THE POWER TO KNOW.

# SAS/SHARE® 9.3 User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS/SHARE® 9.3 User's Guide*. Cary, NC: SAS Institute Inc.

**SAS/SHARE® 9.3 User's Guide**



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

1st electronic book, July 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

# Contents

# About This Book

## Syntax Conventions for the SAS Language

### *Overview of Syntax Conventions for the SAS Language*

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

### *Syntax Components*

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=).

keyword
specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In the following examples of SAS syntax, the keywords are the first words in the syntax:

**CHAR** (*string, position*)

**CALL RANBIN** (*seed, n, p, x*);

**ALTER** (*alter-password*)

**BEST** *w*.

**REMOVE** <*data-set-name*>

In the following example, the first two words of the CALL routine are the keywords:

**CALL RANBIN**(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

**DO**;
*... SAS code ...*

**END;**

Some system options require that one of two keyword values be specified:

**DUPLEX** | **NODUPLEX**

argument
specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed between angle brackets.

In the following example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

**CHAR** (*string, position*)

Each argument has a value. In the following example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:`x=char('summer', 4);`

In the following example, *string* and *substring* are required arguments, while *modifiers* and *startpos* are optional.

**FIND**(*string, substring* <,*modifiers*> <,*startpos*>

*Note:* In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

## Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
identifies SAS keywords such as the names of functions or statements. In the following example, the keyword ERROR is written in uppercase bold:

**ERROR**<*message*>;

UPPERCASE
identifies arguments that are literals.

In the following example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

**CMPMODEL** = BOTH | CATALOG | XML

*italics*
identifies arguments or values that you supply. Items in italics represent user-supplied values that are either one of the following:

- nonliteral arguments In the following example of the LINK statement, the argument *label* is a user-supplied value and is therefore written in italics:

  **LINK** *label*;

- nonliteral values that are assigned to an argument

  In the following example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  **FORMAT** = *variable-1* <, ..., *variable-nformat*><DEFAULT = *default-format*>;

Items in italics can also be the generic name for a list of arguments from which you can choose (for example, *attribute-list*). If more than one of an item in italics can be used, the items are expressed as *item-1, ..., item-n*.

## *Special Characters*

The syntax of SAS language elements can contain the following special characters:

=
: an equal sign identifies a value for a literal in some language elements such as system options.

  In the following example of the MAPS system option, the equal sign sets the value of MAPS:

  **MAPS** = *location-of-maps*

< >
: angle brackets identify optional arguments. Any argument that is not enclosed in angle brackets is required.

  In the following example of the CAT function, at least one item is required:

  **CAT** (*item-1 <, ..., item-n>*)

|
: a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

  In the following example of the CMPMODEL= system option, you can choose only one of the arguments:

  **CMPMODEL** = BOTH | CATALOG | XML

...
: an ellipsis indicates that the argument or group of arguments following the ellipsis can be repeated. If the ellipsis and the following argument are enclosed in angle brackets, then the argument is optional.

  In the following example of the CAT function, the ellipsis indicates that you can have multiple optional items:

  **CAT** (*item-1 <, ..., item-n>*)

'*value*' or "*value*"
: indicates that an argument enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

  In the following example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

  **FOOTNOTE** <*n*> <*ods-format-options* '*text*' | "*text*">;

;
: a semicolon indicates the end of a statement or CALL routine.

  In the following example each statement ends with a semicolon: `data namegame; length color name $8; color = 'black'; name = 'jack'; game = trim(color) || name; run;`

### References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks. If you use a logical name, you usually have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the association. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library*. Note that *SAS-library* is enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

# What's New in SAS/SHARE 9.3

## Overview

The following features are new or enhanced for SAS/SHARE 9.3:

- support for extended data set and library names with the integration of the new VALIDMEMNAME system option into SAS/SHARE
- full support for the THREADEDTCP option in the SERVER procedure
- documentation enhancements

## Extended SAS Library Names with VALIDMEMNAME

SAS/SHARE 9.3 offers you greater flexibility when naming SAS library members (data set names, data views, and item stores) with the integration of the new VALIDMEMNAME= system option. The option extends the allowable characters in a SAS member name to international characters, characters supported by third-party databases, some special characters, and characters comprising names of up to 32 bytes in length.

This option has been enhanced in SAS/SHARE to allow for client-side control of extended library names, regardless of what is set on the server. The value that you specify for this option when connecting with a SAS/SHARE 9.3 client will take precedence over the server setting for that session.

For more information about VALIDMEMNAME in SAS/SHARE, see "VALIDMEMNAME= System Option" on page 180.

## THREADEDTCP Option on PROC SERVER

The previously experimental option, THREADEDTCP, is now fully supported in the PROC SERVER statement for SAS 9.3. The option specifies whether the threaded version of the TCP access method and associated threaded infrastructure should be used

when TCP/IP communication is specified. Threading enables multiple, concurrent reception and transmission activity when the server runs on SMP hardware.

## Enhancements to Document

A section has been added that describes how SAS syntax is represented in this document. This section describes how various text styles (for example, uppercase bold) and special characters (for example, angle brackets) are used to represent various parts of SAS language syntax.

# Part 1

# Usage

*Chapter 1*

# Getting Started with SAS/SHARE



## SAS/SHARE: Learning to Use

### *Introduction*

If you're a new user of SAS/SHARE, this section provides answers to frequently asked questions (FAQs). A step-by-step example exercise shows the different types of

activities that are involved when using SAS/SHARE. Where applicable, operating environment specifics are provided.

*Note:* The following exercise is an example only and should not be used to set up production applications.

If you have some experience with SAS/SHARE and choose not to perform this exercise or read the FAQs, proceed to "Using SAS/SHARE Software" on page 19.

## Setting Up Your Operating Environment

The SAS sessions that you use in this exercise exchange data by using a communications access method. For this exercise, the TCP/IP communications access method is used for all operating environments. SAS/SHARE also supports other communications access methods, which are described in detail in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

To use the TCP/IP access method, you must verify that a SAS/SHARE server ID has been added to the TCP/IP SERVICES file. To find the location of the SERVICES file for your operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE* and the documentation for your TCP/IP software.

- If a SAS/SHARE server ID has already been added to the SERVICES file, proceed to the next section, "Invoking SAS for Client/Server Sessions (All New Users)" on page 5, and use an existing server ID from the SERVICES file in place of **&servername** in the remainder of this exercise.
- If a SAS/SHARE server ID has not already been added to the SERVICES file, edit the SERVICES file and add a line similar to the following:

  ```
  demoserv port-number/tcp  # SAS/SHARE server
  ```

  For *port-number*, specify a number that is not already specified in the SERVICES file.

  Execute the following statement in the server, the client, and the operator sessions:

  ```
  %let servername=demoserv;
  ```

- If you do not have authority to edit the SERVICES file, ask your server administrator to add **demoserv** to the SERVICES file. A server administrator ensures that SAS/SHARE servers are identified in the SERVICES file on each operating environment that accesses SAS/SHARE.
- The TCP/IP access method allows you to specify syntax that uses two consecutive underscores with a port number, in place of a server ID that has been defined in the client TCP/IP SERVICES file. As an alternative to editing the TCP/IP SERVICES file, execute the following statement in the server, the client, and the operator sessions:

  ```
  %let servername=_ _port-number;
  ```

  for *port-number* specify a number that is not already used in the TCP/IP SERVICES file. Do not space after the first underscore or the second underscore.

*Note:* If you choose to use a communications access method that is different from TCP/IP, some configuration of your operating environment might be required. For more information, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

### Invoking SAS for Client/Server Sessions (All New Users)

You need to invoke three SAS sessions for this example exercise. You can run these SAS sessions by logging on to three different machines or by logging on to the same machine three times.

*Operating Environment Information*
To invoke a SAS session for two clients and the SAS/SHARE server, use the commands that are specific to your operating environment.

*Note:* Arrange your SAS sessions so that you can see and use all of them while you are doing this exercise, because you will perform specific tasks in the Program Editor window of the user or server sessions.

In this example, one user logs on to the same machine three times.

- USER1 is john(1).
- USER2 is john(2).
- SERVER is demoserv.

*Note:* Be sure to issue the SAS statements that are appropriate for the specific SAS session. Each step in this example clearly identifies the session for which the instruction is intended.

### Starting a SAS/SHARE Server (All New Users)

*Note:* Usually, a server administrator starts the server so that it is available when end users and applications developers need to share SAS files. It is recommended that the server be run in non-interactive mode.

*z/OS Specifics*
For the z/OS operating environment, the server should be run in line mode.

*Note:* In this example exercise, the data is logged for a UNIX operating environment, and the TCP/IP communications access method is used. If you choose a different method, replace **`tcp`** in every occurrence of the COMAMID= option with the appropriate access method value.

1. In the SERVER session, submit the following statements from the Program Editor window:

```
options comamid=tcp;
libname demo (work);
proc server id=&servername authenticate=optional;
run;
```

   The LIBNAME statement associates a SAS library reference (libref) with a SAS library.

   The omission of the USER= and PASSWORD= options in the LIBNAME statement means that the SAS/SHARE client/server session is running unsecured.

   The COMAMID= option specifies the access method that is used to communicate between a client SAS session and the server. You must specify the COMAMID= option before you invoke PROC SERVER.

   PROC SERVER manages concurrent update access to SAS libraries and the members in those libraries. PROC SERVER runs in its own SAS session, which serves client SAS sessions by executing input and output requests to SAS libraries.

The value OPTIONAL for the AUTHENTICATE= option allows users with valid access permission to connect to a server without requiring verification. See "Ensuring That User IDs Are Valid" on page 38. For more information about the AUTHENTICATE= option, see the PROC SERVER statement.

2. Examine the SERVER Log window, which now contains information similar to this:

```
NOTE: Libref DEMO was successfully assigned as follows:
      Engine:        V9
      Physical Name: /local/u/john
1     options comamid=tcp;
2     libname demo (work);
3     proc server id=&servername authenticate=optional;
4     run;

30Apr2007:15:12:09.095 SAS server DEMOSERV started.
```

## Defining a SAS Library to a Server (All New Users)

When you access a SAS library through a server, your SAS session reads from and writes to that data library through the server instead of reading and writing directly to the library.

The first LIBNAME statement, which specifies a name for the server, connects your SAS session to that server. For a client session, you must specify the COMAMID= option before you try to connect to the server.

1. In the USER1 session, submit the following from the Program Editor window:

```
options comamid=tcp;
libname demo server=&servername;
```

*Note:* If you are connecting to a server on a remote operating environment, you must specify the network node name in the SERVER= option as follows:

```
server=network-node-name.&servername
```

See the TCP/IP chapter for your specific operating environment in *Communications Access Methods for SAS/CONNECT and SAS/SHARE* for more information.

Examine the USER1 Log window, which contains the following information:

```
NOTE: Libref DEMO was successfully assigned as follows:
      Engine:        REMOTE
      Physical Name: /local/u/john
1     options comamid=tcp;
2     libname demo server=&servername;
```

For convenience in this exercise, the libref DEMO is associated with the server library WORK. In SAS, the default name WORK means that the data files that are created are temporary.

2. Examine the SERVER Log window, which now contains information similar to the following lines about your connection to the server. The messages include the server name, the name of the server library that you specified, and the user identification in the form *user-ID(n)*, where *n* is the server connection number.

```
30Apr2007:15:16:46.521 User john(1) has connected to server demoserv.
30Apr2007:15:16:52.566 User john(1) has created "DMS Process"(1)
  under "Kernel"(0).
30Apr2007:15:16:59.079 Server library ('/local/u/john' V9) accessed as
  DEMO by user john(1).
```

### Creating a SAS Data Set (All New Users)

1. In the USER1 session, submit the following from the Program Editor window:

   ```
   data demo.test;
      do i=1 to 5;
         output;
      end;
   run;
   ```

   This DATA step creates a SAS data set that contains five observations and one variable that you will use in the remainder of this example. The Log window displays information about the DATA step and the name of the SAS data set that is opened for output and then closed.

2. Examine the SERVER Log window again.

```
30Apr2007:15:23:17.110 User john(1) has created "DATASTEP"(2)
  under "DMS Process"(1).
30Apr2007:15:23:20.719 DEMO.TEST.DATA(1) opened  for output via
  engine V9 by "DATASTEP"(2) of user john(1).
30Apr2007:15:23:26.835 DEMO.TEST.DATA(1) closed by "DATASTEP"(2)
  of user john(1).
30Apr2007:15:23:27.194 User john(1) has terminated "DATASTEP"(2)
  (under "DMS Process"(1)).
```

### Locking an Observation (All New Users)

1. In the USER2 session, submit the following from the Program Editor window:

   ```
   options comamid=tcp;
   libname demo server=&servername;
   proc fsedit data=demo.test;
   run;
   ```

   An FSEDIT window appears in the center of the screen. It shows the value **1** in the first observation.

   ```
   i:     1
   ```

   Examine the USER2 Log window, which contains the following information:

```
NOTE: Libref DEMO was successfully assigned as follows:
      Engine:        REMOTE
      Physical Name: /local/u/sasvcl
1    options comamid=tcp;
2    libname demo server=shr9;
3    proc fsedit data=demo.test;
4    run;
```

2. Examine the SERVER Log window, to which information similar to the following lines was added:

```
30Apr2007:15:29:39.116 User john(2) has connected to server demoserv.
30Apr2007:15:29:42.483 User john(2) has created "Process"(1)
  under "Kernel"(0).
30Apr2007:15:29:48.155 Server library ('/local/u/john' V9) accessed as
  DEMO by user john(2).
30Apr2007:15:29:54.124 User john(2) has created "FSEDIT"(2)
  under "DMS Process"(1).
30Apr2007:15:29:56.109 DEMO.TEST.DATA(1) opened for input/2 via
  engine V9 by "FSEDIT"(2) of user john(2).
30Apr2007:15:29:56.933 DEMO.TEST.DATA(1) reopened for update/R by
  "FSEDIT"(2) of user john(2).
```

   The FSEDIT procedure accesses the data set that was created by USER1 in the previous section. The first observation is currently locked by USER2 for update access.

3. In the FSEDIT window in the USER2 session, change the value in the first observation by placing the cursor over the value **1** and typing **5**, but do not save it.

   The FSEDIT window of USER2 now looks like this:

```
i:    5
```

### *Accessing a Locked Observation (All New Users)*

In the USER1 session, submit the following from the Program Editor window:

```
proc fsedit data=demo.test;
run;
```

PROC FSEDIT also accesses the data set that was created by USER1.

When the FSEDIT window appears, the following message is displayed because the first observation is already locked by the PROC FSEDIT statement in USER2's session:

```
WARNING: User john(2) (server connection 2) is using this observation.
```

USER1 cannot update the observation until after USER2 releases it. Notice that the value of **i** is still **1** because USER2 did not save the change in the previous step.

### *Releasing a Locked Observation (All New Users)*

In the USER2 session, close the FSEDIT window by selecting **File** ⇨ **Close** from the menu. This action releases the observation that was locked by USER2.

### *Retrying Access to a Locked Observation (All New Users)*

1. After the FSEDIT window in the USER2 session closes, return to the USER1 FSEDIT session, reread the observation by selecting **View** ⇨ **Observation Number** from the menu, and type **1** in the resulting pop-up window. Click **OK** and the USER1 FSEDIT window now looks like this:

   ```
   i:    5
   ```

   Notice that the observation was updated to reflect USER2's change from 1 to 5.

2. In the FSEDIT window in the USER1 session, change the value from **5** to **4**.

The USER1 FSEDIT window now looks like this:

```
i:    4
```

### Stopping the Server (All New Users)

*Note:* In the real world, servers are usually stopped by server administrators, not by end users.

For this example exercise, if you are an end user, stop the server and close all SAS sessions.

1. In the USER1 session, close the FSEDIT window by selecting **File** ⇨ **Close** from the menu.
2. Also, in the USER1 session, stop the server by submitting the following code from the Program Editor window:

   ```
   proc operate server=&servername;
   stop server;
   quit;
   ```

   Examine the USER1 Log window, which now contains the following information:

```
16   proc operate server=&servername;
PROC OPERATE is set to default server DEMOSERV.
================================================
17   stop server;
Default server DEMOSERV is now stopped.
PROC OPERATE was previously set to default server
DEMOSERV but is not set to any server now.
================================================
18   quit;
```

   *Note:* If you are not on the same machine as the server, you must specify the network node name in the SERVER= option in the PROC OPERATE statement:

   ```
   proc operate server=network_node_name.&servername;
   ```

3. In the SERVER Program Editor window, close the server session by submitting the following:

   ```
   endsas;
   ```

4. On the command lines of both the USER1 and USER2 Program Editor windows, close the user sessions by issuing the following command:

   ```
   bye
   ```

For SAS/SHARE end users, you have finished the example exercise. See "Frequently Asked Questions (FAQs) about SAS/SHARE " on page 12.

### Identifying the Server (Server Administrators and Applications Developers)

*Note:* Usually, the OPERATE procedure is used by a server administrator; sometimes an applications developer has responsibilities that include server administration.

The remainder of the steps in this section are mainly here for applications developers and server administrators who are continuing this exercise. In the USER2 session, submit the following from the Program Editor window:

```
proc operate server=&servername;
```

PROC OPERATE is an interactive procedure that is terminated by a QUIT statement. A RUN statement is not used or needed with a PROC OPERATE statement.

PROC OPERATE manages the execution of a SAS/SHARE server. You must identify which SAS/SHARE server you want to manage, even if there is only one server executing. If you are not on the same machine as the server, you must specify the network node name in the SERVER= option in the PROC OPERATE statement:

```
proc operate server=network_node_name.&servername;
```

Examine the USER2 Log window, which contains the following information:

```
proc operate server=&servername;
PROC OPERATE is set to default server DEMOSERV.
```

Usually, you should specify the COMAMID= option before using PROC OPERATE to connect to a server. If you know that you will use the default access method on your operating environment, you might omit the COMAMID= option. You do not need to specify a value for the COMAMID= option in this step because it was already specified for this SAS session in an earlier step. See "Locking an Observation (All New Users)" on page 7.

### *Viewing the Server Libraries (Server Administrators and Applications Developers)*

PROC OPERATE has several commands. You will use some of the commands in the next steps. All output generated by PROC OPERATE is displayed in the Log window.

In the USER2 session, submit the following from the Program Editor window:

```
display library _all_;
```

The DISPLAY LIBRARY command in the PROC OPERATE step displays information about the libref, status, the number of users, and the library name of all SAS libraries that have been defined to the server.

Examine the USER2 Log window.

```
*** PROBLEM*** Log window is missing in support.sas.com and in source file.
```

### *Viewing Information about Clients (Server Administrators and Applications Developers)*

In the USER2 session, submit the following from the Program Editor window:

```
display user _all_;
```

The DISPLAY USER command displays information about the user ID, the status, and the number of libraries that have been defined by each connected client.

Examine the USER2 Log window, which now contains the following information:

```
USER                    NUMBER OF
ID         STATUS       LIBRARIES
------------------------------
john      ACTIVE          0
john      ACTIVE          1
john      ACTIVE          1
==============================
7    display user _all_;
```

### Disconnecting Clients from the Server (Server Administrators and Applications Developers)

In the USER2 session, submit the following from the Program Editor window:

```
quiesce user 1 2;
```

The QUIESCE USER command gradually terminates a user's access to a server by denying new user requests for resources and moving the user from active status to stopped status. The user can continue the SAS program step or window that is currently in use but will not be able to use the server after that step terminates or after the window closes.

Users can be identified by user IDs or connection numbers. For example, user JOHN(1) can be quiesced by executing either of the following:

```
quiesce user 1;
quiesce user john;
```

### Examining the Server Log (Server Administrators and Applications Developers)

1. In the USER1 session, because the FSEDIT window is still opened, USER1 can still edit the data set that was created in an earlier step. See "Creating a SAS Data Set (All New Users)" on page 7. Close the USER1 FSEDIT window by selecting **File** ⇨ **Close** from the menu.
2. Examine the SERVER Log window, which displays information similar to the following:

```
30Apr2007:15:56:57.207 PROC OPERATE command from user john(3):
  QUIESCE USER 1 2;
30Apr2007:15:59:52.065 DEMO.TEST.DATA(1) closed by "FSEDIT"(3)
  of user john(1).
30Apr2007:15:00:02.161 User john(1) has terminated "FSEDIT"(3)
  (under "DMS Process"(1)).
```

### Accessing a Closed Server (Server Administrators and Applications Developers)

In the USER1 session, resubmit the following from the Program Editor window:

```
proc fsedit data=demo.test;
run;
```

Examine the USER1 Log window, which contains the following information:

```
10   proc fsedit data=demo.test;
You cannot open data set DEMO.TEST.DATA because user JOHN(1)
is quiesced on server DEMOSERV.
11   run;

NOTE: The SAS System stopped processing this step because of errors.
```

The messages in the Log window tell you that the attempt by USER1 to communicate with the server is rejected. Because USER1 is stopped, you cannot access the data set.

