear your explicit lock on the higher-level data object. You will retain an implicit lock on the higher-level data object as long as you have lower-level data objects locked, for example,

```
lock educlib;
/* Update various library members  */
/* and catalog entries.            */
```

Now, one of your co-workers needs to work on some SAS files in the library EDUCLIB that you are not updating. So, you lock the SAS files in the library EDUCLIB that you do need, as in the following example:

```
lock educlib.mycat.catalog;
lock educlib.mydata1;
lock educlib.mydata2;
```

Then, you clear your explicit lock on the library to allow your co-worker to use other members of the library:

```
lock educlib clear;
```

You retain an implicit lock on the library because you hold explicit locks on three SAS files in the library.

You continue to update entries in the SAS catalog EDUCLIB.MYCAT and the SAS data sets EDUCLIB.MYDATA1 and EDUCLIB.MYDATA2 that you have locked. After you finish your updates, you can issue one LOCK statement to clear your explicit locks on the three library members and to clear your implicit lock on the library as follows:

```
  lock educlib clear;
```

Listing Locks

You can list to the SAS log the status of the specified data object - whether it is locked and by whom. The format for listing lock status follows:

data-object is *status* by *whom*

Example:

```
lock educlib.mycat.catalog list;
EDUCLIB is locked by sasuser
```

Return Codes for the LOCK Statement

The SAS macro variable SYSLCKRC contains the return code from the LOCK statement. The following actions result in a non-zero value in SYSLCKRC:

- You try to lock a data object but cannot obtain the lock (for example, the data object was already in use or is locked by another user).
- You use a LOCK statement with the LIST argument to list a lock you do not have.
- You use a LOCK statement with the CLEAR argument to release a lock you do not have.

For more information about the SYSLCKRC SAS macro variable, see *SAS Guide to Macro Processing*.

Locking Explicitly in a SAS Window (LOCK Command)

The LOCK command provides a convenient way to lock data objects within a SAS window. As in the LOCK statement, you can use the LOCK command to obtain an explicit lock on the following data objects:

data library

data set

catalog

catalog entry.

You can specify the name of the data object that is to be locked on the command line of a window, such as the Program Editor window in the SAS Display Manager System.

Note: You must first access a SAS data library through a server before you can lock that library or any data object in it. △

The syntax for the LOCK command is

LOCK *libref<.member-name<.member-type*
| .entry-name.entry-type>><LIST | CLEAR>;

libref
is a valid SAS name that serves as a symbolic link to a SAS data library.

member-name
is a valid SAS name that specifies a member of the referenced data library.

member-type
is the type of the SAS file to be locked. Valid values include DATA, VIEW, and CATALOG. The default is DATA.
If you omit *member-type* or if you specify either the value DATA or VIEW, two locks are obtained automatically: one on *libref.member-name*.DATA and one on *libref.member-name*.VIEW.

entry-name.entry-type
is the name and type of the SAS catalog entry to be locked.

LIST
writes to the SAS log whether the specified data object is locked and by whom. This argument is optional.

CLEAR
releases a lock on the specified data object that was acquired in your SAS session by use of the LOCK command. This argument is optional.
For details about releasing locks, see “Clearing an Explicit Lock” on page 65.

Setting and Clearing Locks by Using the LOCK Command

You can issue the LOCK command in any SAS window. It works exactly like the LOCK statement. For details about the LOCK statement, see Chapter 11, “The LOCK Statement and the LOCK Command,” on page 113.

Output 5.1 on page 68 shows that the catalog MAPSLIB.MAPSCAT.ASIAMAP.CMAP has successfully been locked. From the Program Editor window, the LOCK command was issued to obtain a lock on another catalog- MAPSLIB.MAPSCAT.EUROMAP.CMAP.

Output 5.1 Locking a Catalog Entry

```
LOG
 Command ===>

 1   LIBNAME MAPSLIB 'SASXYZ.SHRTEST.SASDATA' SERVER=SHARE1;
 NOTE: Libref MAPSLIB was successfully assigned as follows:
       Engine:        REMOTE
       Physical Name: SASXYZ.SHRTEST.SASDATA
 NOTE: MAPSLIB.MAPSCAT.EUROMAP.CMAP is now locked for
       exclusive access by you.
```

```
PROGRAM EDITOR
 Command ===> LOCK  MAPSLIB.MAPSCAT.EUROMAP.CMAP

 00001
 00002
 00003
 00004
 00005
 00006
```

In Output 5.2 on page 69, you can see that MAPSLIB.MAPSCAT.EUROMAP.CMAP was successfully locked. From the Program Editor window, the LOCK command that contains the CLEAR argument was issued to release the lock on catalog MAPSLIB.MAPSCAT.EUROMAP.CMAP.