### *Stopping the Server (Server Administrators and Applications Developers)*

In the USER2 session, submit the following from the Program Editor window:

```
stop server;
quit;
```

The STOP SERVER command in the PROC OPERATE step terminates a server as quickly as possible. If users are connected to the server when you execute a STOP SERVER command, changes that they have not saved are lost. The QUIT command terminates PROC OPERATE in interactive mode.

### *Closing the SAS Sessions (Server Administrators and Applications Developers)*

1. In the SERVER Program Editor window, close the server session by submitting the following:

   ```
   endsas;
   ```

2. On the command lines of both the USER1 and USER2 Program Editor windows, close the user session by submitting the following:

   ```
   bye
   ```

## Frequently Asked Questions (FAQs) about SAS/SHARE

### *General Questions*

#### *What is SAS/SHARE software? Why would I use it?*

You use SAS/SHARE software in the following situations

- More than one user needs to update a SAS file (or several SAS files) at the same time.
- Users want to access SAS files on a server without having to use a separate SAS/CONNECT remote login for each user.

#### *Where can I read about SAS/SHARE software?*

You can read about SAS/SHARE in the following documents:

- This document, which explains SAS/SHARE software, describes the parts of the software, and applies to all operating environments. It also includes basic and detailed reference material for PROC SERVER, PROC OPERATE, the LIBNAME statement, and the LOCK statement.
- Also see *Communications Access Methods for SAS/CONNECT and SAS/SHARE* for information about using a communications access method to connect from a client session to a server session and for instructions to configure the access method.

### *Do people have to use a new SAS procedure to share their data?*

No. The users who add and maintain data continue to use the SAS procedures and windows that they already know: PROC FSEDIT, PROC APPEND, PROC FSVIEW, and so on.

Instead of requiring users to change the SAS tools they already know and use, SAS/SHARE takes advantage of the SAS Multi Engine Architecture to allow those SAS tools to access data through a "traffic cop" that is formally known as a SAS/SHARE server. A SAS/SHARE server allows many users to read and update data concurrently in one or multiple SAS files by tracking locks on observations, catalog entries, and SAS files.

### *Does each person responsible for maintaining data have to run an individual SAS/SHARE server?*

No. There are three roles that users assume with respect to SAS/SHARE:

End user
: reads, adds, and updates data.

Applications developer
: writes SAS programs used by the end users.

Server administrator
: makes sure SAS/SHARE servers are available to the end users.

The three roles can be performed by the same person, or one person might perform two roles, or each role might be assigned to a separate group of people.

It's not unusual for the same person to perform the tasks of an applications developer and a server administrator, for example, when the person who develops an application is responsible for the SAS/SHARE server or servers used by that application.

### *Three people? I hope this software doesn't require the effort of a large team of people.*

No, three roles. The three roles help organize the efforts so that shared maintenance of data is possible. In real life, the responsibilities of the various people involved in a project might overlap. Often, the same person who develops an application also maintains a SAS/SHARE server.

To help you keep track of how responsibilities usually are divided when multiple users need to update a SAS file at the same time, the remainder of this section answers the questions most frequently asked by end users, application developers, and server administrators.

## FAQs by End Users

### How do I get started with SAS/SHARE?

You use an application that someone else developed to read, add, or update data in one or in multiple SAS files. Occasionally, you find that an observation, a catalog entry, a file, or a library is locked by another user. If that happens, a message appears and you cannot modify the data. SAS/SHARE keeps track of which users have which data locked, so users cannot cause each other's changes to become mysteriously "lost."

### How can I find out if I'm accessing a SAS library through a SAS/SHARE server?

The SERVER= option is required in a LIBNAME statement (or, in SCL programs, any LIBNAME() function) for a library to be accessed through a SAS/SHARE server.

When a library is accessed through a server, the information that is displayed in the Log window about the LIBNAME statement shows you that the engine that was used to access the library is named REMOTE, and the physical name is a subdirectory accessed by the server SAS session.

Use the LIST option in a LIBNAME statement to obtain information about how a SAS library is defined to a SAS session. This information includes the following:

- the name of the server through which the library is accessed.
- the libref used by the server to refer to the library. (This libref might be the same as or different from the user's libref for that library.)
- the engine used in the server SAS session to read and write files in the library.
- the operating environment and machine type on which the server is running.

## FAQs by Applications Developers

### How do I get started with SAS/SHARE?

See "SAS/SHARE: Learning to Use" on page 3.

Topics of importance for applications developers include SAS library access, locking data objects, and SAS programming considerations. See "Writing End-User Applications to Access Shared Data" on page 43. For a sample SCL application, see "SAS Component Language (SCL) Application" on page 237. For complete details about locking, see "Locking SAS Data Objects" on page 59.

You might also find helpful information about how server administrators manage SAS/SHARE servers. See "Managing a SAS/SHARE Server (Server Administrators)" on page 29. For specific details about creating a SAS/SHARE server and setting SAS options to enhance performance and to establish logging parameters, by operating environment, see "Creating the SAS/SHARE Server Environment" on page 201.

### What do I have to do to a SAS library so that users can access it through a SAS/SHARE server?

You have to add a SERVER= option to each LIBNAME statement that a user will use to access the library.

You might want to predefine one or more libraries to a server. Include a LIBNAME statement for each library before executing the PROC SERVER statement. Including a

LIBNAME statement removes the requirement for a physical name in each user's LIBNAME statement that accesses any of those libraries. This can make it easier to maintain your application.

For more information about server libraries, see "Managing a SAS/SHARE Server (Server Administrators)" on page 29. For details about the LIBNAME statement, see Chapter 10, "Remote Library Services," on page 123.

### *Can a SAS/SHARE server access a SAS library across a network?*

Yes, but you usually do not want to organize it that way.

Even though a SAS/SHARE server is not exactly like other file servers that you might be familiar with, it is still a process that generates a lot of disk I/O because a server generates I/O to files on behalf of a large number of users. Therefore, you want the path length between the server SAS session and the physical disk to be as short as possible. Try to store data on the same computer as the server that is used to access that data, whenever possible.

### *I've used servers before. A SAS/SHARE server is similar to the file servers we have on our network, isn't it?*

Not really. Usually, file servers are not aware of the content of the files they manage, but a SAS/SHARE server allows several users to update the same copy of a SAS file at the same time.

SAS/SHARE is tuned to manage locking conflicts within SAS files, such as two users attempting to update the same observation of a SAS data file or two users attempting to modify the same entry in a SAS catalog. SAS/SHARE is not optimized to provide the bulk data transfer services at which many file servers excel.

### *Can a server use more than one communications access method?*

Yes. A server administrator uses SAS options to enable this.

If your application requires the use of more than one communications access method, ask your server administrator to set up the server for your application with the access methods that you need. For more information about access methods, see "Specifying a Communications Access Method" on page 30.

### *Do I have to use a different SAS/SHARE server for each file that is updated by the users of my application?*

No. A server can share many files in the same SAS library and in many different SAS libraries at the same time.

### *Is there a limit on how many users or libraries a SAS/SHARE server can support?*

No. There are no limits coded into the software, and you do not need to use SAS options to specify how many users or files a server will support at one time.

However, a server is the same as any other process on a computer; as it is asked to handle greater workloads it takes longer to do the work. It is possible to put so much traffic through a server that users complain about response time. If any of your servers become that busy, you should consider creating one or more additional servers and dividing the files among the servers.

See your server administrator about creating additional servers.

### Do I need to ask my server administrator to start and stop my application's server each day?

Probably not. As with other processes on a computer, SAS/SHARE servers can usually run for long periods of time without intervention. Sometimes periodic maintenance or backup activity requires processes to be stopped for a period of time and then restarted. Servers are not immune to such interruptions.

## FAQs by Server Administrators

### How do I get started with SAS/SHARE software?

Read "Getting Started with SAS/SHARE" on page 3, giving special attention to when PROC SERVER is started and stopped. Occasionally, you might need to use PROC OPERATE, so you should read those tasks in this exercise.

Usually, a SAS/SHARE server is started when initialization of the operating environment is completed, and it continues to run until the computer is shut down or the server is terminated. You should be familiar with creating and managing those types of processes. Of course, a server only generates I/O or uses the processor while users are accessing data through it; a server doesn't process a residual amount of work when it is not processing work on behalf of other users.

Because a server executes within a SAS session, you need to know how to invoke SAS on each computer on which a server will run.

### I've used servers before. A SAS/SHARE server is similar to the file servers we have on our network, isn't it?

Not really. Ordinarily, file servers are not aware of the content of the files they manage, but a SAS/SHARE server allows several users to update a single copy of a SAS file at the same time. Also, SAS/SHARE servers automatically translate transmitted data when the client operating environment represents data differently from the server operating environment.

### Is being a SAS/SHARE server administrator a full-time job?

No! SAS/SHARE is designed to require no regular maintenance or other administrative activity.

### Can a server administrator control access to a server?

Yes. By default, SAS/SHARE does not restrict who can connect to a server or which files they can access, but you can restrict access to a server with the OAPW= and UAPW= options in the PROC SERVER statement. The OAPW= option specifies a password, which the server administrator must supply (in the OPERATE procedure), to connect to the server. The UAPW= option specifies a password that the user must supply in the LIBNAME statement to connect to the server. Of course, your file system restricts a server's access to files based on the access permission set for the files and the server's process. You can also set up a secured server. For more information, see "Server Security" on page 36.

### Can a server administrator control which libraries users can access through a server?

Yes. For each server, you can prevent users from defining libraries to the server and restrict them to using only those libraries that you define. To do this, use the NOALLOC option in the PROC SERVER statement. See "Limiting the Libraries a Server Can Access" on page 37.

Remember that passwords can be used to restrict access to individual SAS files. See "PW= Data Set Option" in *SAS Data Set Options: Reference* in *SAS Data Set Options: Reference* for more information about data set passwords.

### *How can I terminate a user's connection to a SAS/SHARE server?*

First, decide whether you want the access terminated immediately or as soon as it is convenient for the user.

The QUIESCE USER command disconnects a user from a server when the user ends the SAS program step currently being executed or closes the window currently being used. The STOP USER command immediately terminates a user's connection to a server and might cause loss of updates that have not been communicated to the server.

In either instance, the user cannot reconnect to the server until a START USER command is executed, which lets the user reconnect, or until the server is restarted. Servers do not retain a list of stopped users when they are terminated and restarted.

See "Quiescing User Access to a Server " on page 149 and "Terminating User Connections to a Server " on page 150.

### *How can I stop a SAS/SHARE server?*

The QUIESCE SERVER command causes a server to stop when all users have disconnected from the server. The STOP SERVER command immediately stops the server and might cause loss of updates that have not been communicated to the server.

See "Quiescing a Server " on page 144 and "Stopping a Server " on page 147.

### *Can a server use more than one communications access method?*

Yes, if your operating environment supports more than one communications access method. See "Specifying a Communications Access Method" on page 30 for information about the communications access methods available on your operating environment.

### *How can I determine when I need to create a second SAS/SHARE server?*

You need to create a second server when the traffic on a server becomes so heavy that an application's performance is less efficient.

Just as you periodically check the resource consumption of the other service processes on a computer, you should, from time-to-time, consider the amount of CPU, I/O, and virtual storage the servers are using. Using operating environment management tools, you might notice that a server is executing a very large number of disk I/O operations or needs a very high percentage of the processor. When you observe those conditions, consider moving some of the work from that server to another, possibly new, server.

Distributing the workload among servers must be a cooperative effort between server administrators and applications developers. SAS provides a set of autocall macros that assign resources to servers symbolically. These macros can make moving resources from one server to another much easier. See "SAS/SHARE Macros for Server Access" on page 75.

## *Accessibility Features in SAS Products*

For information about accessibility for any of the products mentioned in this book, see the documentation for that product. If you have questions or concerns about the accessibility of SAS products, send e-mail to **accessibility@sas.com**.

*Chapter 2*

# Using SAS/SHARE Software



## SAS/SHARE Is a Multi-User Data Server

### *SAS/SHARE Enables Concurrent Update Access*

SAS/SHARE is a multi-user data server that provides several advantages for local and remote SAS clients and external clients (that is, not SAS applications). The multi-user SAS/SHARE server enables two or more clients to write to the same SAS file at the same time. This is called concurrent update access.

Often, SAS/SHARE is run in an environment in which multiple client sessions want to share (read from and write to) records in the same SAS data set.

The following list gives a sample of operations that multiple clients can perform concurrently:

- While one user is creating a member in a data library, other users can create, read, delete, and update members in the same library.
- While one user is using the SAS Explorer window on a data library, other users can open the same window to browse, delete, edit, or re-name members in the same library. You can also copy a member from the SAS Explorer window.

- While one user opens the CATALOG window on a catalog, other users can open the CATALOG window to browse, copy, delete, or rename entries in the same catalog.
- While one or more users are using the FSEDIT procedure, the FSVIEW procedure in edit mode, the UPDATE statement in the SQL procedure, or an SCL program to update a SAS data set, other users can perform the following activities:
  - update the SAS data set by using the MODIFY, REMOVE, or REPLACE statement in a DATA step
  - read the SAS data set as input data by using the SET statement in a DATA step
  - add observations to the SAS data set by using the APPEND or SQL procedure or remove observations by using the SQL procedure
  - copy the SAS data set that is being updated (and copy other members of the library) into another library by using the COPY or the CATALOG procedure

### *SAS/SHARE Provides a Path to Remote Data*

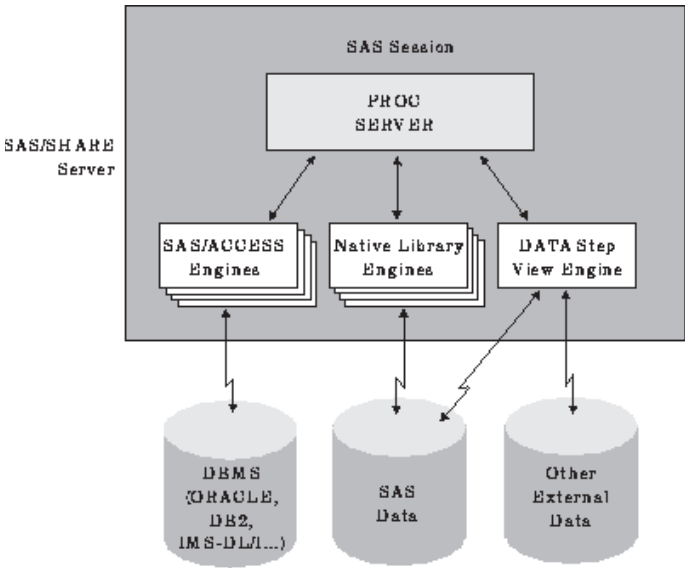
The multi-user SAS/SHARE server also provides remote clients a path to shared data, even if they want only to read that data, without the overhead of a SAS/CONNECT sign-on.

In this scenario, you might have a network of client machines that need read access to a data set that resides on a central server system machine. You have two choices. You can use SAS/CONNECT and have each client create a SAS/CONNECT server session on the central server machine. However, if you need to read only a small-to-moderate amount of data, the overhead for each client that is signing on to the central server and starting a SAS/CONNECT server session might be significant. Also, the additional load on the central machine that comes with each of these server sessions might have a negative impact. Alternatively, you can have those client sessions access data through a SAS/SHARE server that is running on the central machine and avoid the overhead and additional load. Because the server is already running and it serves multiple users, connecting to the server and accessing the data takes very little time.

### *SAS/SHARE Is the Hub between Data and Clients*

Think of a SAS/SHARE server as a hub that serves clients with data from many different sources. For example, a server must use a SAS/ACCESS engine to Oracle in order to access data that is stored in an Oracle DBMS. Or, a server can access SAS data through a Native Library engine. See your SAS/ACCESS documentation for details about using an engine to access specific data. The following figure shows a sample of the data sources that a SAS/SHARE server can provide to its clients.

***Figure 2.1*** *Data Sources for a SAS/SHARE Server*




With an identified DBMS, a SAS/SHARE server provides data to the requesting client for its data processing needs. Beginning with SAS 8, this support extends to clients other than the classic SAS client.

Licensing SAS/SHARE*NET software enables you to send requests to a SAS/SHARE server from a client that is not a SAS application. A SAS/SHARE*NET server is a SAS/SHARE server that includes the Data Services component of SAS/IntrNet software.

Here are some examples of clients that are not SAS applications:

htmSQL
: runs a Web server and provides a gateway to your SAS data from a Web browser. It enables you to incorporate data into a Web page by using SQL queries.

Java applets or applications
: use SAS/SHARE*NET Driver for JDBC, which enables you to write Java applets or applications that can view and update data through a direct connection to a SAS/SHARE*NET server.

C programs
: use the SAS SQL Library for C, which is an API that enables you to create applications that use SQL queries and statements to access data in SAS data sets and in other database management systems.

Applications that use the ODBC driver, such as Microsoft Excel
: use the ODBC driver, which provides ODBC-compliant Windows applications with read-and-write access to local and remote SAS data sets.

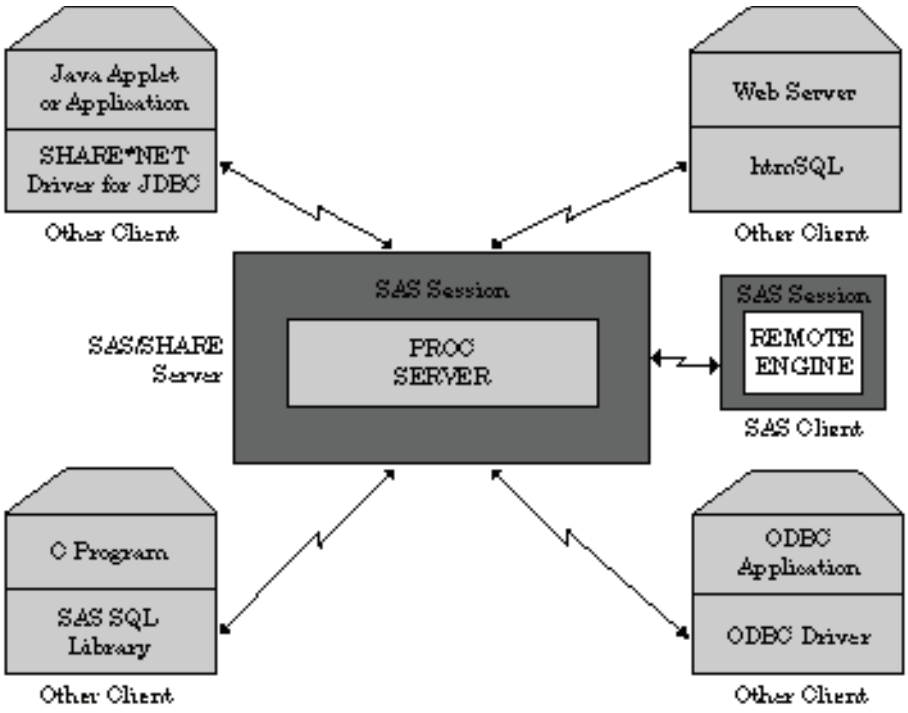
OLE DB consumer or ADO applications
: (beginning in SAS 8) use ShareProvider to view and update data through a direct connection to a SAS/SHARE*NET server. ShareProvider implements the Microsoft OLE DB specification and can be used by OLE DB-compliant or ADO-enabled applications.

Each of the preceding client interfaces or applications has its own documentation.

The following figure shows a sample of the types of clients that a SAS 8 (and later) server supports and a SAS/SHARE server that is running in a SAS session in a supported operating environment. A server administrator starts the SAS/SHARE server session.

SAS/SHARE clients can connect to the server from any machine on your network. SAS clients use the REMOTE engine to access data through a SAS/SHARE server.