Output 5.2 Releasing a Lock on a Catalog Entry

```
LOG
 Command ===>

 1   LIBNAME MAPSLIB 'SASXYZ.SHRTEST.SASDATA' SERVER=SHARE1;
 NOTE: Libref MAPSLIB was successfully assigned as follows:
       Engine:        REMOTE
       Physical Name: SASXYZ.SHRTEST.SASDATA
 NOTE: MAPSLIB.MAPSCAT.EUROMAP.CMAP is now locked for
       exclusive access by you.
```

```
PROGRAM EDITOR
 Command ===> LOCK  MAPSLIB.MAPSCAT.EUROMAP.CMAP  CLEAR

 00001
 00002
 00003
 00004
 00005
 00006
```

In Output 5.3 on page 70, you can see that MAPSLIB.MAPSCAT.EUROMAP.CMAP was successfully unlocked. The log also reports who clears the lock. In this example, the user who set and cleared the lock is referred to as “you.”

Output 5.3 SAS Log Message after the Lock Has Been Cleared

```
LOG
 Command ===>

 1   LIBNAME MAPSLIB 'SASXYZ.SHRTEST.SASDATA' SERVER=SHARE1;
 NOTE: Libref MAPSLIB was successfully assigned as follows:
       Engine:        REMOTE
       Physical Name: SASXYZ.SHRTEST.SASDATA
 NOTE: MAPSLIB.MAPSCAT.EUROMAP.CMAP is now locked for
       exclusive access by you.
 NOTE: MAPSLIB.MAPSCAT.EUROMAP.CMAP is no longer locked
       for exclusive access by you.
```

How Implicit Locking Works in SAS Program Steps

The following example shows the effect of implicit locking when two clients, John and Maria, share access to the SAS data set FUEL in their respective PROC FSEDIT sessions at the same time. Maria is updating observation 6.

John terminates his FSEDIT session to do some data analysis. He wants a sorted report of fuel inventory data, so he submits statements to sort and print the data set FUEL. Output 5.4 on page 71 shows the SAS log for this portion of John’s session:

Output 5.4 Attempt to Sort a Locked Data Set

```
Command ===>

3   PROC SORT DATA=DATALIB.FUEL;
4   BY AREA;
5   RUN;

ERROR: You cannot open DATALIB.FUEL.DATA for output access
       with member-level control because DATALIB.FUEL.DATA
       is in use by FSEDIT.
NOTE: The SAS System stopped processing this step because
      of errors.
NOTE: The PROCEDURE SORT used 0.03 CPU seconds and 3969K.

6   PROC PRINT DATA=DATALIB.FUEL;
7   BY AREA;
8   RUN;

ERROR: Data Set DATALIB.FUEL is not sorted in ascending
       sequence. The current by-group has AREA = TEX1
       and the next by-group has AREA = TEX2.
NOTE: The SAS System stopped processing this step because
      of errors.
NOTE: The PROCEDURE PRINT used 0.03 CPU seconds and 4068K.
```

For details about error message formats, see “Locking Message Format” on page 74.

Because the OUT= option is not specified in the PROC SORT statement, the process defaults to the data set named by the DATA= option and the SORT procedure tries to replace the SAS data set. However, because Maria’s FSEDIT session has the data set open for update, the SORT procedure cannot open it for output.

The SAS log shows that the PROC PRINT step executes because the PRINT procedure opens its input data set with observation-level control. However, the PRINT procedure terminates prematurely because the data set is not sorted properly. Note that even if the data set were in sorted order when John terminated PROC FSEDIT, Maria could have changed the value of AREA in one or more observations so that the data set would no longer be sorted properly when the PRINT procedure executed.

To avoid the conflict and ensure that he gets the report he wants, John can use the OUT= option to write a copy of the sorted data set into his WORK library, as shown in the following example:

```
proc sort data=datalib.fuel out=fuel;
   by area;
run;
```

John avoids a conflict because this PROC SORT statement opens the data set DATALIB.FUEL only for input with observation-level control.

John can then use the PRINT procedure to display the temporary data set WORK.FUEL.

Default Circumstances for Selected SAS Operations

Knowledge of the default data objects and how they are accessed will help you to anticipate the behavior of certain operations when you write your application or issue SAS statements in interactive mode. Table 5.2 on page 72 presents the default circumstances for some frequently used SAS operations.