***Figure 2.2*** *SAS/SHARE Server Clients*

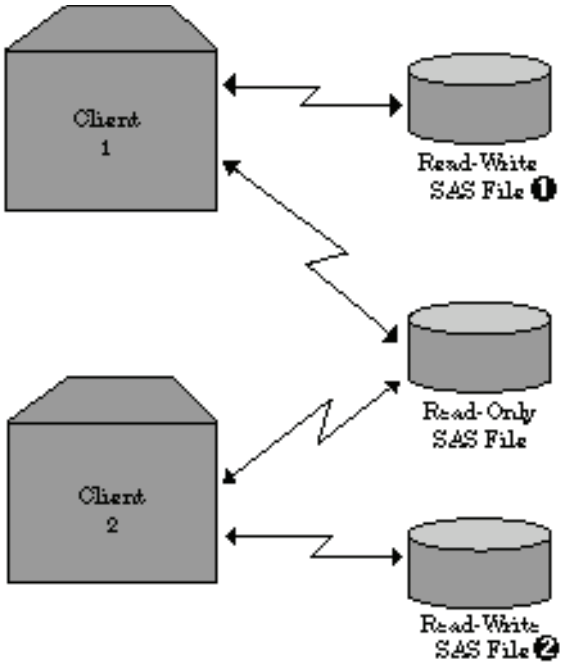



"Other client" refers to a client that is not a SAS application. For each of these clients, the appropriate client-side drivers and libraries must be invoked.

## Accessing SAS Files through an Operating Environment

The following figure illustrates two client sessions that are accessing SAS files without using a SAS/SHARE server.

***Figure 2.3*** *SAS Library Access through an Operating Environment*




In this instance, your operating environment provides two types of access: read-only and read-write. With read-only access, multiple users can simultaneously read the same member or different members in the same library.

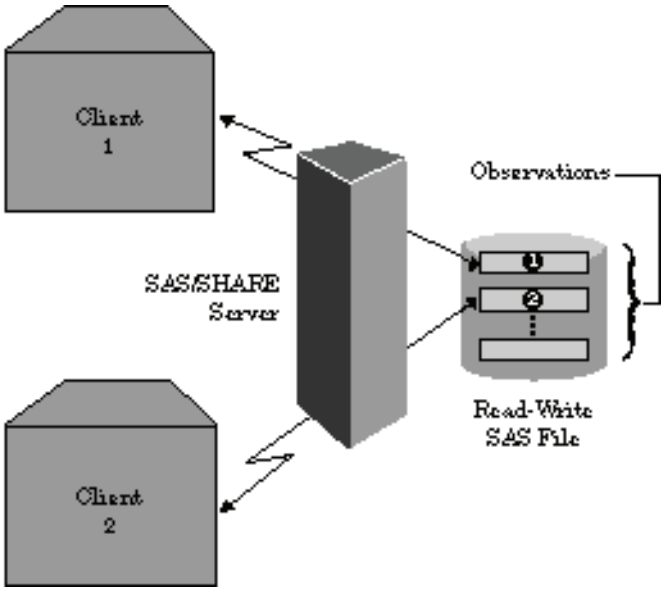
SAS files that have read-write access are usually associated with only one user. Some operating environments support only read access to libraries that are accessed directly by more than one user. Other operating environments permit two users to write to different files in the same SAS library but do not permit them to write to the same file simultaneously. Still other operating environments permit multiple users to write to the same file at the same time, even though doing so is not safe and often results in lost changes made by one or more users.

The only safe way for more than one user to share a SAS file (and on some operating environments an entire library) is to use a common third-party process to sequentially access the low-level parts of the file and coordinate updates to the data. SAS/SHARE is a product that regulates access to SAS files. For more information about the SAS library model, see *SAS Language Reference: Concepts*.

## Accessing SAS Files through a SAS/SHARE Server

The following figure illustrates two client sessions that are accessing SAS files by using a SAS/SHARE server.

***Figure 2.4*** *SAS Library Access through a SAS/SHARE Server*




The SAS/SHARE server enables multiple clients to effectively share the same SAS file at the same time. In this context, "share" means to allow access by multiple clients to a different unit (for example, an observation) in the same SAS file. In the preceding figure, Client 1 can read from and write to observation 1, and Client 2 can read from and write to observation 2 in the same SAS file. Both clients can also read the same observation at the same time. However, only one client at a time can write to an observation.

The server enables client access to the lowest unit of the SAS library hierarchy through its powerful lock manager facility. Locks are applied in either of two ways:

- A client might specify an explicit lock with a LOCK command or a LOCK statement.
- A client operation might automatically submit a request for an implicit lock.

The SAS/SHARE server evaluates each incoming client request to access a specific SAS library unit (for example, a data library, a data set, or an observation) against a complex set of locking rules whose application seems transparent to clients. The server balances the client requests for access to data while ensuring the integrity of that data. The server grants permission to a qualifying client and denies a conflicting request. The denied client receives an informational message. For more information about server locking rules, see "Locking SAS Data Objects" on page 59. For information about the use of the LOCK command and the LOCK statement, see "Locking Objects Explicitly (LOCK Statement)" on page 63, and "Locking Explicitly in a SAS Window (LOCK Command)" on page 68.

***CAUTION:***

**Although a SAS/SHARE server can effectively control multi-client access to a SAS library, independent direct access to the data from other SAS sessions or operating environments can interfere with the SAS/SHARE server.** Simultaneous and unregulated access to a common SAS library causes unpredictable results and possibly corrupt data. Your site policies should prohibit unregulated data access. Make sure that your data access policies are clearly communicated.

# Remote Library Services Provides Remote File Access

SAS/SHARE provides remote file access through its Remote Library Services (RLS). RLS provides transparent access to remote data libraries for moving data through the network as the local SAS session requests it.

This access to remote data is provided through the REMOTE engine. Therefore, SAS products can gain single-user or multi-user access to remote SAS data or third-party DBMS data, as applicable, by invoking a SAS/SHARE server and assigning a library to the server through the REMOTE engine.

The LIBNAME statement associates a SAS library reference (libref) with a permanent SAS library, which can be specified by using an operating environment-specific full physical name. Usually, the SAS/SHARE server uses the BASE engine to access data; however, alternative engines can be assigned by using the RENGINE= option. Attributes for the BASE or alternative engine can be supplied by using the ROPTIONS= option.

For information about the LIBNAME statement in SAS/SHARE, see Chapter 10, "Remote Library Services," on page 123. For the LIBNAME statement in SAS/CONNECT, see "LIBNAME Statement" in *SAS/CONNECT User's Guide*. For more information about the LIBNAME statement in SAS/ACCESS, see *SAS/ACCESS for Relational Databases: Reference*.

# SAS/SHARE and the SAS Intelligence Platform

## *About the SAS Metadata Repository and the SAS Intelligence Platform*

The SAS Metadata Repository is a collection of files that store metadata that is used by SAS client applications that are deployed in a SAS Intelligence Platform environment. Objects that define SAS/SHARE servers and their access to specified SAS libraries through various engines are examples of metadata that is available to client applications.

The SAS/SHARE objects that are configured as metadata using the user interface SAS Management Console are similar to the arguments that can be specified in PROC SERVER and LIBNAME statements in a SAS/SHARE application. SAS/SHARE applications that execute in the traditional SAS interactive and batch execution modes do not typically access and use SAS/SHARE metadata that has been configured in a SAS Metadata Repository. However, a LIBNAME statement for the METADATA engine can be used to programmatically access and use SAS/SHARE metadata that is stored in a SAS Metadata Repository. For details, see SAS Language Interfaces to Metadata. Also, the AUTHDOMAIN= option in the LIBNAME statement can be used obtain a user ID and password that is stored in the SAS Metadata Repository. For details, see the AUTHDOMAIN= option on page 126.

For information about SAS/SHARE in a SAS Intelligence Platform environment, see the *SAS Intelligence Platform: Overview*. For details about using SAS/SHARE in the traditional interactive or batch modes, use the information in this document.

### Configuring a SAS/SHARE Server and Server Libraries in a SAS Intelligence Platform Environment

A SAS/SHARE server can be included during the installation and configuration of a SAS Intelligence Platform environment. For details, see the *SAS Intelligence Platform: System Administration Guide*. To add an instance of another SAS/SHARE server after the SAS Intelligence Platform is operational, you can use the SAS Management Console and invoke the New Server wizard, which prompts the user for connection information about the SAS/SHARE server, such as the communications protocol, the address of the computer that the server will run on, and the server ID. For more information, see the Help for SAS Management Console.

After a SAS/SHARE server is configured, you can associate data sources with the SAS/SHARE server that will be available for shared access. You can use the Data Library Manager in SAS Management Console to define server libraries that are accessible via the SAS/SHARE server. For more information, see the *SAS Intelligence Platform: Data Administration Guide*.

### Managing a SAS/SHARE Server

There are two basic meanings of the term SAS/SHARE management. You can manage the metadata about client access to SAS/SHARE servers and libraries in a SAS Intelligence Platform environment, and you can manage an instance of an active SAS/SHARE server in the SAS Intelligence Platform environment or in a traditional SAS interactive or batch execution environment:

- In a SAS Intelligence Platform environment, you can manage SAS/SHARE metadata using the user interface SAS Management Console to add and modify a SAS/SHARE server and access to libraries. These objects are stored and managed as metadata in the SAS Metadata Repository.
- You can manage an instance of an active SAS/SHARE server.
  - In a SAS Intelligence Platform environment, scripts are generated during a phase of the installation that can be invoked to start, stop, restart, and pause a SAS/SHARE server. For details, see the *SAS Intelligence Platform: System Administration Guide*.
  - In the traditional SAS interactive or batch modes of program execution, you can use the OPERATE procedure in SAS/SHARE in order to monitor and control the SAS/SHARE server, server libraries, and server users. For example, you can use PROC OPERATE to define a SAS library to a server after a server session has started, to display information about assigned libraries, to terminate access to a library, and to display user IDs of users who are connected to the current server session. For details, see Chapter 11, "The OPERATE Procedure," on page 135.

    *Note:* You cannot monitor and control an active instance of a SAS/SHARE server using SAS Management Console.

## SAS/SHARE Software Components

SAS/SHARE software consists of the following procedures and engine:

SERVER procedure
: manages and performs input and output requests to SAS files on behalf of SAS clients and clients that are not SAS applications.

OPERATE procedure
: manages server, library, and client resources. For example, you can allocate a library to a server, free a library, stop a server, restart a server, and display information about clients.

REMOTE engine
: enables a client SAS session to access SAS data by means of a SAS/SHARE server. (The REMOTE engine is also licensed and distributed as part of SAS/CONNECT software.)

  The REMOTE engine does not operate directly on files in a SAS library; it communicates with a SAS/SHARE server. Each client runs a private copy of the REMOTE engine to communicate with a server.

  A server uses one or more library engines or view engines to operate directly on files in SAS libraries that clients access through the server. The server routes requests from the REMOTE engine to the appropriate engine for the SAS library or the file that the client accesses. A server's default library engine is specified in the ENGINE= system option. You might override the default and specify another engine when you start a server. Other engines are documented in the SAS documentation for your operating environment.

## SAS/SHARE Users

SAS/SHARE users are divided into three groups:

End users
: update or use concurrently accessed data in other ways. End users might not even be aware of SAS/SHARE. They just need some basic information about how to update data sets and search for observations that they want to modify.

Programmers or developers
: write applications that use a server to access shared data. They must know how to define a libref for a SAS library that is accessed through a SAS/SHARE server and have knowledge of how the server handles locking.

Server administrators
: start a server, manage its session, and evaluate its performance. They give a server access to SAS libraries (taking into account library security) and supply clients with information to access these libraries. Server administrators also make a server accessible to clients who have permission to use it. At some sites, a server administrator might delegate specific functions to other personnel and work with systems personnel to automate administrative functions.

## Migration and Cross-Version Compatibility (SAS 6 through SAS 9.3)

Accessing your data is a primary concern when migrating to a new version of SAS. If the server (and clients) have been upgraded to SAS 9.3, and you want the SAS data and

SAS applications to run at the same level as the server and clients, you can migrate the data to the version that the server runs. For complete details about data migration, see **http://support.sas.com/rnd/migration**.

If you do not migrate your SAS data and applications to the new version of SAS, you will be accessing SAS files and using SAS applications in a cross-version environment. Therefore, it is important to be aware of any restrictions when operating in a cross-version environment.

*Note:* SAS 9.3 has many of the same features as Version 8, but a SAS 9.3 client cannot access a SAS 6 server, and a SAS 6 client cannot access a SAS 9.3 server. Because SAS 8 can communicate with both SAS 6 and SAS 9.3, SAS 8 serves as the intermediate release.

Access to data depends on the following:

- the SAS data object being accessed (library, file, view, or catalog)
- the version and release of SAS being used (SAS 6 through SAS 9.3) to create the data that is being accessed or to develop the client application
- the version and release of SAS being used by the client session or by the server session

For more information about cross-version issues, see "Limitations of Cross-Version Client/Server Access" on page 245.

*Chapter 3*

# Managing a SAS/SHARE Server (Server Administrators)



## Starting a Server: A Fast-Track Approach

You might need to start a new server quickly if in the following situations:

- You are installing SAS/SHARE for the first time.
- A new application presents a sudden demand.
- System resources become overloaded and require that the workload be shifted to a new server.

If you are a new user of SAS/SHARE, there are various issues to consider as you fine-tune your server environment, such as system options, predefined libraries, and automated start-up. However, you might want to defer some of these issues and start a server right away so that your applications developers can begin to create new applications or migrate old ones to the multi-user environment.

To get a server started and running with minimal effort, perform the following tasks in a SAS session on the server machine:

1. After the necessary configuration is completed for the communications access method, specify the access method to be used between a SAS/SHARE server and its clients by using the COMAMID= option in an OPTIONS statement at SAS invocation or in a SAS configuration file. For example:

   ```
   options comamid=tcp;
   ```

2. Start the server and assign it the name that you just configured. For example, start a server named SHARE1 as follows:

   ```
   proc server authenticate=optional id=share1;
   run;
   ```

   The following message appears in the server SAS log with a default time stamp:

```
30Apr2003:09:47:30.000 SAS server SHARE1 started.
```

The server SHARE1 can now be used by SAS clients and other clients that are not SAS applications.

***CAUTION:***
**Here are two limitations to the fast-track approach to starting a server.**

- Server security is not set. It might be possible for a client that accesses the server to have the same permissions as the server to access data. Also, any SAS client that can access the server can manage the server by using PROC OPERATE statements. This means that a client can stop the server or stop access to any library through the server. For details about setting server and library security, see “Server Security” on page 36.
- If you run a server SAS session interactively, the SAS session assumes that, by using a dialog box, you can resolve any problems that it encounters. While the SAS session waits for a response to its query, the server might not be able to continue to service client requests until the query is answered. However, you might not be aware that a response is required if the window in which the server is running is not visible or is not being monitored. Therefore, it is recommended that you specify the SAS system option NOTERMINAL so that SAS does not display dialog boxes and performs whatever is required without prompting.

# Specifying a Communications Access Method

A communications access method is the interface between SAS and the network protocol that you use to connect two operating environments. The access method that you use is determined by the network protocols that you have available at your site and the operating environments that you are connecting. Supported access methods for SAS 9.3 are TCP/IP (Transmission Control Protocol/Internet Protocol) and XMS (Cross Memory Services).

For a complete list of valid connections among operating environments and access methods for SAS/SHARE, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Operating environments and access methods usually require the configuration of access method resources on the operating environments on which the server will run. Consult your network administrator about this configuration. For details about configuring access method resources, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Server administrators should also be aware of special considerations and limitations when the server is accessed by clients on an operating environment that is different from that of the server. For information about cross-architecture limitations and considerations, see "Cross-Architecture Access" on page 186.

After you verify that access method resources have been configured but before you start a server, you must specify the access method by using the following syntax:

**OPTIONS COMAMID=***access-method*;

*access-method* is the abbreviation for the method that is used by the server to communicate with a client. In SAS 9.3, the supported access-method identifiers are TCP/IP and XMS.

For a server that runs under an operating environment on which only one access method is available, use only the COMAMID option. For example:

```
options comamid=tcp;
```

You might specify the COMAMID= option in an OPTIONS statement, at SAS invocation, or in a SAS configuration file.

If the operating environment on which the server runs supports multiple access methods, you can use the COMAUX1 option to specify alternate auxiliary access methods by which clients can access the server.

All of the access methods initialize when the server initializes. The activation of multiple access methods makes a server available to several groups of clients, each group using a different access method simultaneously.

The COMAUX option can be specified only at a SAS invocation or in a SAS configuration file. Here is the syntax for the COMAUX option:

**COMAUX1=***alternate-method*

*z/OS Specifics*

Here is an example of configuration file entries for a server that is running under z/OS:

```
-comamid tcp
-comaux1 xms
```

When the server starts, all of the communications access methods are initialized. The server is simultaneously available to client sessions that use the TCP/IP access method and to clients that use the XMS access method.

For more information about access methods, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

# Predefining SAS Libraries to the Server

## Advantages of Predefining Libraries

Although it is not required, predefining one or more selected SAS libraries to the server for use by client applications provides the following advantages:

- Applications can use the libref that you define to identify the library. This protects the applications against changes to the library's physical pathname.
- The library is available to all clients that use the server's SQL services, including SAS clients that use the Remote SQL Pass-Through (RSPT) facility, and other clients that use the SAS ODBC driver, the JDBC driver, or the SQL library for C.
- The server administrator can refer to the library with a libref in PROC OPERATE commands.

Be aware that the user ID from which the server runs must have the appropriate permissions to access the SAS library that you want to predefine to a server. You should be familiar with the permissions requirements of the operating environment, the network, and the security software before you predefine a library. For information about server security, see "Server Security" on page 36.

## Methods for Predefining a Server Library

### Predefining a Server Library by Using the LIBNAME Statement

In your server program, prior to specifying the PROC SERVER step, you can predefine one or more SAS libraries to the server by using the following syntax:

**LIBNAME** *libref* '*SAS-data-library*' <*options*>;

How you specify the physical pathname of the SAS library and options depends on your operating environment. For details about the LIBNAME statement and options specific to your operating environment, see *SAS Language Reference: Concepts* and the SAS documentation for your operating environment.

Here is an example of how to predefine a library to the server for a UNIX operating environment by using the LIBNAME statement:

```
libname mylib '/payroll/div2/emp';
```

After you define a library to the server, a message is displayed in the SAS log. For example, here is a message for a server on a UNIX operating environment:

```
1     libname mylib '.';
NOTE: Libref MYLIB was successfully assigned as follows:
Engine: V9
Physical Name: /payroll/div2/emp
```

***CAUTION:***

**You are discouraged from using the FILELOCKWAIT= option in the LIBNAME statement,** which specifies the maximum length of time that a SAS/SHARE client or server will wait for a SAS file that has been locked by another process to become available for use. If the wait time expires before the file becomes available, the

SAS/SHARE request to open the file will fail and a message is written to the log. When using SAS/SHARE to access SAS files, do not perform additional jobs that might contend for access to the same files. For details about the FILELOCKWAIT= option in the LIBNAME statement, see the *SAS Companion for UNIX Environments* or the *SAS Companion for Windows*, as appropriate.

### Predefining a Server Library by Using the ALLOCATE LIBRARY Command in PROC OPERATE

In a server administrator session, while the server is running, you can predefine one or more SAS libraries to the server by using the following syntax:

**PROC OPERATE SERVER=***server-ID*;

**ALLOCATE LIBRARY** *libref* '*SAS-data-library*' <*options*>;

How you specify the physical pathname of the SAS library and options depends on your operating environment. For details about the ALLOCATE LIBRARY command, see "Defining a Library to a Server That Is Running" on page 139. For details about options specific to an operating environment, see *SAS System Options: Reference* and the SAS documentation for your operating environment.

Here is an example of using the ALLOCATE LIBRARY command in a PROC OPERATE step under Windows:

```
proc operate server=share1;
  allocate library mylib '\payroll\dev2\emp';
```

*Note:* In addition to these recommended methods, you can also define a libref externally to SAS. For this information, see the SAS documentation for your operating environment.

## Starting a Server

### PROC SERVER Statement

To start a server, execute the PROC SERVER statement, which enables multiple clients to simultaneously access and use SAS libraries and members in those libraries. As part of server start-up, you must assign a server name, but specifying operational attributes is optional.

To start a server with selected options, use the following syntax:

**PROC SERVER** <ID=*server-ID*> <ALLOC|NOALLOC> <AUTHENTICATE=OPT|REQ> <CLIENTID

The following sections explain the preceding options. For complete information about server options, see Chapter 9, "The SERVER Procedure," on page 105 . For an example of a typical log, see "Sample Log for SAS/SHARE Server SHARE2" on page 88.