Table 5.2 Default Circumstances for Selected SAS Operations

Typical Statements and Commands	How Object Is Locked	What Object Is Locked
DATA step		
DATA statement with MODIFY statement	update	observation
DATA statement without MODIFY statement	output	member
SET statement without POINT= and KEY= options	input	observation
SET statement with POINT= and KEY= options	input	member
MERGE statement	input	observation
MODIFY statement without POINT= and KEY= options	update	observation
MODIFY statement with POINT= and KEY= options	update	member
UPDATE statement	input	observation
Procedures		
APPEND procedure		
BASE= option	update	observation
DATA= option	input	observation
COPY procedure		
IN= option	input	observation
IN= option with MOVE option	output	observation
OUT= option	output	member
FSBROWSE procedure		
DATA= option	input	observation
FSEDIT procedure		
DATA= option	update	observation
FSVIEW procedure		
DATA= option without EDIT option	input	observation
DATA= option with EDIT option	update	observation
PRINT procedure		
DATA= option	input	observation
UNIFORM= option	input	member
SORT procedure		
DATA= option	input	observation

Typical Statements and Commands	How Object Is Locked	What Object Is Locked
OUT= option	output	member
SQL procedure		
CREATE TABLE statement	output	member
DELETE statement	update	observation
INSERT statement	update	member
UPDATE statement	update	member

Changing the Data Set Option Default Object

In some cases, you can change the SAS data set option default object. When the syntax of a statement or a command allows you to specify SAS data set options, you can use the CNTLLEV= option to override the default object and specify the object that you want, instead.

For example, for a SET statement that contains the POINT= option, you can change the default from member to observation by specifying the CNTLLEV= data set option:

```
set datalib.fuel (cntllev=rec) point=obsnum;
```

Note: If you make this change, the values in a specific observation may differ each time that you read the observation. △

Note: The value, rec (for record), means the same as observation. △

You can also change observation to member. You might do this to ensure that a data set does not change while you are processing it. For example, if you use a SET statement with a BY statement and you cannot use an index to retrieve the observations in sorted order, you can use the CNTLLEV= option to re-set observation to member.

```
set datalib.fuel (cntllev=mem);
by area;
```

In some cases, you cannot override the default setting because the statement or the command requires it. For example, a DATA statement requires a member setting when the MODIFY statement is omitted from the DATA step. Without the MODIFY statement, the data set that is specified in the DATA statement must be opened for output. Thus, even if you specify CNTLLEV=REC in such a DATA statement, the DATA step tries to set the object as member but will fail if other operations are accessing the data set.

Note: Be careful when using the CNTLLEV= option for a procedure. Some procedures make multiple passes through an input data set and require that the data remains the same to guarantee the integrity of the output. If they have this requirement, such procedures issue a warning but allow their objects to be re-set with the CNTLLEV= option. △

For details about the syntax of the CNTLLEV= option in the SET statement, see Chapter 12, “The CNTLLEV= Data Set Option,” on page 117.

Locking Message Format

SAS/SHARE delivers an informational or error message if you attempt to access a data object that is already in use or is locked by another operation. The message takes the following form:

object is *status* by *whom*

object
SAS data library | SAS data member | SAS data file observation or catalog

status
locked for exclusive access| in use | not locked

whom
you | user *user*(*server-connection-number*) |
n other users of this server|
task FSEDIT (*server-connection-number*)

The messages explain the status of the data object that is being accessed. To recover, you usually must wait until the data object is available or find out when the data object will be available by talking to the person who has locked the object.

Examples of typical messages follow:

```
NOTE: SASUSER.MYLIB is not locked or in use by you,
but is locked for exclusive access by user sasuser(1).
```

The SAS data library that is referenced by MYLIB is locked by userid SASUSER(1). A lock on a library prevents other clients from reading, updating, or deleting existing SAS files in the library or from creating new SAS files in the library. The lock also prevents other clients from obtaining a list of files in the library. It does not prevent client operations from issuing LIBNAME statements to access the library, but it does prevent them from using SAS files in the library while it is locked.

You must wait for userid SASUSER(1) to unlock the library before you can use it.

```
MYLIB.MYCAT.CATALOG is not locked or in use by you,
but is in use by 2 other users of this server.
```

Because two users are already accessing the MYCAT member in the MYLIB library, you can infer that no locks have been set on the catalog, and that client operations are reading catalog entries or adding entries to the catalog. Although you may browse the catalog or add entries to the catalog, you cannot attempt to lock the catalog until there are no client operations using it.

```
MYLIB.MYCAT.MYCATENTRY.CMAP is not locked by sasuser(1).
```

The catalog entry MYCATENTRY of type CMAP in the catalog MYLIB.MYCAT is not locked by userid SASUSER(1). This message results when user SASUSER attempts to unlock a catalog entry that another client has locked.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/SHARE User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 247.

SAS/SHARE User's Guide, Version 8

ISBN 1–58025–478–0

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

1st printing, September 1999