### Identifying the Server

ID=*server-ID* assigns a valid SAS name to the server. In SAS, valid names can contain a maximum of eight alphanumeric characters and can include the following special characters: dollar sign ($), at sign (@), and pound sign (#). For more information about the rules for naming SAS variables, see *SAS Language Reference: Concepts*.

Server names are also constrained by the operating environment under which that server runs and the access method that is used. For complete information about server IDs by operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Here is an example of specifying a server ID:

```
proc server id=share1;
```

### Limiting Users to Predefined Libraries

Use the NOALLOC option to limit users to accessing only libraries that you predefine to the server and to control which data users can access through the server.

### Validating Server Users

Use the AUTHENTICATE= option to control whether the server will require clients to provide valid user IDs and passwords before they are connected to the server. See "Ensuring That User IDs Are Valid" on page 38.

### Selecting How Clients Are Identified in the Log

Use the CLIENTID= option to specify whether clients are identified in the server log by their session names or their secured names. For more information, see PROC SERVER Statement on page 106.

### Logging Server Usage Statistics

Usage statistics can be requested for each client that accesses a server. These statistics are useful for debugging and tuning server applications. Usage statistics also enable you to charge users for the amount of server resources that they consume. By default, the server writes a client's usage statistics to the server log when the client disconnects from the server. Here are some of the statistics that you can request:

Number of messages processed
: shows the number of messages (requests and replies) that a client exchanges with a server in a single session.

Bytes transferred
: shows the cumulative number of bytes that are received from a client and that are sent to a client in a single session.

Active and elapsed time
: shows the cumulative elapsed time during which the server processed requests on behalf of a client in a single session. Although this figure is not CPU time, it is related to CPU time. Whereas CPU time for a specific operation usually is relatively independent of other server usage, this figure increases with an increased level of server activity. However, active time should give a good indication of the CPU usage by the client compared with other clients' values that are tracked during similar levels of server activity. The active time value can exceed the elapsed time value, especially in the server totals, because many server requests can be active (therefore, they are being timed) concurrently.

Here are examples of setting log usage:

```
proc server id=share1 log=message;
proc server id=share1 log=bytecount;
```

```
proc server id=share1 log=(message bytecount activetime elapsedtime);
proc server id=share1 log=all;
```

Here is an example of a client log for all statistics:

```
Usage statistics for user mike(1):
   Messages processed:       5,143
   Bytes transferred:       10,578K
   Active time:       1:47:23.6148
   Elapsed time:      3:28:64.7386
```

For complete information about the LOG option, see Chapter 9, "The SERVER Procedure," on page 105. For a more complete example of a SAS log, see "Usage Statistics in the Server Log" on page 88.

To charge users for the amount of server resources that they consume, allocate consumption proportionately according to the usage statistics. You can allocate consumption based on a single statistic or on a combination of statistics. The most useful statistics for this purpose are the number of messages that are processed, the number of bytes that are transferred, and the amount of active time used.

You might need to experiment with the relative weights of these statistics in your charge-back formula. These statistics are sensitive to the specific operating environment, access method, level of server activity, and types of applications.

The number of messages that are processed represents actual, billable work by the server. When used together, the MESSAGE, BYTECOUNT, and ACTIVETIME values in the LOG= option report data that characterizes the work that a user asked the server to perform. Here are some examples:

- Small BYTECOUNT and MESSAGE values and a large ACTIVETIME value might indicate that a small amount of data was selected from a large file by sequentially searching the file or by interpreting a complex view.
- Moderate or large BYTECOUNT values and a large MESSAGE value might indicate that a small amount of data is being read by the user on each message that is exchanged with the server. This might be caused by random access or exceptionally long observations and might suggest taking a snapshot of the data for the user's analysis.
- Small ACTIVETIME values and a large BYTECOUNT value suggest that the user is exchanging data "in bulk" with the server. Usually, that does not indicate a problem, but if the server is overloaded, you might want to suggest that the user try another bulk-data transfer technique.

### *Specifying the Format for the Server Log Datetime Stamp*

You can prepend a datetime stamp of a specific format to each message that is written to the server log, or you can suppress the datetime stamp. The default format DATETIME22.3 presents the date and time in the form *DDMMMYYYY:hh:mm:ss.ddd*.

Here are examples of how to set the datetime stamp:

```
proc server id=share1 alloc log=cpu dtformat=time11.2;
proc server id=share1 noalloc log=io dtformat=_NODTS_;
```

Here is an example of a datetime format:

```
30Apr2003:14:02.39.186
```

### Using the SAS Console Log To Analyze Server Errors

If the SAS/SHARE server encounters problems at SAS initialization or at SAS termination, the server log might not be available to receive error messages. If the server log is not available, error messages are written to the SAS console log. For details about the SAS console log, see the companion documentation that is appropriate for the operating environment that you are using.

### Specifying a Time Limit on SAS File Availability for Client Access

The client can specify a maximum time to wait for a SAS file to be unlocked and available for access. In this example, the client specifies a limit of 15 seconds.

```
libname sales '.' server=shr1 rengine=base
roptions="filelockwait=15";
```

When you invoke SAS on the computer on which the SAS/SHARE server will run, you can negate the possibility of a stalled server by explicitly setting the maximum wait time to zero. This value will override the value that the client specifies for the FILELOCKWAIT= option in the LIBNAME statement.

```
c:\Program Files\SAS2\SASFoundation\9.2\sas.exe -dmr -noterminal
-nosyntaxcheck -sasuser work -icon -nosplash -filelockwaitmax=0
```

## Server Security

### Controlling Administrator Access to a Server

By default, any user can send administrator requests to a SAS/SHARE server to stop the server or to display and modify its characteristics, such as who can access the server and what SAS libraries it can serve. To prevent unauthorized users from sending these types of requests, you can assign an administrator password to a server when you start it by using the OAPW= option in the PROC SERVER statement. Here is an example:

```
proc server id=share1 oapw=blue31;
```

To specify an administrator password when you use PROC OPERATE to send administrator commands to the server, you must use the SAPW= option. For example:

```
proc operate id=share1 sapw=blue31;
```

You can also specify an administrator password in each PROC OPERATE server management command. For details, see “Displaying Information about a Server” on page 142.

### Controlling Access to Data through a Server

#### Setting Server Security

A SAS/SHARE server must have valid access permission to read and write data that is requested by any of the server's clients. Usually, this means that a server is authorized to access more libraries and data sets than any one of its clients. For this reason, you might

want to set server security to prevent unauthorized access to data through the server by using one or more of the following:

- assigning a single password on the server
- limiting which libraries the server can access
- setting passwords for SAS data sets, views, and catalogs
- placing operating environment or file system protections on libraries and data sets
- requiring user ID validation

### Assigning a User Password to a Server

The simplest method that you can use to limit access to data through a server is to assign a user password to the server when you start the server. This password must be supplied whenever a client connects to the server. This method offers the broadest control over who can access data through the server. It applies to all users regardless of the data that they access or the operations that they perform.

To assign a user password to a server, use the UAPW= option in the PROC SERVER statement. Here is an example:

```
proc server id=share1 uapw=hotwings;
run;
```

When a user specifies the server for the first time in a LIBNAME statement or in a PROC SQL CONNECT TO statement, use the SAPW= option to specify the user password. Here is an example:

```
libname invoice server=share1 sapw=hotwings;
```

### Limiting the Libraries a Server Can Access

To prevent unauthorized access to libraries or data sets that should not be accessed through a specific server, use the operating environment or file system security software to deny access by the server to those libraries or files.

You can also use the NOALLOC option in the PROC SERVER statement to limit users to accessing only those libraries that you predefine. For more information, see "Predefining SAS Libraries to the Server" on page 32 and "Limiting Users to Predefined Libraries" on page 34.

### Operating Environment or File System Protections

When a user reads or writes SAS libraries and SAS files, most operating environments and file systems validate the user's authority to read from or write to that data.

Because a server is interposed between a user and the data, checking access permissions, which is usually performed by the operating environment or the file system security software, must be performed in the server's session to protect access to that data through the server. For this reason, a SAS/SHARE server calls the operating environment or the file system to validate a user's authority whenever an attempt is made to read from or write to a library through the server.

Providing a validated user ID and password as arguments in the LIBNAME statement, the ALLOCATE LIBRARY command in the PROC OPERATE statement, or the Remote SQL Pass-Through statement preempts earlier SAS methods of supplying a user ID and password by using a communications access method. Regardless of the method used for collecting a user ID and password, the server uses the validated user ID in making the authority check. To permit access by user IDs that are not validated, you can use the AUTHENTICATE=OPTIONAL option in the PROC SERVER statement. For

more information about setting options in the PROC SERVER statement, see Chapter 9, "The SERVER Procedure," on page 105. For more information about user IDs and passwords, see "LIBNAME Statement" on page 123, Syntax on page 153, or Chapter 11, "The OPERATE Procedure," on page 135.

In order to validate a user ID, most access methods require using an access method-specific mechanism to provide the user ID and corresponding password for the server operating environment. The access method encrypts the user ID and password and transmits them to the server session to be validated. For information about the mechanisms that control whether an access method validates connecting users and the mechanisms by which users can provide their user IDs and passwords, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

In most operating environments, to validate a user ID and to verify access permissions, a SAS/SHARE server calls the operating environment or the file system directly.

*UNIX Specifics*

Under UNIX, the server performs these functions by calling an external program (that is, not a SAS application) that you can modify.

### Ensuring That User IDs Are Valid

Because a SAS/SHARE server calls the operating environment or the file system to verify a user's access permission to data, the accuracy of the user ID that the server presents to the operating environment or to the file system determines whether a user is allowed to access data.

When all communications access methods are configured (see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*), on most operating environments each user ID is validated by the communications access methods when a user connects to the server. That validation is used for the duration of the connection. The server presents that user ID to the operating environment or to the file system whenever the user attempts to access data through the server.

Specifying the option AUTHENTICATE=OPTIONAL in a PROC SERVER statement bypasses the requirement that the access methods validate a connecting user ID. Use the option AUTHENTICATE=OPTIONAL to get a server running quickly with an absolute minimum of work. Also, this use shows the value of configuring production servers to run without the option AUTHENTICATE=OPTIONAL, which means that production servers can run unattended while only allowing users to access data that they have permission to access.

If you are running a SAS/SHARE server in an operating environment that does not have a security facility that controls user access to files, or if you are running a SAS/SHARE server in an operating environment in which every user should have full access to every file that is available to the server, you might want to use the option AUTHENTICATE=OPTIONAL instead of implementing security for each of the access methods that is used by the server. For example, if a SAS/SHARE server is running under UNIX without a security facility installed, it is not possible for communications access methods to validate connecting user IDs, nor is it possible for the server to verify access to files for each individual user ID.

## Using Encryption Services

As an alternate form of security, you can implement encryption services that protect data that is sent between operating environments across a network. For details, see *Encryption in SAS* .

## Writing a SAS Program to Start a Server

When you write a simple SAS program to start a server, you can set SAS options in a configuration file, which is processed during SAS invocation, or you can set the options explicitly in a server session. To enable communication between the server and clients that are running under different operating environments, you must specify a communications access method. Also, you can predefine SAS libraries to the server. Finally, you start the server and specify the options that you want.

The following sample program shows how to start a server on UNIX. (The exact syntax varies by operating environment.)

```
options comamid=tcp;
libname payable '/dept/acct/pay';
proc server authenticate=optional id=share1 msgnumber;
```

## Automating Server Start-Up

An alternative to starting a server in interactive mode is to create a command file that runs during system initialization and automates server start-up. The tasks that you perform vary by operating environment. For more information about how to automate server start-up under specific operating environments, see "Creating the SAS/SHARE Server Environment" on page 201.

## Managing a Server, Its Libraries, and Its Users

### *Server Management: OPERATE Procedure*

Use commands in the OPERATE procedure to manage the following:

- an active server, or a server that is running. You can display information about a server, quiesce a server, reset a server (for subsequent management operations), restart a quiesced server, and stop a server. For information about managing a server, see "Server Management Commands" on page 142.
- server libraries. You can allocate a library to an active server, display information about a library, and free, quiesce, or stop a library. For information about managing a library, see "Library Management Commands" on page 139.
- users. You can display information about users, quiesce a user's access, restart a quiesced or a stopped user, or terminate user connections to a server. For information about managing users, see "User Management Commands" on page 148.

### *Server Log Reporting: OPERATE Procedure*

The following log shows part of the server administrator's log for PROC OPERATE. The log reports client/server transactions for users John and Maria, who are working on

a server named SHARE1. For information about interpreting server logs that are generated by users, see "Interpreting SAS/SHARE Server Log Messages" on page 87.

***Log 3.1*** *Administrator's Log for the OPERATE Procedure*

```
LOG
Command ===>

1    proc operate serverid=share1;
=============================================================================
2    display user _all_;

                                        NUMBER OF
                  USER ID      STATUS   LIBRARIES
                  ---------------------------------
                  john(1)      ACTIVE         1
                  maria(2)     ACTIVE         2
                  myid(3)      ACTIVE         0
=============================================================================
3    stop user maria;
User maria(2) stopped from active state.
User maria is now disallowed from connecting to server SHARE1.
=============================================================================
4    display user maria;

                                        NUMBER OF
                  USER ID      STATUS   LIBRARIES
                  ---------------------------------
                  maria(2)     STOPPED        0
=============================================================================
5    quiesce user 1;
User john(1) quiesced from active state.
=============================================================================
6    display user 1;
                                        NUMBER OF
                  USER ID      STATUS   LIBRARIES
                  ---------------------------------
                  john(1)      QUIESCED       1

User john(1) is accessing these libraries:

        USER LIBREF    SERVER LIBREF    LIBRARY NAME
        ------------------------------------------------------------
        DATALIB        SYSUSE
<SAS-library-name>

User john(1) is accessing these data sets:

USER LIBREF    SERVER LIBREF    MEMBER       TYPE         OPEN MODE
-----------------------------------------------------------------------
DATALIB        SYSUSE           USAGE        CATALOG       UPDATE
=============================================================================
7    start user maria;
User maria started from stopped state and therefore has become unknown
to server SHARE1.
=============================================================================
8    quit;
```

1 PROC OPERATE executes commands 2 through 8 (shown in the preceding output).

2 The DISPLAY USER command requests general information about all users who are currently accessing SHARE1. The users are listed by user ID, current status, and the number of library assignments to the server.

3 The STOP USER command immediately disconnects user MARIA(2) from the server. If Maria is using the FSEDIT procedure to update an observation when the STOP command is issued, she loses the updates on her display, but she does not lose previous updates. The STOP USER command terminates all attachments to that server, and she is prohibited from accessing that server until the administrator issues a START USER command.

4 The DISPLAY USER command shows Maria's current status. No libraries are listed for Maria because the STOP USER command released them.

5 The QUIESCE USER command gradually terminates user JOHN(1), which allows John to finish work in the data sets that he currently has open.

6 The subsequent DISPLAY USER command reports that John is still accessing member USAGE in library DATALIB. When John closes USAGE in that library, the server releases the library and disconnects John. Because John's session was quiesced by its connect number, John can reconnect to the server when he wants to. He receives a new connection number at that time.

7 The START USER command allows Maria to access the server again. Note that the START USER command does not establish a communication path between the server and Maria. It only enables Maria to re-establish a path by using a LIBNAME statement or an SQL CONNECT TO statement. She must explicitly re-access the server.

8 The QUIT statement terminates PROC OPERATE.

### *Freeing a Library that Contains a Locked Data Set*

An administrator might need to free a library from a server so that another process can directly manipulate data in the library. Use these steps to free a library from a server. For details about the commands used in these steps, see Chapter 11, “The OPERATE Procedure,” on page 135.

1. Use the QUIESCE LIBRARY command to change the library's status from active to stopped.

   The QUIESCE command returns immediately, but the quiesce process does not complete until the final user has voluntarily released the library from use.

2. Use the DISPLAY LIBRARY command to find out the active users and their user librefs.

   Contact these users and ask them to release the library. Wait a sufficient period of time for users to finish their work. Display library usage again to verify that all users have released the library from access.

3. Use the STOP USER command to terminate all client sessions that reference SAS views into the specified SAS library.

   These views were identified using the DISPLAY LIBRARY commands. Client sessions that hold open views into the specified library from another library should also be terminated.

4. If the library has not yet stopped, use the STOP LIBRARY command to change the library's status to stopped. This command immediately terminates all remaining user access to the specified library.

5. Use the FREE LIBRARY command to release the specified library from use.

   The library should be freed from use.

*Chapter 4*

# Writing End-User Applications to Access Shared Data



## Accessing Libraries through a Server

### *Introduction*

The information in this section is primarily directed to applications developers. However, server administrators and end users might also find it of interest. For more information about programming techniques and adjusting SAS system option values to

improve the performance of your client/server applications, see "Tuning Tips for Applications That Use SAS/SHARE Software" on page 215.

### Using the LIBNAME Statement

To access a SAS library or an external DBMS through a SAS/SHARE server, you must use the REMOTE engine to define a libref for the library. In a LIBNAME statement, specify the libref, which identifies the library or the DBMS, and the SAS/SHARE server that you'll use to access that library or DBMS.

In the following example, the engine name REMOTE, which is usually specified between the libref and the pathname, is omitted because it is implied by the SERVER= option.

```
libname invoice '/dept/acct/data/invoice' server=share1;
```

If the library is predefined to the server by a server administrator, you can omit the pathname and use only the libref, which is defined for the library in the server SAS session, to identify the library. Omitting the pathname protects your application if the pathname for the library has changed.

In the following example, the REMOTE engine and the server assume that the libref (invdata), which you defined for your SAS session, is the same libref that is defined by the server administrator in the server SAS session.

```
libname invdata server=share1;
```

If the library is predefined to the server but you want to define a libref that is different from the one that is defined in the server SAS session, use the SLIBREF= option in the LIBNAME statement to specify a newly defined server libref, as shown in the following example. However, if a server runs with the option NOALLOC in effect, all libraries that are accessed through that server must be predefined by the server administrator.

```
libname invoice server=share1 slibref=invdata;
```

For details about the LIBNAME statement syntax, see Chapter 10, "Remote Library Services," on page 123.

### Using Macros to Generate a LIBNAME Statement

Hard coding the server name in a LIBNAME statement can be a problem if the server administrator shifts one server's traffic to another server, thereby invalidating the server name.

You can avoid this problem by using a SAS macro to generate the required LIBNAME statement. If you use a macro in your end-user application, you can change the name of the server in one place, even though multiple applications access data through that server. Using macros improves maintenance of your SAS programs.

SAS/SHARE provides macros for generating LIBNAME statements. To associate a server name with a SAS library, register the library in the table that is maintained by the server administrator. Instead of specifying a LIBNAME statement to access the library, use the LIBDEF macro. When you invoke the LIBDEF macro, it searches the table of registered data libraries for the specified library. When the LIBDEF macro finds the library, it uses the associated server name and the specified libref to construct the LIBNAME statement. Before the first invocation of the LIBDEF macro in a SAS execution, you must invoke the SHRMACS macro with the keyword USER.

```
%shrmacs(user);
```

The SHRMACS macro generates and compiles other SAS/SHARE macros and builds the SAS server-alias and library-alias tables in which server libraries are registered.

To invoke the LIBDEF macro, specify the libref and the optional server library. Here is an example:

```
%libdef(datalib<,SAS-data-library>);
```

For more information about using the LIBDEF macro, see "Generating a LIBNAME Statement By Using the LIBDEF Macro" on page 84. For complete details about macro syntax, see Chapter 14, "SAS/SHARE Macros," on page 165. Contact your server administrator for information about how the macros and tables are implemented and how they are used at your site.

# Locking Data Objects in Your Programming Environment

The SAS/SHARE lock manager enables multiple clients to share the same SAS file at the same time. The server's ability to manage multi-client access of selected data objects is contingent on a complex set of locking rules that are affected by many factors. These locking rules must be considered in relation to each programming product (SAS and other products) that you use to write applications that enable accessing data through the SAS/SHARE server. For complete information about locking, see "Locking SAS Data Objects" on page 59.

# SAS Programming Considerations

## *DATA Step Processing*

John and Maria have concurrent access to the SAS data set FUEL in their respective sessions. While Maria is editing the data set DATALIB.FUEL in an FSEDIT window, John can use a SET, a MERGE, or an UPDATE statement in a DATA step to read DATALIB.FUEL. Although John cannot create a new version of DATALIB.FUEL, he can create other data sets or written reports.

The following program shows the effect of implicit locking when two clients access the same SAS data set at the same time:

```
data _null_;
   set datalib.fuel;
   file report ps=24 n=ps;
   ...
run;

data composit;
   merge datalib.fuel fuel96;
run;
```

If John uses a SET statement to read DATALIB.FUEL, he cannot specify the KEY= or POINT= option unless he overrides the member-level control. By default, member-level control is required when either of these options are included in a SET statement. Here's an example.

```
data pressure;
   set fuel (keep=fuel maxpress);
   set datalib.fuel (cntllev=rec) key=fuel;
   ...
run;
```

If John uses an UPDATE statement or a SET or a MERGE statement with a BY statement to read DATALIB.FUEL, he should consider specifying member-level control to ensure that the data set remains correctly ordered while his DATA step runs. Here's an example.

```
data composit;
   merge datalib.fuel (cntllev=mem) fuel96;
   by grade;
run;
```

John cannot create a new version of DATALIB.FUEL, but he can use a MODIFY statement in a DATA step to update the shared data set. Here's an example.

```
data datalib.fuel;
   modify datalib.fuel;
   if (grade='03N') then
       do;
           grade='3Np';
           revised=today();
           replace datalib.fuel;
       end;
run;
```

When John uses the preceding DATA step to update an observation that Maria is editing in her FSEDIT window, the replace operation for that observation fails because Maria has the observation locked. This failure causes the DATA step to terminate as soon as the locked observation is encountered. However, any observations that are updated before the termination retain their updated values.

For applications that update shared data by using a MODIFY statement, it is very important to include error-checking statements to prevent failure in the updating process and premature termination. The automatic variable _IORC_ includes the return codes from the read operation (performed by the MODIFY statement) and the update operations (performed by the REPLACE, OUTPUT, and REMOVE statements). The preceding DATA step would be more effective if it was written as follows:

```
data datalib.fuel;
   modify datalib.fuel;
   if (grade='03N') then
       if (_iorc_ = 0) then
         /* read with lock for successful update */
           do;
             grade='3Np';
             revised=today();
             replace datalib.fuel;
           end;
       else
             put 'Observation' _n_
                '(fuel' fuel ') was not replaced.';
run;
```

The preceding DATA step checks the value of _IORC_ to determine whether a warning or an error condition occurred while reading an observation in DATALIB.FUEL. If the observation was read successfully, it can be replaced. If the observation could not be

read, a message is written to the SAS log to record the failure and to identify the observation that was not updated.

To check for specific values of _IORC_, use the SYSRC macro. For example:

```
data datalib.fuel;
   modify datalib.fuel;
   if (grade='03N') then
       if (_iorc_ = 0) then
          /* read with lock for successful update */
           do;
               grade='3Np';
               revised=today();
               replace datalib.fuel;
           end;
       else if (_iorc_ = %sysrc(_SWNOUPD)) then
                put 'Observation' _n_
                   '(fuel' fuel ') was not replaced.';
       else
                put 'Observation' _n_
                   '(fuel' fuel ') read with rc' _iorc_;
run;
```

For complete information about the MODIFY statement, see *SAS Statements: Reference*. For information about the SYSRC macro and _IORC_ return code checking, see *SAS Macro Language: Reference*.

### Using Ordered Data in a Shared Environment

Many applications that use SAS data sets require the data to be stored in sorted order according to the value (or values) of one or more variables. Beginning in SAS 6, indexes can be defined for one or more variables in a SAS data file to help SAS applications maintain the order of the observations in SAS data sets. This prevents the application from having to sort the entire data set for each use. Because SAS detects if indexes are used in its processing, indexes must be carefully defined to avoid inadvertently causing less efficient SAS performance. For more information about defining indexes, see *SAS Language Reference: Concepts*.

Shared SAS data sets are frequently ordered according to one or more variables. Programmers who develop SAS applications that use shared, ordered data should be aware of the following ways in which shared data can be used:

- concurrent-update applications
- reporting applications

Concurrent-update applications usually involve several users who repeat the following type of cycle: select an observation, update data; select another observation, update that data; and so on. If these users specify a WHERE clause to move to the next observation and the variable (or variables) in the WHERE clause are indexed, indexing can improve the server's performance by minimizing the server's effort to search for each observation. Because the users' access pattern when using the concurrent-update applications is often random instead of sequential, processing with an index does not usually increase the amount of physical I/O that is performed by the server for each user.

Reporting applications frequently read the data of one or more shared data sets, capturing the data as it is at that moment, and develop a report from that data. If the application uses a BY statement to return the data in sorted order, the server's performance can vary greatly while the data is being read. The server's performance is

based on multiple factors, such as whether the BY variable is indexed, and whether options are added to the BY statement that result in the index not being used.

To optimize the server's performance, the server needs to read the data in its physical, unsorted order, and then sort the data in the SAS process that is used to produce the report. You can do this by using the SORT procedure to read the data in physical order through the server, and produce a sorted data file in your library WORK. Here's an example.

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

Alternatively, you can use the SQL procedure to create a temporary SAS data file and sort it by using an ORDER BY clause. Here's an example.

```
proc sql;
create table fuel as
   select * from datalib.fuel
   order by area;
```

Defining more indexes than are necessary on shared SAS data sets can increase the amount of memory that a server needs. Avoid defining indexes that will not be used by your applications when they access shared data sets through a server.

## *Using Non-interactive SAS Applications in a Shared Environment*

Shared data is sometimes maintained by SAS applications that use the batch or a non-interactive method of processing. As in interactive applications, these non-interactive applications update SAS files through a server. Non-interactive applications can be written as one or more SCL programs or as a combination of DATA steps and procedures.

Usually, it is important that no other users access any of the shared SAS files while a non-interactive application runs. To ensure uninterrupted access, use the LOCK statement or the SCL LOCK function (for SCL programs) at the beginning of your program to get exclusive access to the SAS files that your application uses. After your program has completed, be sure to release your exclusive access to these SAS files so that other users can access them.

Here is a two-step SAS program that includes a LOCK statement that opens a shared SAS data set and copies to another data set all data that has not been updated for one month. Then the program deletes the data from the original data set. The following example program gives exclusive access to a specific SAS file and clears the exclusive lock after the program has completed processing.

```
%libdef(datalib);

/* Try to get exclusive access to the SAS data set. */
lock datalib.fuel;

/* Did we succeed? If not, stop here. */
data _null_;
   put "SYSLCKRC=&SYSLCKRC";
   if "&SYSLCKRC" ^= '0' then
      abort return;
run;

/* Copy any observations that have not been updated in */
```

```
/* 30 days to a different, locally-accessed library.  */

data permlib.a;
   drop now;
   retain now;
   if (_N_=1) then now=today();
   set datalib.fuel;
   if (accdate<(now- 30)) then output permlib.a;
run;

/* Now delete those observations from the master file. */

proc sql;
delete from datalib.fuel where (accdate<(today()- 30));
quit;

/* Tasks completed. Release the lock on the master file. */

lock datalib.fuel clear;
```

*z/OS Specifics*

SAS does not support concurrent host sorts under the z/OS operating environment. Attempting to invoke a host sort while one is already running will cause SAS to revert to the internal sort, which might have an adverse effect on performance. Attempts to run concurrent sorts usually occur in a server environment, but running sorts in a server environment is not recommended.

## SQL Programming Considerations

The REMOTE engine supports the SQL procedure pass-through facility, which allows you to pass SQL statements to a SQL server or a DBMS through a SAS/SHARE or a SAS/CONNECT server.

You can use RSPT to reduce network traffic and to shift CPU load by sending requests for data to a server.

*Note:* If the server is a SAS/CONNECT server, you can also remotely submit queries by using the RSUBMIT statement and achieve the same goals.

For example, consider the following statement:

```
select emptitle as title, avg(empyears), freq(empnum)
       from sql.employee
       group by title
       order by title;
```

SQL is the libref for a library that is accessed through a SAS/SHARE or a SAS/CONNECT server. Each row in the table EMPLOYEE must be returned to your client in order for the summary functions AVG() and FREQ() to be applied to them. However, you might specify the statement as follows:

```
select * from connection to remote
       (select emptitle as title,
            avg(empyears), freq(empnum)
            from sql.employee
```

```
            group by title
            order by title);
```

In this case, the query is passed through the SAS/SHARE server to the SAS SQL processor, which processes each row in the table EMPLOYEE and returns only the summary rows to your client.

You can also use RSPT to join server data with client data. For example, you might specify the statement as follows:

```
libname mylib 'c:\sales';

proc sql;
    connect to remote (server=mvs.shr1 dbms=db2
    dbmsarg=(ssid=db2p));

    select * from mylib.sales97,
                 connection to remote
                    (select qtr, division,sales, pct from revenue.all97
                        where region = 'Southeast')
       where sales97.div = division;
```

In this case, the subquery against the DB2 data is sent through the SAS/SHARE server to the DB2 server. The rows for the divisions in the Southeast region are returned to your SAS/SHARE client, where they are joined with the corresponding rows from the client data set MYLIB.SALES97.

If your server is a SAS/CONNECT server, you can also use RSPT to send non-query SQL statements to a DBMS. For example, the following statements send the DELETE statement in PROC SQL through the SAS/SHARE server to the Oracle server.

```
proc sql;
   connect to remote (server=sunserv dbms=oracle);

   execute (delete from parts.inventory
           where part_bin_number = '093A6')
           by remote;
```

# SCL Programming Considerations

## *Concurrent SCL Applications*

You can use SAS Component Language (SCL) with SAS/SHARE software to access data through a SAS/SHARE server. SCL has the ability to read and update SAS tables that are used concurrently by other clients or SCL applications. For complete information about SCL, see *SAS Component Language: Reference*.

A concurrent SCL application opens one SAS data table for update while other SAS operations (possibly in different SAS sessions) have the same data table open for update. You can open other data tables for update by using other invocations of the first SCL application, using a different SAS application or SCL application, or using the FSEDIT or FSVIEW procedure on the table.

Consider the following issues when writing an SCL application that updates data concurrently:

- locking rows in SAS tables

- implications of locking rows in SCL
- programming in PROC FSEDIT and PROC FSBROWSE
- locating and fetching control rows
- unlocking a row

For an application that uses SAS tables that contain inventory and ordering information for each product in a store, see "SAS Component Language (SCL) Application" on page 237. The purpose of the application is to automate a system that develops orders and maintains the inventory list while sales representatives simultaneously write orders for products.

### Locking Rows in SAS Tables

A row in a SAS table is locked implicitly when it is read by a SAS procedure, a DATA step, or an SCL application. A lock on a row is held until a different row is read or until the SCL application calls the UNLOCK function.

When a SAS table is opened for update, only one row can be locked at a time. However, a SAS table can be opened for update more than one time in the same SAS session or in different SAS sessions (through a server), and a different row can be locked by each user who opens the table. For example, if two users are running an SCL application that calls the OPEN function to open a SAS table for update, row 7 can be locked by one user and row 10 can be locked by the other user.

### Locking Rows in SCL

Row locking can give a programmer an important advantage, and should be a key consideration in concurrent SCL programming. While an SCL application has a row locked, no other SAS operation (especially in another SAS session) can alter or delete that row. After a lock on a row is released, your application cannot know that the values in that row remain the same; another user might have already modified the values. Any data modifications that you make that are based on the old values might damage the data integrity of the system.

Therefore, you must never assume that the data values in a specific row will not change in a shared table, even though only a very brief amount of time has elapsed between consecutive reads and locks of the row.

When each row in a SAS table can represent a specific instance of a resource that the application must govern, row locking provides a resource-specific, protected period of time in which the application can safely test and modify the state of the resource.

An example of a specific-resource instance is information about one of your customers or the number of items of a specific type that is currently in inventory. The SCL example in "SAS Component Language (SCL) Application" on page 237 applies locks to its inventory table to maintain the correct inventory count for each item, even if several sales representatives are simultaneously writing orders for those items.

### Programming with PROC FSEDIT and PROC FSBROWSE

Unlike other SCL environments (such as SAS/AF software and the FSVIEW procedure), PROC FSEDIT and PROC FSBROWSE give the SCL programmer multiple labeled sections for structuring an SCL application. The sequence in which these procedures run some of the sections has several implications for concurrent SCL programming.

The INIT section is especially useful in applications that read and update shared data. The initial values of the columns in a row (as currently stored in the SAS table) can be preserved in SCL columns. Preserving initial values in SCL columns is important for applications that update auxiliary tables that are based on the PROC FSEDIT user's modification or on the creation of a row in the primary table. SCL applications that read and update data usually need to perform the following tasks:

- determine whether a modified window column actually contains a value that is different from its initial, validated value
- explicitly restore the initial values if the user's modifications to a row in the primary table are not allowed because they could not be validated
- modify the auxiliary tables because changes (from their initial values) were made to the values in a row in the primary table

Although the MAIN or TERM sections must validate the user's modifications to a row in the primary table and update auxiliary tables, it is usually best that no row of an auxiliary table remain locked between executions of these sections. Such locks prevent other users or applications from modifying the row while a user is working in the current row in the primary table.

## Programming with the Data Table and Data Form Classes

The Data Table and Data Form classes in SAS/AF FRAME entries allow you to specify an SCL entry to use for the model SCL. This SCL entry is separate from the frame's SCL entry. Usually, model SCL is used to initialize computed columns and to perform error checking and data validation.

As in PROC FSEDIT, the Data Table and the Data Form objects give the SCL programmer multiple labeled sections for structuring the order in which events will occur for each row in the table. These sections, which include INIT, MAIN, and TERM, work in the same way as explained in "Programming with PROC FSEDIT and PROC FSBROWSE" on page 51.

If multiple instances of the Data Table or the Data Form objects are displayed within a single SAS/AF FRAME entry, the objects share data, then the model SCL for each data table or data form runs separately. The application developer must remember whether a previous object has a lock on a row that the current object attempts to read or update. In addition, the frame SCL might also be working on the shared data, and timing within the frame could be critical. For more information about when SCL labels are run, see *SAS Component Language: Reference*.

## Locating and Fetching Control Rows

SCL provides a set of functions that are useful for locating and fetching the required auxiliary table rows (observations) in a data-concurrent SCL application. However, you should use caution with these functions in applications that access shared data. The return code, which is obtained directly from the called function or from the SYSRC function, must be checked to ensure that a lock was obtained on the row or that an update was successful. The return value, which is generated by the macro invocation %SYSRC(_SWNOUPD), is generated when a fetch or update function fails to lock or update the row because it is locked by another application.

The FETCHOBS table function is useful when the row number can serve as the row identifier. Remember that the FETCHOBS function accepts a relative row number by default. That number might or might not equate to the physical row number. If you can

delete rows in the auxiliary table, you probably want to use the ABS option in the FETCHOBS function for absolute row numbering.

The LOCATEC and LOCATEN table functions can be useful for finding rows in small tables when the data can remain sorted by a unique identifier (column) and a binary search is specified. However, due to the overhead of searching with these SCL functions, it is better to use the WHERE and FETCH functions to find the rows. In a shared-data environment, when you use the LOCATEC and LOCATEN functions to find rows, each row must be requested from the server and transmitted to the client's SAS session.

The SYSRC function must be queried for warnings when the LOCATEC and LOCATEN functions find a row because these functions only return a return code of 0 for either condition: row found or row not found. For more information about the LOCATEC and LOCATEN functions, see *SAS Component Language: Reference*. The following SCL program example checks whether the located row is locked by another task:

```
gotrec=locatec(data-set-id,var-num,search-string,
               sort-order);
if (gotrec<=0) then do;
   /* Handle row not found */
 end;
else if (sysrc()=%sysrc(_swnoupd)) then do;
   /* Handle row locked */
 end;
```

*Note:* The LOCATEC and LOCATEN functions cannot perform binary searches on compressed tables, SAS data views, or SAS data files that have deleted rows.

The more general and, usually, more efficient way to find a row is to use the WHERE function followed by a FETCH function call. The WHERE clause is evaluated in the server's SAS session, and only the row that needs the specifications in the WHERE clause is transmitted to the client's SAS session.

If the WHERE clause does not find the specified row, the FETCH function returns a −1 return code, which indicates that the end of the table has been reached. If the WHERE clause is cleared by issuing a null WHERE function call, the next FETCH call that the application issues fetches the first row in the table. The FETCH call, not the WHERE clause, locks the row (if possible). Notice that the WHERE function returns a harmless warning, %SYSRC(_SWWREP), when the WHERE clause is replaced.

The DATALISTC and DATALISTN selection-list functions help a client to select a valid row. These functions actually fetch the entire selected row into the Table Data Vector (DDV) and lock the row (if possible). Because these functions do not return a system return code, the SYSRC function must be queried for warnings. The DATALISTC and DATALISTN functions might cause the entire SAS table to be read, which means that each row that is read is transferred individually from the server to the client SAS session.

### Unlocking Rows

In addition to releasing a lock on the current row by reading another row, an SCL application can use the SCL function UNLOCK. The UNLOCK function leaves the read-pointer at its current position in the table and does not update the DDV.

The OBSINFO function in SCL returns information about the primary table's current row in an FSEDIT application. You can query whether the row has been deleted, locked, or newly created. A row does not attain deleted status until the DELETE command is run on the client. For example, if you specified the CONTROL ENTER statement to

force your MAIN section to run, the OBSINFO function will not return a deleted status when issued from the MAIN section (because the DELETE command that was executed on the client has caused MAIN to be run.) However, the OBSINFO function will return a deleted status when the MAIN statement or TERM section is run again.

# SAS Data View Programming Considerations

## *Data Sets of Type VIEW*

Beginning with SAS 6, SAS data sets can be classified as member type DATA (SAS data file) or member type VIEW. A data set of type VIEW is called a SAS data view. It contains a definition or description of data that is stored elsewhere. SAS data views can be created by using a DATA step, the SQL procedure, or the ACCESS procedure in SAS/ACCESS software. In most SAS programs, whether the data comes from a SAS data view or data file is not important.

Many SAS/ACCESS interface products, such as DB2 or Oracle, enable you to update product data by using a SAS/ACCESS view. However, for views that are interpreted in the server's session, whether you can update a view's underlying DBMS data depends on the specific SAS/ACCESS interface engine that you are using. For information about how to use SAS/ACCESS engines in a SAS/SHARE server session, see the SAS/ACCESS documentation.

## *Interpreting SAS Data Views*

A SAS data view that is accessed by using a server can be interpreted in either the server or the client session.

The user specifies where a SAS data view is interpreted by specifying the RMTVIEW= option in the LIBNAME statement. When RMTVIEW= YES or the option is omitted, a data view is interpreted in the server session. When RMTVIEW= NO, a data view is interpreted in the client session. For more information about the LIBNAME statement and its options, see Chapter 10, “Remote Library Services,” on page 123.

Interpreting a view consists of loading and calling the view engine to read the view's underlying data. When a view is interpreted in the client session, the view engine is loaded and called by the client to read and present the underlying data and present it as a SAS data set. When a view is interpreted in the server session, the view engine is loaded and called by the server to read and present the underlying data.

Whether a SAS data view is interpreted in a client or a server session, the underlying data must be accessible to its view engine. Data accessibility is based on whether the view was created by using a DATA step, PROC SQL, or PROC ACCESS.

For DATA step views, “accessible” means that any external file (or files) must be available and that any filerefs and librefs that are stored in the view must be defined in the SAS session in which the view is interpreted.

For PROC SQL views, “accessible” means that all librefs that are used in the view must be predefined in the SAS session in which the view is interpreted, or included in the USING clause of the query that is stored in the view. This libref can be associated with a SAS library that is accessed through a server or a library that is stored at the client. You do not have to specify a libref in a PROC SQL view for data sets that are in the same data library as the view itself.

For PROC ACCESS views, “accessible” means that the interface product and its data, as well as the SAS/ACCESS interface view engine, must be available to the SAS session in which the view is interpreted.

Where SAS data views should be interpreted in a shared environment is based on the following:

- how the view was created
- how the view's data will be used
- specific site considerations

Is the underlying data accessible? If the data is accessible only from one of the sessions, the view must be interpreted there.

If the data is accessible from the client session and the server session, then performance must be a consideration. If interpreting the view requires the SAS session to read a large number of rows in order to select a small subset, having the SAS/SHARE server interpret the view greatly reduces the number of records that are transmitted to the client session. This method reduces network load and might be faster than having the client session interpret the view. However, putting a heavy processing load on the server (especially if joins are involved) might adversely affect server performance for other clients.

If the view selects most of the input rows or if the selection criteria are processed by a DBMS server, interpreting the view in the client session is probably preferable.

### *Example: Using RLS and a DATA Step View to Improve the Performance of PROC APPEND*

The following code shows the creation and storage of a master data set on a SAS/SHARE server:

```
libname share 'path-to-library' server=shr1;

data share.master;
   do key = 1 to 100000;
      var = 'Original Value' ;
      output;
   end;
   array othervars {20};
run;
```

The following program creates a transaction data set, which PROC APPEND adds to the master data set that is located on the server:

```
data transactions;
   do key = 100001 to 200000 by 10;
      var = 'New Value: '||put(key,z6.);
      output;
   end;
run;

proc append base=share.master new=transactions force;
run;
```

With the following changes to the preceding code, append processing can be shifted to a SAS/SHARE server.

*Note:* When using a SAS/SHARE server, you will see performance gains increase as the size of the transaction data set increases.

The SAS DATA step view can be created and stored on the server before the view is actually used. In order to create the view, a copy of the transactions data set (in this example, **swork.transactions**) must be available as a "template" for the expected contents (for example, variables and variable length). The template does not have to contain any observations.

```
libname swork slibref=work server=shr1;

   data swork.transactions;
      do key = 100001 to 200000 by 10;
         var = 'New Value: '||put(key,z6.);
         output;
      end;
   run;

   data share.append_view share.master / view=share.append_view;
      modify share.master;
      set swork.transactions;
      output;
   run;
```

The following DATA step now references the view that is located on the server and causes the view to be interpreted and run on the server:

```
data _null_;
   set share.append_view;
 run;
```

After the initial setup of the DATA step view, future production runs that use the Append view might appear as follows:

```
libname share '' server=shr1;
   libname swork slibref=work server=shr1;

   data swork.transactions;
      do key = 100001 to 200000 by 10;
         var = 'New Value: '||put(key,z6.);
         output;
      end;
   run;

   data _null_;
       set share.append_view;
   run;
```

Although there are several ways to create SAS DATA step views in order to improve the performance of the update process when using a SAS/SHARE server, the preceding example shows a simple way to create views. The update process can be improved by using the MODIFY statement in a DATA step view that gets interpreted at the server.

*Note:* You should consider the advantages and disadvantages of using the MODIFY statement to create views. A primary advantage is that the MODIFY statement allows locking at the observation level rather than at the member level. Locking at the observation level is less restrictive than locking at the member level. However, the MODIFY statement is not the most efficient query technique.

# Using SAS Catalog Entries in Programs

Many catalog entries of the following types are stored in the library SASUSER. However, because you cannot access them through a SAS/SHARE server, you are advised against storing them in a SAS library that is accessed through a server.

***Table 4.1*** *Catalog Entry Types That Cannot Be Accessed through a SAS/SHARE Server*

| | |
|---|---|
| AFCBT | MODEL |
| AFGO | MSYMTAB |
| AFPGM | OLDMACRO |
| ENGINE | PROFILE |
| GEDIT | STATGRAP |
| GLOBAL | TITLE |
| GOPTIONS | WSAVE |

*Note:* You cannot move a stored compiled macro to another operating system. You can, however, move the macro source code to another operating system where you can then compile and store it. You might need to recompile these macros if you use them in a different release of SAS. For more information, see your host companion.

Other than stored compiled macros and the catalog entry types listed in Table 4.1 on page 57 you can gain full access to all other types of entries through a SAS/SHARE server.

You can obtain exclusive access to a catalog or to individual entries (other than the types shown in the preceding table) by using the LOCK statement or the LOCK command to lock the catalog or catalog entries. See Chapter 5, “Locking SAS Data Objects,” on page 59 and Chapter 13, “The LOCK Statement and Command,” on page 161. You can also lock a catalog or catalog entry by using the SCL function LOCK (see the LOCK function in *SAS Component Language: Reference*).

# Using SAS/CONNECT with SAS/SHARE

## *SAS/CONNECT Used with SAS/SHARE*

You can use SAS/CONNECT with SAS/SHARE to extend your access to SAS files and execute SAS tasks on one or more servers. All output and messages that are generated from that server session are directed back to the client for display. This execution can be done in parallel to provide scalability of large jobs and reduce the amount of time to completion.

SAS/CONNECT provides the connection between the client and the server that enables you to run SAS statements on the server, and gives you the ability to perform this execution in parallel. SAS/SHARE allows concurrent update access to data. You must use SAS/SHARE with SAS/CONNECT if SAS files that are used in server processing require concurrent update access. For complete details about SAS/CONNECT, see the *SAS/CONNECT User's Guide*.

## *Example: Using a SAS/SHARE Server in a SAS/CONNECT Server Session*

The following example shows the need to access a SAS/SHARE server in a SAS/CONNECT server session.

You have to create a report from DATALIB.FUEL, but John and Maria are currently accessing this data through a SAS/SHARE server. You can use SAS/CONNECT to connect to the operating environment where the library DATALIB is stored. However, because DATALIB is already being accessed through a SAS/SHARE server, you must use the same SAS/SHARE server to access DATALIB.FUEL and generate the report. Therefore, you connect to the SAS/SHARE server and submit the following code:

```
signon apex;
rsubmit;
   libname datalib server=shr1;
   proc print data=datalib.fuel;
      where area='TEX3' and profits<=0;
      title 'Losses in Texas, Area 3';
   run;
endrsubmit;
```

The LIBNAME statement identifies DATALIB as the library to access through the same server that John and Maria are using. Your SAS/CONNECT server session connects to the SAS/SHARE server and executes the PRINT procedure to produce the report. The report is displayed at your client session. Except for some interactive limitations that are imposed by SAS/CONNECT, you can remotely submit the same SAS program statements to read from or write to the same data in DATALIB that other users work on when they log on directly to the server.

***CAUTION:***

**Do not remote submit the SERVER procedure when using SAS/CONNECT.**

*Chapter 5*

# Locking SAS Data Objects



## SAS/SHARE Lock Manager Facility

*Note:* In this documentation, the term "operation" refers to any SAS procedure, statement, or command.

The SAS/SHARE lock manager facility enables multiple clients to share the same SAS file concurrently. Using a set of complex locking rules, the lock manager evaluates each incoming client request for access to SAS data objects while monitoring the status of all other client activities. The lock manager grants access to a specific data object by locking the data object, and denies all other requests for the locked data object until an operation has been executed or the lock has been cleared explicitly.

# Locking and SAS Data Object Hierarchy

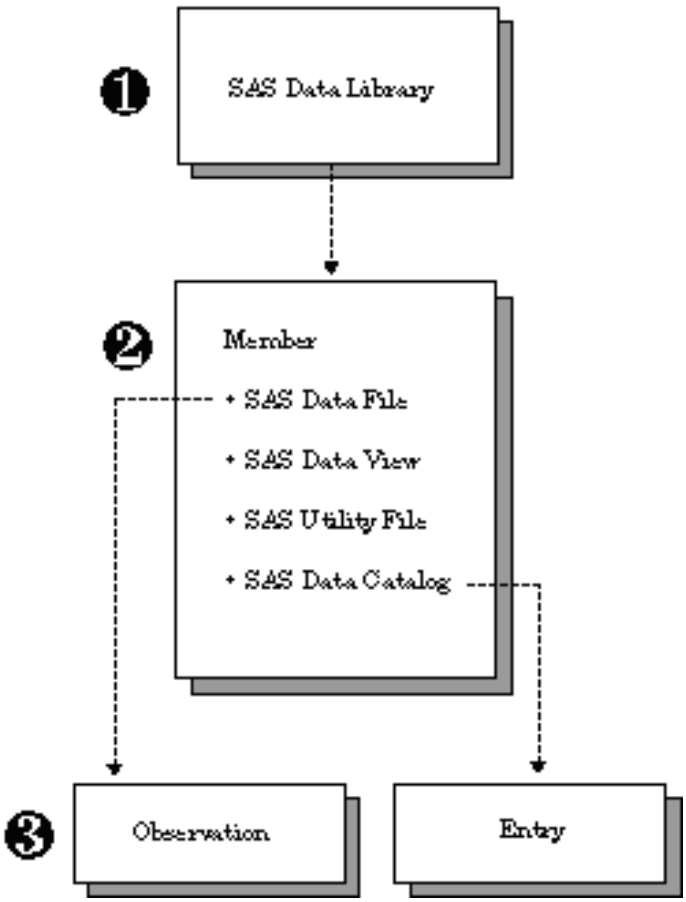
## SAS Data Object Hierarchy

The information in this section is primarily directed to applications developers, but it might also be of interest to end users.

Knowing the concepts for locking SAS data objects and the SAS data object hierarchy will help you to understand explicit locking, which is set by using the LOCK statement or a LOCK command; and implicit locking, which is set automatically.

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

***Figure 5.1*** *Hierarchy of SAS Data Object Types*




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

2 Member is a file in a SAS library that can be a SAS data file, a SAS data view, a SAS utility file, or a SAS data catalog.

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

SAS data view
: is a SAS data set 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
: is a SAS file that stores information that is exclusive to a component of SAS. For example, SAS/ACCESS descriptors, MDDB (Multi-Dimensional Database) files, and DMDB (Data Mining Database) files.

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

3 Observation is a row in a SAS data file that contains a collection of data values that are associated with a single entity, such as a customer or a state. Each row (observation) contains one data value for each column (variable) in the data file.

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

## *Accessing and Using SAS Data Objects*

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. Here are the ways to access a data object:

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 action for each object that is accessed and the way that the object is accessed. For example, given that the server engine allows an observation to be locked and the observation is not already locked, the server can open and lock an observation in a data set. If the server engine does not allow an observation to be locked, the engine locks the member (above the observation).

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

- V8 and V9 (the default) engines allow locking at the library, member, and observation level.
- The V8TAPE engine, V9TAPE engine, and other sequential engines allow locking only at and above the member level.

- If an engine does not allow access to SAS catalogs, that engine does not allow locking at any level.
- The view engine default-locking action is based on how the view is created, that is by using a DATA step, PROC SQL, or PROC ACCESS (available in SAS/ACCESS). The specific SAS/ACCESS engine that is used is based on the DBMS. See the SAS/ACCESS documentation for information about view engine default-locking action.

The following table 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*

| Which Data Object Is Locked | Mode in Which Data 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

The following types of locks can be used on a data object:

explicit lock
: is set with the LOCK statement or the LOCK command for exclusive access to the data object type. The action 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, see Chapter 13, “The LOCK Statement and Command,” on page 161.

implicit lock
: is automatically set on the data object type as required by the SAS operation that is being executed. Each SAS 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, a DATA step that includes 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 library (higher-level lock) that contains it, the data library is locked implicitly. An

implicit lock allows other users to access the locked data library, even though you have exclusive access to the locked data set.

# Locking Objects Explicitly (LOCK Statement)

## *LOCK Statement: Advantages*

Explicit locks protect data while it is being updated in a multi-step SAS program. For example, a nightly update process that uses a DATA step to remove observations that are no longer useful 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.

To set an explicit lock, execute a LOCK statement before the first DATA step to acquire exclusive access to the file. If exclusive access cannot be obtained, the LOCK statement return code &SYSLCKRC is issued to indicate that, and the update program can reschedule 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 exclusive access to the data object and an explicit lock on data libraries, data sets, catalogs, and catalog entries. No other users can read from or write to a data object that you have locked by using this statement.

When you use a LOCK statement to lock a data object, you can open that data object as often as you want to and in any mode that you want. For example, you can create, replace, update, or read from the object, if 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 library through a server before you can lock that library or any data object in it. Also, you cannot lock a data object that another user has open.

## *Syntax for the LOCK Statement*

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

*libref*
is the name temporarily associated with a SAS library.

*member-name*
is the name of a member of the referenced SAS 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 using the LOCK statement. This argument is optional.

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

## Locking a SAS Library

The following statement locks the SAS library MYLIB.

```
lock mylib;
```

Locking a 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 accessing SAS files in the library while it is locked.

## Locking a SAS Data Set

The following statements lock the SAS data set FUEL in the library MYLIB. All these statements are equivalent.

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

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

Beginning with SAS 6.06, 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 that a SAS data file and a SAS data view that have the same name be locked at the same time. Therefore, when you execute the LOCK statement on one of these data sets, both of them are automatically locked. In the statements given at the beginning of this section, the server locks the SAS data file MYLIB.FUEL.DATA and the SAS data view MYLIB.FUEL.VIEW concurrently. For more information about SAS data sets, see *SAS Language Reference: Concepts*.

***CAUTION:***
**The LOCK statement does not lock 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 session.

### *Locking a SAS Catalog*

The following statement locks the member MYCAT in the library SCLLIB. The member type CATALOG indicates that MYCAT is a SAS 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 creating, reading, updating, deleting, or renaming any of the entries in the catalog by other users.

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. This ensures that no other users are executing your application while you are updating its entries. After you have updated all the entries and tested your application, you can clear the lock by using the argument CLEAR in a LOCK statement. This allows other users to gain access to your catalogs and to execute your application. For more information, see "Clearing an Explicit Lock" on page 65.

### *Locking a Catalog Entry*

The following 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*

#### *GENERATED SUBSUBTOPIC TITLE*

How you clear an explicit lock depends on the level in the data object hierarchy at which the lock was obtained. There are three ways to clear locks. Each is explained in detail in the sections that follow.

- 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. This allows you to clear the single higher-level lock after you have finished accessing the lower-level objects.

#### *Explicitly Locking and Clearing Each Data Object*

When you explicitly lock a specific data object, you must clear each lock individually. Here is an example.

```
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 in the preceding example sets implicit locks on the SAS library EDUCLIB and on the SAS catalog EDUCLIB.MYCAT. Then it 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 argument CLEAR 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 argument CLEAR releases the explicit lock on the catalog entry CHOICE2.MENU. Because no catalog entries remain locked, the argument CLEAR releases the implicit lock on the SAS catalog EDUCLIB.MYCAT. Also, because no members of the library are locked, this argument clears the implicit lock on the SAS library EDUCLIB.

### *Clearing a Higher-Level Data Object to Clear Multiple Lower-Level Objects*

You can set explicit locks on data objects at low levels. However, when you clear a higher-level implicit lock, all of the lower-level explicit locks are cleared automatically. Here is an example.

```
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 in the preceding example sets implicit locks on the SAS library EDUCLIB and on the SAS catalog EDUCLIB.MYCAT. Then it 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 argument CLEAR 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, this argument also clears the implicit lock on the SAS library.

### *Locking a Higher-Level Data Object to Lock Multiple Lower-Level Data Objects and Clearing the Higher-Level Lock*

To update several lower-level data objects without having to lock each one separately when all the 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 that higher-level data object.

However, you might 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 instance, you can explicitly lock the lower-level data objects that you need and 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.

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

If one of your co-workers needs to work on some SAS files in the library EDUCLIB that you are not updating, you can lock the SAS files in the library EDUCLIB that you need, by using the following statements:

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

Then, use the following statement to 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 your implicit lock on the library, as follows:

```
lock educlib clear;
```

### *Listing Lock Status*

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

*object* is *status* by *whom*

*object*
SAS 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. Here are some examples of messages.

In the first example, the SAS library that is referenced by MYLIB is locked by user SASUSER(1). A lock on a library prevents other users from reading, updating, or deleting existing SAS files or from creating new SAS files in that library. The lock also prevents other users from obtaining a list of files in the library. The lock 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. You must wait for user SASUSER(1) to unlock the library before you can use it.

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

In this example, 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 users are reading catalog entries or adding entries to the catalog. Although you can browse the catalog or add entries to the catalog, you cannot attempt to lock the catalog until there are no others using it.

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

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

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

The following LOCK statement lists in the SAS log whether a specified data object is locked and by whom. The format used in the LOCK statement for listing lock status is

*data-object* is *status* by *whom*

```
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 a LOCK statement. A nonzero value in SYSLCKRC results when you use a LOCK statement with the argument LIST to list a lock.

## Locking Explicitly in a SAS Window (LOCK Command)

### Advantages of Using the LOCK Command

The LOCK command provides a convenient way to lock data objects that are in a SAS window. As with the LOCK statement, you can use the LOCK command to obtain an explicit lock on data libraries, data sets, catalogs, and catalog entries.

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.

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

### Syntax for the LOCK Command

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

*libref*
is the name that is temporarily associated with a SAS library.

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

*member-type*
is the type of 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 a specified data object that was acquired in your SAS session by using the LOCK command. This argument is optional.

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

## Locking and Clearing Locks on Data Objects

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 13, "The LOCK Statement and Command," on page 161.

The following SAS log shows the message in the Log window that lets you know that the catalog MAPSLIB.MAPSCAT.EUROMAP.CMAP has been locked successfully. In the Program Editor window, the LOCK command was issued to obtain a lock on the catalog MAPSLIB.MAPSCAT.EUROMAP.CMAP.

***Log 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
```

The following Program Editor window shows a LOCK command that contains the argument CLEAR to release the lock on the catalog MAPSLIB.MAPSCAT.EUROMAP.CMAP.

***Log 5.2*** *Releasing a Lock on a Catalog Entry*

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

 00001
 00002
 00003
 00004
 00005
 00006
```

In the next SAS log, the messages in the Log window show that MAPSLIB.MAPSCAT.EUROMAP.CMAP was successfully unlocked. The log also displays the name of the user who clears the lock. In this example, the user who set and cleared the lock is referred to as "you."

***Log 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 concurrently to the SAS data set FUEL in their respective PROC FSEDIT sessions.

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. The following SAS log shows this part of John's session.

***Log 5.4*** *Log Window - PROC SORT on 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 "Listing Lock Status" on page 67.

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 correctly. Notice 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 not be sorted correctly when the PRINT procedure executed.

To avoid the conflict and ensure that John 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;
```

The preceding PROC SORT statement opens the data set DATALIB.FUEL only for input with observation-level control. Then John can use the PRINT procedure to display the temporary data set WORK.FUEL.

# Defaults for Selected SAS Operations

## *Default Data Objects: Reference*

Knowledge of the default data objects and how they are accessed will help you to anticipate the results from specific operations when you write your application or issue

SAS statements in interactive mode. The following table shows the defaults for some frequently used SAS operations when locking is executed.

***Table 5.2*** *Defaults for Selected SAS Operations*

| SAS Operation | What the Data Object Is Locked For | The Data Object Locked (Default) |
|---|---|---|
| DATA step | | |
| DATA statement | | |
| without MODIFY statement | Output | Member |
| with MODIFY statement | Update | Observation |
| SET statement | | |
| without POINT= and KEY= options | Input | Observation |
| with POINT= and KEY= options | Input | Member |
| MERGE statement | Input | Observation |
| MODIFY statement | | |
| without POINT= and KEY= options | Update | Observation |
| with POINT= and KEY= options | Update | Member |
| UPDATE statement | Input | Observation |
| Procedures | | |
| APPEND procedure | | |
| BASE= option | Update | Member* or 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 | | |

| SAS Operation | What the Data Object Is Locked For | The Data Object Locked (Default) |
|---|---|---|
| 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 |
| OUT= option | Output | Member |
| SQL procedure | | |
| CREATE TABLE statement | Output | Member |
| DELETE statement | Update | Observation |
| INSERT statement | Update | Member |
| UPDATE statement | Update | Member |

* If no other tasks are currently accessing the BASE= data set, then PROC APPEND opens the data set with a member lock.

### *Changing the Data Set Option Default Object*

In some SAS operations, 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 to specify the object that you want. For example, in a SET statement that contains the POINT= option, you can change the default from member to observation by specifying the CNTLLEV= data set option. In this example, the value, REC (for record), means the same as observation.

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

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

You can also change the data object 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 reset the data object observation to member.

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

In some SAS operations, 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. Therefore, even if you specify CNTLLEV=REC in such a DATA statement, the DATA step tries to set the data object as member, but this will fail if other operations are accessing the data set.

*Note:* Be careful when using the CNTLLEV= option in 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 a procedure has this requirement, a warning is issued but the procedure will allow its objects to be reset if you use the CNTLLEV= option.

For details about the syntax of the CNTLLEV= option in the SET statement, see "CNTLLEV= Data Set Option" in *SAS Data Set Options: Reference*.

*Chapter 6*

# SAS/SHARE Macros for Server Access



## Using Macros for Server Library Access

### *Overview of Macro Usage*

The information in this section is recommended primarily for server administrators and programmers who write applications that access shared data. For complete details about each macro that you can use to access a server and its libraries, see Chapter 14, "SAS/SHARE Macros," on page 165.

Programs that use SAS/SHARE must include a LIBNAME statement that identifies the SAS/SHARE server through which a specified library will be accessed. Adding servers and changing server IDs can require users and server administrators to obtain current server ID information each time they want to access a server. That could make maintaining production or utility programs difficult.

Although there is no permanent connection between a SAS library and a specific server, there is frequently a logical connection. Programmers, server administrators, and users might always want to access a specific library through the same server because only one server at a time can provide access to a library. The same logical connection can also exist between a group of users and a server. This is an advantage if all the members of a department needed to use the same server, especially if they are sharing libraries.

To use SAS/SHARE most effectively without compromising performance, administrators often need a dynamic and flexible server environment. They need to be able to start and stop servers as the need arises, and to easily redistribute the load on the servers. They want to be able to switch libraries and users from one server to another quickly and easily. To balance the needs of both administrators and users, SAS/SHARE includes macros to be defined through the autocall function of the SAS macro facility.

These SAS/SHARE macros enable the administrator to define aliases for a server and to associate an alias with a specific library. Programs issue these macros to generate the requisite LIBNAME statements for accessing that library through the server that is associated with the alias. Then, the administrator can add servers, change server IDs, and switch libraries and users from one server to another with a process that is totally transparent to the program or the SAS user.

SAS/SHARE macros can be used to do the following:

- generate and display the tables of macro variables that associate libraries with server aliases and server aliases with server IDs
- generate part or all of a LIBNAME statement
- start and stop servers
- generate PROC OPERATE and SET SERVER statements

The server ID that is associated with an alias can be changed during any appropriate server or application outage (for example, down time). You update only the file that contains the table of macro variables that maps aliases to server IDs. Additionally, a library can be logically associated with a different server by updating the table that associates libraries with server aliases.

For example, a site might have four logical servers (that is, four different server aliases) but only one physical server by having all the aliases map to the same server ID. Whenever the load on that one server gets too heavy, the site can start an additional server and shift specific libraries and users to it by simply pointing one of the aliases to that new server.

### *Utility Macros*

The following utility macros can be used by all SAS/SHARE programs and sessions:

SHRMACS
: compiles all the other macros and builds the server-alias and library-alias tables.

SERVERID
: takes a server alias and looks up the server ID in the server-alias table and generates **`server-ID`** or SERVER=**`server-ID`**, as appropriate.

SERVIIDX
: returns the index of the entry in the server information table for the specified server.

LISTSRV
: writes the server-alias table to the log.

LISTLIB
: writes the library-alias table to the log.

LISTSRVI
: writes the server information table to the log.

### User Program Macro

The following is a user program macro:

LIBDEF
: takes a libref and an optional physical library name and looks up the SAS library name in the library-alias table and generates a LIBNAME statement.

### Server Administrator (Operator) Macros

The following server administrator macros are used in server administrator programs.

STRTSRV
: starts a server with the appropriate server ID by using the SERVERID macro to convert the alias. The STRTSRV macro takes a server alias and PROC SERVER statement options.

SHUTSRV
: generates the PROC OPERATE statement and STOP SERVER command for the appropriate server ID by using the SERVERID macro to convert the alias. The SHUTSRV macro takes a server alias and an optional password.

OPERATE
: generates PROC OPERATE statements for the appropriate server ID by using the SERVERID macro to convert the alias. The OPERATE macro takes a server alias and an optional password.

SETSRV
: generates a SET SERVER server ID statement by using the SERVERID macro to convert the alias. The SETSRV macro takes a server alias and an optional password.

## Macros Generated by the SHRMACS Macro

The SHRMACS macro compiles all other macros. A server administrator or an applications programmer must always invoke %SHRMACS before invoking any other macro. Use the following syntax:

%SHRMACS(*category*,<*log-info*,>
<APPLSYS=*app-sys-lib-tab*,>
<SASSAML=*alt-sys-lib-tab*>);

*category*
: specifies the category of macros to be compiled. Valid values for *category* are SERVER, USER, OPER, or ALL.

*log-info*
: specifies whether descriptive information is written to the SAS log about each macro. Valid values for *log-info* are NOMSG, MSG, or HELP. The default is MSG.

APPLSYS=
: specifies an alternate applications systems library-alias table. For details, see "The APPLSYS Macro Library" on page 79.

SASSAML=
specifies an applications systems library, which is a set of files that specify SAS libraries and servers. For details, see "The APPLSYS Macro Library" on page 79.

***Table 6.1*** *SAS/SHARE Macros Generated by the SHRMACS Macro*

| SHRMACS Macro Categories | | | |
|---|---|---|---|
| **Server** | **User** | **Operator** | **Implicit Macros Generated** |
| | • | | LIBDEF |
| • | • | • | LISTLIB |
| • | • | • | LISTSRV |
| • | • | • | LISTSRVI |
| | | • | OPERATE |
| • | • | • | SERVERID |
| • | • | • | SERVIIDX |
| | | • | SETSRV |
| | | • | SHUTSRV |
| • | | | STRTSRV |

For example, here are three possible macro definitions:

- **`%SHRMACS(SERVER)`**
- **`%SHRMACS(USER)`**
- **`%SHRMACS(OPER)`**

All of these macro definitions generate these macros: LISTLIB, LISTSRV, LISTSRVI, SERVERID, and SERVIIDX.

However, only %SHRMACS(USER) generates the LIBDEF macro, only %SHRMACS(SERVER) generates the STRTSRV macro, and only %SHRMACS(OPER) generates the OPERATE, SETSRV, and SHUTSRV macros.

In addition to compiling the requested macros, the SHRMACS macro also builds the appropriate library-alias table and server-alias table. These tables are used for generating the server name for the PROC SERVER, PROC OPERATE, and LIBNAME statements. The SERVER category of macros generates the server-alias table; the OPER category of macros generates the library-alias table; and the USER category of macros generates both the server-alias and the library-alias tables.

Specifying the ALL category of macros generates all of the macros and both the server-alias and library-alias tables.

The following log displays the information in the SAS log about the server macros.

***Log 6.1*** *Server Macro Information*

```
LOG
 Command ===>
1085  %shrmacs (server,msg);

            *** SAS/SHARE macros are now available ***

For further information about SAS/SHARE macros:

   %SHRMACS(ALL,HELP)     - for information on all macros
   %SHRMACS(USER,HELP)    - for information on macros used
                            in user applications
   %SHRMACS(OPER,HELP)    - for information on macros used
                            with PROC OPERATE
   %SHRMACS(SERVER,HELP) - for information on macros used
                            with PROC SERVER
   or %macro(HELP)        - for information on a specific
                            macro
SAS/SHARE macros generated are:

SERVERID - translate server alias
LISTLIB  - list library table
LISTSRV  - list server alias table
STRTSRV  - start a server

                        SERVER ALIAS TABLE

--- SERVER ALIAS ------- SERVERID ----------------------------
    CESERV               V6DSERVR
    COMSERV              MYSERV
    DEVSERV              MYSERV
    GLOSSERV             V6DSERVR
    LIBSERV              V6DSERVR
    MISSERV              MYSERV
    PRDSERV              V6DSERVR
-------------------------------------------------------------
```

For more information about the macros that are compiled by %SHRMACS, you can specify the HELP keyword. For example:

```
%shrmacs(server,help);
```

HELP lists the syntax, a brief description, and an example for each macro that SHRMACS defines in the SERVER category.

# The APPLSYS Macro Library

## *Overview of the APPLSYS Macro Library*

The data that is used by the macros is stored in tables in another macro library that you maintain as you add and delete libraries, servers, and application systems. This library is called the APPLSYS (an acronym for application system) macro library.

The tasks to customize macros are explained in the following sections.

See Chapter 14, “SAS/SHARE Macros,” on page 165 for complete information about the syntax of the SAS/SHARE autocall macro library and how to use it.

### Specifying the APPLSYS Macro Library

The default name of the APPLSYS macro library depends on the operating environment. Here are the default macro library names:

*z/OS Specifics*
**SAS.SASSAML**

*UNIX Specifics*
***!sasroot*/saspgm/sassaml**

*Windows Specifics*
***!SASROOT*\SHARE\SASMACRO**

Files in the APPLSYS macro library are called members, and they must have a .SAS extension.

To use the default library table, do not include the APPLSYS= argument in the SHRMACS macro. For example,

```
%shrmacs(user);
```

To specify an alternate library-alias table, include the APPLSYS= argument in the SHRMACS macro. For example,

```
%shrmacs(user, applsys=purchas);
```

It might be more convenient to allow different users or departments to maintain their own APPLSYS macro libraries. To use an alternate APPLSYS macro library instead of the default library, specify the SASSAML= argument in the SHRMACS macro.

The value for the SASSAML argument can be the operating environment-specific physical name of the alternate library or the string _DEFINED_, which indicates that the fileref SASSAML has already been assigned to the alternate library. For example:

```
%shrmacs(user,applsys=purchas,sassaml=library-path);

%shrmacs(user,applsys=_DEFINED_);
```

Here are some operating environment-specific examples to specify an alternate APPLSYS macro library:

*UNIX Specifics*
**%shrmacs(user,applsys,sassaml=/dept/mis/applsys);**

*Windows Specifics*
**%shrmacs(user,applsys,sassaml=c:\dept\mis\applsys);**

### Defining Server Aliases (SERVID)

Server aliases can help you do the following tasks:

- use your existing SAS programs with new releases of SAS/SHARE without having to change them. This is accomplished by using the name of the existing SAS/SHARE server as an alias for the name of a SAS/SHARE server that is executing the new release of SAS/SHARE.
- shift server traffic easily. When you begin using SAS/SHARE, you might create many aliases for a single server, with each alias used by only one or a small number of applications. As server use increases, you'll want to add a second server and move some of your applications to that server. You can do this by changing the entry in the

SERVERID member to point to the new server for the applications that you want to move.

To define server aliases, create a member named SERVERID in the APPLSYS macro library. The member name must be SERVERID because the SHRMACS macro looks for that specific name.

Define a server alias in the SERVERID member by using the following syntax:

%SERVID(*alias,server-ID*);

The following code shows an example of a SERVERID member.

***Example Code 6.1*** *Server-Alias Table*

```
/************************************************/
/*                                              */
/*  NAME: SERVERID                              */
/*                                              */
/*  SERVER ALIAS TABLE ENTRIES                  */
/*                                              */
/*  This member defines aliases for server names.  */
/*  The entries in this member are loaded into     */
/*  the server-alias table by the SHRMACS macro.   */
/*  This server-alias table is used by the         */
/*  SERVERID macro to translate an alias to an     */
/*  actual server ID.                               */
/*                                              */
/*  To add aliases to the table, specify each      */
/*  alias and its real server ID in a SERVID call  */
/*  at the end of this member.                  */
/*                                              */
/************************************************/
%servid(shr7,shrserv7)
%servid(share1,shrserv3)
%servid(pubserv,shrserv7)
```

## *Associating SAS Libraries with Server Aliases (SERVLIB)*

Create a member in the APPLSYS macro library for each application system that is specified by the APPLSYS= argument in the SHRMACS macro. Doing this defines library-server pairs that a specific application will probably use. For example, you can specify an APPLSYS library named PURCH in the SHRMACS macro as follows:

```
%SHRMACS(user,APPLSYS=purch);
```

If you do this, you also create a member in the APPLSYS library of the same name (in this example, PURCH).

In a selected member, use the following syntax to specify the library and server name pairs.

%SERVLIB(*SAS-library, server-name*);

*SAS-library* is specific to the operating environment. *server-name* can be a server ID or its alias.

The following code contains a member named PURCH, which references operating environment-specific SAS library names.

***Example Code 6.2*** *Library-Alias Table*

```
/*********************************************************************/
/*                                                                   */
/*  NAME: PURCH                                                      */
/*                                                                   */
/*  LIBRARY TABLE ENTRIES - SPECIFIC APPLICATION                     */
/*                                                                   */
/*  This member associates server names with libraries. The entries   */
/*     in this member are loaded into the library table if the        */
/*     SHRMACS macro is called by using the argument APPLSYS=APPLSAMP.*/
/*     If APPLSYS=APPLSAMP is specified, the entries can also be      */
/*     loaded by using a call to the LIBDEF.                          */
/*                                                                   */
/*  To add libraries to the definition table, add a SERVLIB call      */
/*    for each library at the end of this member. Specify the         */
/*    physical name for the library and the name of the server to be  */
/*    associated with the library. The name can be an alias or an     */
/*    actual server ID.                                               */
/*                                                                   */
/*********************************************************************/

%servlib(shrtest.appljan.lib1, testserv);                # z/OS
%servlib(/shrtest/appljan/lib1, testserv);               # UNIX
%servlib(d:\shrtest\appljan\lib1, testserv);             # Windows
```

To add aliases to the table, use a SERVLIB call for each library-server pair.

Additionally, create a member in the APPLSYS macro library named DEFAULTS. This member can be empty but must be created to avoid having error messages generated from the SHRMACS macro.

The DEFAULTS member is used when the APPLSYS= argument is omitted in a call to the SHRMACS and LIBDEF macros. The syntax of the PURCH and the DEFAULTS members is identical.

## *Creating the Server Information Table (SERVINFO)*

A server information table is created to store information about the servers at your site. You can use this information in a program, or you can display it. By default, the table contains the following information:

- a default value for the RMTVIEW= option in the REMOTE engine's LIBNAME statement
- a network node name that is represented in the two-level server name format: *node.server-id*.

You can use the server information table to store other attributes of the server, its users, or its administrators, such as server access passwords, PROC SERVER statement options, and the SAS release that the server runs under.

To add an entry to the server information table, use the SERVINFO macro as follows:

%SERVINFO (*node.server-id*, netnode=*fully-qualified-node-name*, RMTVIEW=NO);

Usually, the SERVINFO macro is used to cause the SERVERID macro to generate an alias for a node name that is not a valid SAS name. For example, you can specify the two-level server name and the netnode in the server information table as follows:

```
%servinfo (hp.shrserv,netnode=hp103.dom2.acme.com);
```

When you do this, the server SHRSERV runs on HP103.DOM2.ACME.COM.

In resolving an alias for HP.SHRSERV, the SERVERID macro generates the following:

```
%let hp=hp103.dom2.acme.com;
```

### *Customizing a Server Information Table*

The following example shows how to customize your server information table to include the SAS software release number for the SAS/SHARE server. The tasks in this example do the following:

- account for a new parameter in the SERVINFO macro
- reformat the table's appearance
- add the new information to the table
- view the server information table with the LISTSRVI macro

The following eight steps show how to alter the server information table for display only.

1. Add the new parameter SASREL= to the SERVINFO macro statement.

   ```
   %macro servinfo(servid,version=,rmtview=,netnode=,sasrel=);
   ```

2. Add the new variable ISREL&SRVINUM to the %GLOBAL statement to account for the new parameter SASREL.

   ```
   %GLOBAL isrvr&srvinum irmtv&srvinum inode&srvinum isrel&srvinum;
   ```

3. Because the table is implemented as sets of macro variables, assign the value &SASREL to the macro variable ISREL&SRVINUM.

   ```
   %LET isrel&srvinum = &sasrel;
   ```

4. In the LISTSRVI macro, modify the line that prints the headers for the table.

   ```
   %put &pline RMTVIEW %shrrpt(-,3)
      NETWORK NODE %shrrpt(-,20) RELEASE %shrrpt(-,3);
   ```

5. You might want to change the %PUT statement in the LISTSRVI macro to extend the dashed line following the table to match the length of the modified header line.

   ```
   %put %shrrpt(-,78);
   ```

6. Change the loop that prints the table so that it looks like this:

   ```
   %do i=1 %to &srvinum;
     %let pline=%shrrpt(&blank,3)
        %shrfmt(&&isrvr&i,16);
     %let pline=&pline
        %shrfmt(&&irmtv&i,11)
        %shrfmt(&&inode&i,36);
     %let pline=&pline &&isrel&i;
     %put &pline;
   %end;
   ```

7. If you want to be able to access and use the information in the table for a macro or a program, you would have to use the SERVIIDX macro.

   ```
   %let i=%serviidx(&new_id);
   %if (&&isrel&i^=) %then
   ```

```
%do;
 /* some use of &&isrel&i here */
%end;
```

8. After you have accounted for the new parameter and modified the format of the server information table, you can add entries to the table for all parameters.

```
%servinfo(rmthost.share1,netnode=.acme.com,
  rmtview=no,sasrel=6.12);
%servinfo(rmthost.share2,netnode=smith.acme.com,
  rmtview=yes,sasrel=7);
```

When you invoke the LISTSRVI macro, the server information table is displayed in the SAS log.

```
%listsrvi;
```

```
              SERVER INFORMATION TABLE
--- SERVERID --- RMTVIEW -- NETWORK NODE --- RELEASE
RMTHOST.SHARE1   NO        rmthost.acme.com   6.12
RMTHOST.SHARE2   YES       smith.acme.com     7
-------------------------------------------------
```

## Generating a LIBNAME Statement By Using the LIBDEF Macro

If your SAS application accesses a server library, use the LIBDEF macro instead of a LIBNAME statement. Use the syntax that follows.

*Note:* Before you invoke the LIBDEF macro, you must first invoke the SHRMACS macro.

**%LIBDEF**(*libref,SAS-library-name*) <,APPLSYS=*app-sys-lib-tab*>;

***CAUTION:***
**Do not enclose the SAS library name in quotation marks. Using quotation marks will cause the generation of the LIBNAME statement to fail.**

The LIBDEF macro generates a LIBNAME statement by searching the library table for the library name. Then it invokes the SERVERID macro to convert the server alias into a server ID. Here are some operating environment-specific examples for using the LIBDEF macro.

*z/OS Specifics*

```
%libdef(mylib,shrtest.appljan.lib1);
```

*UNIX Specifics*

```
%libdef(mylib/shrtest/appljan/lib1);
```

*Windows Specifics*

```
%libdef(mylib,d:\shrtest\appljan\lib1);
```

Each macro invocation defines the library to the libref MYLIB. You can define additional libraries without invoking the SHRMACS macro again.

If you want to use server aliases but you have not created the library- and server-name pairs in the APPLSYS macro library yet, you can use a LIBNAME statement and invoke the SERVERID macro in place of the SERVER= argument. For more information, see Chapter 14, “SAS/SHARE Macros,” on page 165.

## Using APPLSYS= to Call the SHRMACS and LIBDEF Macros

You can specify the APPLSYS= argument in either the SHRMACS or the LIBDEF macro. The following example contains application excerpts that show how to access libraries from three applications systems (PURCH, MAINT, and FACIL) by using the APPLSYS= argument.

*Operating Environment Information*
For *SAS-library*, use the syntax convention that is appropriate for your operating environment.

***Example Code 6.3*** *Using the APPLSYS= Argument to Call the SHRMACS and LIBDEF Macros*

```
/* Most libraries will come from purchasing appl sys.*/
%shrmacs(user,nomsg,applsys=purch);

/* Access purchase order library.*/
%libdef(polib,SAS-data-library1);
            .
            .
            .
            .
/* Access vendor service library from maintenance appl sys.*/
%libdef(vndsvc,SAS-data-library2,applsys=maint);
            .
            .
            .
            .
/* Access vendor account library from purchasing appl sys. */
/* (Note: It is not necessary to specify APPLSYS= for this */
/* application system because it was specified above.)     */
%libdef(vndacct,SAS-data-library3);

/* Access vendor contact library from maintenance appl sys.*/
/* (Note: It is not necessary to specify APPLSYS= for this */
/* application system because it was specified above.)     */
%libdef(vndcon,SAS-data-library4);
            .
            .
            .
            .
/* Access inventory library from facilities appl sys.      */
%libdef(invlib,SAS-data-library5,applsys=facil);

/* Access invoice library from purchasing appl sys.        */
%libdef(invoice,SAS-data-library6);
```

*Chapter 7*

# Interpreting SAS/SHARE Server Log Messages



## The SAS/SHARE Server Log

The server log displays messages that result from starting and stopping a server and from intervening client/server transactions. This section gives an example of a server log for multiple users and interprets some of the common messages that are seen in the server log.

To make the raw data in the server log messages meaningful, you can use a set of server log analysis programs to examine specific data resources and to create usable reports. For more information, see "Analyzing the Server Log" on page 97.

Server administrator logs record messages that result from using the OPERATE procedure. For more information, see "Managing a Server, Its Libraries, and Its Users" on page 39.

# Starting the Server Log

Use the PROC SERVER statement to explicitly start the server logging with the specific features that you want. To prepare for server log analysis, set the message numbering feature (MSGNUMBER).

Message numbering assigns a number to each message that is recorded in the log. The server log analysis programs parse messages by using the associated numbers.

Here is an example of specifying message numbering when the server is started:

```
proc server msgnumber id=demoserv;
```

For more information, see Chapter 9, "The SERVER Procedure," on page 105.

# Usage Statistics in the Server Log

## *Sample Log for SAS/SHARE Server SHARE2*

The following server log shows that users JOHN(1), MARIA(2), and JOHN(3) started and closed three separate server sessions. For details about the types of actions that were performed by the users, which created specific messages in the log, see "Reading the Server Log" on page 93. When each of the three sessions was closed, usage statistics were generated. In addition, cumulative usage statistics were generated for the server SHARE2. The usage statistics are controlled by values that you provide for the LOG option in the PROC SERVER statement. For explanations of usage statistics for messages processed (MESSAGE), bytes transferred (BYTECOUNT), active time (ACTIVETIME), and elapsed time (ELAPSEDTIME), see Chapter 9, "The SERVER Procedure," on page 105.

The following log shows a typical server log with all logging statistics shown for the server SHARE2 that is running under a UNIX operating environment.

***Log 7.1*** *Sample Log for SAS/SHARE Server SHARE2*

```
Command ===>
1?  PROC SERVER ID=share2 LOG=(ACTIVETIME BYTECOUNT ELAPSEDTIME MESSAGE)
msgnumber;
2?  run;
30Apr2008:07:15:36.690 043131 SAS server SHARE2 started.
30Apr2008:07:16:20.048 043021 User john(1) has connected to server SHARE2.
30Apr2008:07:16:20.442 043143 User john(1) has created "Line Mode Process"(1)
                              under "Kernel"(0).
30Apr2008:07:16:21.206 043069 Server library TESTDATA
                              ('/local/u/john/server' V9) accessed as
                              TESTDATA by user john(1).
30Apr2008:07:16:31.593 043021 User maria(2) has connected to server SHARE2.
30Apr2008:07:16:31.846 043143 User maria(2) has created "Line Mode Process"(1)
                              under "Kernel"(0).
30Apr2008:07:16:31.923 043069 Server library DEMOTEST
                              ('/local/u/john/server' V9) accessed as
                              DEMOTEST by user maria(2).
30Apr2008:07:17:32.462 043143 User maria(2) has created "PRINT"(2) under
                              "Line Mode Process"(1).
30Apr2008:07:17:33.537 043100 DEMOTEST.X.DATA(1) opened for input/S via
                              engine V9 by "PRINT"(2) of user maria(2).
30Apr2008:07:17:40.361 043102 DEMOTEST.X.DATA(1) closed by "PRINT"(2)
                              of user maria(2).
30Apr2008:07:17:40.422 043144 User maria(2) has terminated "PRINT"(2)
                              (under "Line Mode Process"(1)).
30Apr2008:07:18:05.575 043143 User maria(2) has created "DATASTEP"(3)
                              under "Line Mode Process"(1).
30Apr2008:07:18:05.668 043100 DEMOTEST.DEMO.DATA(1) opened for output via
                              engine V9 by "DATASTEP"(3) of user
maria(2).
```

```
30Apr2008:07:18:06.016 043102 DEMOTEST.DEMO.DATA(1) closed by "DATASTEP"(3)
                              of user maria(2).
30Apr2008:07:18:06.096 043144 User maria(2) has terminated "DATASTEP"(3)
                              (under "Line Mode Process"(1)).
30Apr2008:07:18:48.262 043143 User john(1) has created "PRINT"(2)
                              under "Line Mode Process"(1).
30Apr2008:07:18:48.313 043100 TESTDATA.DEMO.DATA(1) opened for input/S via
                              engine V9 by "PRINT"(2) of user john(1).
30Apr2008:07:18:49.734 043102 TESTDATA.DEMO.DATA(1) closed by "PRINT"(2)
                              of user john(1).
30Apr2008:07:18:49.765 043144 User john(1) has terminated "PRINT"(2)
                              (under "Line Mode Process"(1)).
30Apr2008:07:18:58.322 04306A Server library DEMOTEST (accessed as DEMOTEST)
                              released by user maria(2).
30Apr2008:07:18:58.338 043144 User maria(2) has terminated "Line Mode
                              Process"(1) (under "Kernel"(0)).
30Apr2008:07:18:58.909 043022 User maria(2) has disconnected from server SHARE2.
30Apr2008:07:18:59.886 043151 Usage statistics for user maria(2):
                                  Messages processed:                  24
                                  Bytes transferred:                8,028
                                  Active time:               0:00:02.7525
                                  Elapsed time:              0:02:28.3527
30Apr2008:07:19:06.298 04306A Server library TESTDATA (accessed as TESTDATA)
                              released by user john(1).
30Apr2008:07:19:06.319 043144 User john(1) has terminated "Line Mode
                              Process"(1) (under "Kernel"(0)).
30Apr2008:07:19:06.411 043022 User john(1) has disconnected from server SHARE2.
30Apr2008:07:19:06.425 043151 Usage statistics for user john(1):
                                  Messages processed:                  14
                                  Bytes transferred:                3,133
                                  Active time:               0:00:01.8139
                                  Elapsed time:              0:02:46.6840
30Apr2008:07:19:16.018 043021 User john(3) has connected to server SHARE2.
30Apr2008:07:19:16.569 0430A9 PROC OPERATE command from user john(3): STOP
SERVER;
30Apr2008:07:19:16.603 043132 Normal termination of SAS server SHARE2 has
occurred.
30Apr2008:07:19:17.212 043022 User john(3) has disconnected from server SHARE2.
30Apr2008:07:19:17.246 043151 Usage statistics for user john(3):
                                  Messages processed:                   2
                                  Bytes transferred:                  102
                                  Active time:               0:00:01.0691
                                  Elapsed time:              0:00:01.9230
30Apr2008:07:19:17.642 043150 Usage statistics for server SHARE2:
                                  Messages processed:                  40
                                  Bytes transferred:               11,263
                                  Active time:               0:00:05.6355
                                  Elapsed time:              0:03:42.8948
NOTE: PROCEDURE SERVER used:
      real time           3:45.51
      cpu time            0.74 seconds
```

## Format for Server Log Messages

In a server log, a message is posted for each significant client/server transaction. A log message is presented in the following form:

*dtformat msgnumber message*

The dtformat and msgnumber fields are controlled by options that you provide in the PROC SERVER statement. For explanations of these options, see Chapter 9, "The SERVER Procedure," on page 105.

# Server Log Message Components

Server log messages consist of the following components, which are repeated throughout the log:

*engine-name*
is the name of the engine that will process the SAS library in the server's SAS execution.

*libref*
is the name temporarily associated with a SAS library. You assign a libref by using a LIBNAME statement or operating system control language.

*libref.member-name.member-type (open sequence number)*
*libref*
is the first part of a multi-level SAS filename that is temporarily associated with the SAS library in which the file is stored.

*member name*
is the filename in a SAS library that references an access descriptor, or a stored program.

*member type*
is the name assigned by SAS that identifies the type of information that is stored in a SAS file (for example ACCESS, DATA, CATALOG, PROGRAM, or VIEW).

*open sequence number*
in parentheses, is a counter that is used for tracking.

*open mode / access pattern*
The following table shows the types of open mode and their functions.

***Table 7.1*** *Open Modes*

| Open Mode | Function |
|---|---|
| Input | Opens files to read. |
| Output | Creates or replaces files. (Do not use a slash when specifying the Output mode.) |
| Update | Modifies existing observations or adds new observations, or both. |
| Utility | Modifies the header data (for example, assigning a new label or format to a variable). |

The following table shows the types of access pattern and their functions.

***Table 7.2*** *Access Patterns*

| Access Pattern | Function |
|---|---|
| Random (R) | Processes observations according to the value of an indicator variable without processing preceding observations. |
| Sequential (S) | Processes observations one after the other, starting at the beginning of the file and continuing to the end of the file. |
| Two-pass (2) | Enables a SAS procedure to pass through the data more than one time. |
| BY-group rewind (B) | Enables a SAS procedure to pass through the data more than one time. |
| Contents type (C) | Reads header data, such as names of variables, but does not read observation data, such as data that PROC PRINT reads. |

The two-pass and BY-group rewind access patterns both enable SAS procedures to pass through the data more than one time. For example, during the initial pass a sum or count is computed; during the second pass, the values of the variables in each observation are compared to, added to, or subtracted from the value that was computed in the first pass. The distinction between these two access patterns is subtle.

When a SAS data set contains only one BY group, there is no difference. When a SAS data set contains multiple BY groups, rewinding a BY group after the first BY group is processed requires the ability to position to a random location in the file, which is not complicated when using disk devices but is almost impossible when using tape devices. (The I/O supervisor is able to remember the starting position of the current BY group.)

Only the two-pass access patterns require the ability to rewind the entire SAS data set.

*resource environment (resource environment number)*
is a structure that is used within SAS to scope and manage the usage of system resources. Examples of resource environments include SAS procedures, SAS windows, DATA steps, or other internal SAS activity.

The resource environment number, in parentheses, is a counter that starts at 1 for each connection. To precisely identify a resource environment in a server's log, you need the connection number and the resource environment number.

*serverid*
specifies a name for the server. The server name must meet the criteria for a valid SAS name, which can include the following special characters: dollar sign ($), at sign (@), and pound sign (#). For more information about the rules for naming SAS variables, see *SAS Language Reference: Concepts*.

Naming a server must also include criteria that are imposed by the operating environment and the access method that you specify for communication between a server and a client session. For example, if you are using the TCP/IP communications access method, the serverid that is specified must be a valid TCP/IP service as defined in the TCP/IP SERVICES file.

For information about naming servers by operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

*userid(connection number)*
specifies a valid user ID for the accessing client on the server. The operating environment on which the client runs can affect user naming conventions. For details about specifying valid user IDs, see Details on page 131.

The connection number is shown in parentheses after the user ID. This number begins at 1 and increases by 1 each time a user connects to the server. The server maintains this counter. The connection number is shown in most of the messages that are recording activity for each connection. If the same user ID is connected to a server more than one time, it is possible to track the activity of each connection separately.

# Reading the Server Log

## The Start Message

```
30Apr2008:07:15:36.690 043131 SAS server SHARE2 started.
```

A message such as this always appears in the log when the server starts. The message tells the server administrator that the server initialization completed successfully. In this example, **`server SHARE2`** identifies which server was started. After this message is printed, the server waits for clients to connect to it. For information about starting a server, see "Starting a Server" on page 33. For information about server IDs, see "Server Log Message Components" on page 91.

## The Connect Message

```
30Apr2008:07:16:20.048 043021 User john(1) has connected to server SHARE2.
```

A message such as this appears when a client establishes a communication path with the server. The message contains a user ID, a connection number, and a server ID that tell which user has connected to which server. In this example, user **`john(1)`** has connected to server **`SHARE2`**. For more information about user IDs and server IDs, see "Server Log Message Components" on page 91. The Connect message is bracketed in the log by the disconnect message. See "The Disconnect Message" on page 96.

## The Create Message

```
30Apr2008:07:16:20.442 043143 User john(1) has created "Line Mode Process"(1)
under "Kernel"(0).
```

A message such as this shows that the user has created a resource environment. During a connection, the user, in this example **`john(1)`**, creates mirror resource environments, which are used to maintain (track, scope) the resources that the user consumes in the server's session (or that the server consumes on the user's behalf). The number of each newly created resource environment will appear in later messages about the resources that were consumed.

The quoted string, **`"Line Mode Process"`**, is the name of the resource environment that was created. The number in parenthesis is the resource environment number.

The quoted string **`"Kernel"`** is the name of another resource environment, and the number in parenthesis is the resource environment number. **`"Kernel"`** is the parent resource environment of the created resource environment that this message records. In

this example, the **"Kernel"** resource environment is the parent of the **"Line Mode Process"** resource environment. It is possible to deduce a resource environment tree that is in effect during a server's session.

The **"Kernel" (0)** resource environment is automatically created when a user connects to a server. The primary purpose for kernel resource environments is to create child resource environments.

For more information, see "Server Log Message Components" on page 91.

The Create message is bracketed in the log by the Terminate message.

For more information, see "The Terminate Message" on page 95.

### The Access Message

```
30Apr2008:07:16:21.206 043069 Server library TESTDATA
('/local/u/john/server' V9) accessed as TESTDATA by user john(1).
```

A message such as this appears when a SAS client executes a LIBNAME statement, or an external client (that is, a client that is not a SAS client) performs an action that associates a libref with a SAS library. This message gives you an association between the libref that you'll see in subsequent messages and the physical name, engine, and server's libref of that library. Under z/OS, accessing a library results in a physical file being opened; in other operating environments, accessing a library is similar to changing directories.

The Access message begins with the libref that the server uses to reference the library. In this example, the libref is **TESTDATA**. The information in parentheses **('/local/u/john/server' V9)** is the physical name of the library (the quoted string), and the name of the engine (V9) that is used in the server's session to access the library.

The end of the message identifies the user who is accessing the library. In this example, the user is **john**, and the number in parenthesis **(1)** is the connection number, which is used for tracking a user's activity on the connection.

For more information about librefs, engine names, and user IDs, see "Server Log Message Components" on page 91. For information about accessing a SAS library, see "Defining a SAS Library to a Server (All New Users)" on page 6.

The Access message is bracketed in the log by the Release message. See "The Release Message" on page 95.

### The Open Message

```
30Apr2008:07:17:33.537 043100 DEMOTEST.X.DATA(1) opened for input/S via
engine V9 by "PRINT"(2) of user maria(2).
```

A message such as this appears when a user opens a SAS data file.

Opening a SAS file allows the user to move around within the file, see which variables are in the file, check the file size and the file formats, and so on.

Opening a SAS data view is more complicated than opening a data file. To open a data view, the server gets help from the engine supervisor to do the following:

- load the engine that will interpret the view
- open the view file
- have the view engine open the underlying files and prepare to interpret the instructions in the view file.

The Open message begins with the **libref.member-name.member-type**. In this example, **DEMOTEST.X.DATA** shows which file or view is being opened, and the number in parentheses **(1)** is the open sequence number. This number is a counter to track how many times a data set is opened within a resource environment.

**Input** is the type of Open mode, and **S** is the access pattern for this open.

**engine V9** identifies which engine is being used to interpret the view for the member type VIEW.

**"PRINT" (2)** identifies the resource environment that is being used to open the file or view. Openings are considered resources, so resource environments track them. The same resource environments also track the resources, such as memory, that are consumed when a file or view is opened.

**maria** is the user ID, which shows who opened the file or view, and **(2)** is the number of times that the file was opened. For more information about librefs, member names, member types, open modes, access patterns, engine names, resource environments, and user IDs, see "Server Log Message Components" on page 91.

The Open message is bracketed in the log by the Close message. See "The Close Message" on page 95.

### *The Close Message*

```
30Apr2008:07:17:40.361 043102 DEMOTEST.X.DATA(1) closed by "PRINT"(2)
of user maria(2).
```

A message such as this appears when a user closes a file. In this example, user **maria** has closed the file **DEMOTEST.X.DATA** under the resource environment **"PRINT"**. For more information about librefs, member names, member types, resource environments, and user IDs, see "Server Log Message Components" on page 91. See also "The Open Message" on page 94.

### *The Release Message*

```
30Apr2008:07:19:06.298 04306A Server library TESTDATA (accessed as TESTDATA)
released by user john(1).
```

A message such as this appears only if the user explicitly executes a LIBNAME statement that includes the CLEAR option. For example:

```
libname TESTDATA clear;
```

In this example, user **john** has released the library **TESTDATA**. The Release message is the opposite of the Access message. See "The Access Message" on page 94.

For more information about user IDs and librefs, see "Server Log Message Components" on page 91.

### *The Terminate Message*

```
30Apr2008:07:19:06.319 043144 User john(1) has terminated "Line Mode
Process"(1) (under "Kernel"(0)).
```

A message such as this appears when a user terminates a resource environment which was previously created. See "The Create Message" on page 93. In this example, user **john** terminated the resource environment **"Line Mode Process" (1)**, which was a child resource environment of **"Kernel" (0)**. For more information about resource environments and user IDs, see "Server Log Message Components" on page 91.

### The Disconnect Message

```
30Apr2008:07:19:06.411 043022 User john(1) has disconnected from server SHARE2.
```

A message such as this appears when a user who was connected to the server has ended that connection. In this example, user **john** has disconnected from server **SHARE2**. For more information about user IDs and server IDs, see "Server Log Message Components" on page 91.

Usually, the Close message appears before the Disconnect message. However, a Disconnect message might appear before a Close message appears if the client's session with the server was ended abnormally (for example, if the line is disconnected or a client's machine crashes).

### Accounting Information

```
30Apr2008:07:19:06.425 043151 Usage statistics for user john(1):
                                  Messages processed:                  14
                                  Bytes transferred:                3,133
                                  Active time:                0:00:01.8139
                                  Elapsed time:               0:02:46.6840
```

By default, accounting data is written to the server's log when a user disconnects from the server. (See "The Disconnect Message" on page 96 .) This data shows the usage of server resources—that is, the messages processed, bytes transferred, active time, and elapsed time—that resulted from a user's activity while connected to the server. In this example, the data shown is for user**john** while connected to server **SHARE2**.

For explanations of usage statistics for messages processed (MESSAGE), bytes transferred (BYTECOUNT), active time (ACTIVETIME), and elapsed time (ELAPSEDTIME), see Chapter 9, "The SERVER Procedure," on page 105. For more information about accounting level, see Chapter 9, "The SERVER Procedure," on page 105. For more information about user IDs, see "Server Log Message Components" on page 91.

### The Stop Message

The Stop message appears in the log if an administrator uses the STOP command under PROC OPERATE. The server stops all activity and shuts down; all processes end normally. The server log shows that the STOP command was issued, and shows the Close, Release, Terminate, and Disconnect messages that follow as the files, libraries, resource environments, and the connection to the server are shut down. For more information about the server administrator's log and an example of what the STOP command generates in the server administrator's log, see "Managing a Server, Its Libraries, and Its Users" on page 39.

*Chapter 8*

# Analyzing the Server Log



## Starting the Server Log

The information in this section is recommended for SAS/SHARE server administrators who write SAS programs.

The server log records messages that result from starting and stopping a server and from many intervening client/server transactions. The PROC SERVER statement is used to explicitly start server logging with the specific features that you want. For more information about the PROC SERVER statement, see Chapter 9, “The SERVER Procedure,” on page 105. Here is the syntax for the PROC SERVER statement with two options that you can use to start logging:

**PROC SERVER** MSGNUMBER ID=DEMOSERV;

To make the log's raw data meaningful, you can use a set of server log analysis programs to examine specific data resources and to create usable reports. Analysis of the logged data resources of several SAS/SHARE servers allows you to compare server performance and to balance workloads among them.

To prepare for server log analysis, set the message numbering feature. Message numbering assigns a number to each message that is recorded in the log. The server log analysis programs parse messages by using the associated numbers.

## Using the Server Log Analysis Tools

SAS provides a set of sample programs that you can use as a basis for developing your own programs to analyze server log data. The location of the sample programs varies according to the operating environment.

*UNIX Specifics*
**!SASROOT/samples/share**

*Windows Specifics*
**!SASROOT\Share\Sample**

*z/OS Specifics*
**&prefix.SAMPLE**

Here are descriptions of the sample programs:

SLTOOLM.SAS
is a look-up table that associates a macro variable with each message number that is generated in the server log. For example, the DINIMSG macro variable is assigned to the 43131 message number, which corresponds to a PROC SERVER start-up.

SLTOOL0.SAS
is a driver program that automates the execution of all the other programs.

SLTOOL1.SAS
converts the server's log from a file into a SAS data set.

SLTOOL2.SAS
creates a set of SAS data files from the SAS data set that is generated by SLTOOL1.SAS and stores the files in the library SLOGDATA. Each of the created files relates to specific data that is collected in the log. For example, SLOGDATA.SERVINFO is a data file that records the server's name and the times when it was started and stopped.

SLTOOL3.SAS
is a sample program.

SLTOOL4.SAS
is another sample program.

Before you use these programs, you must customize them to your site's operating environment and your log analysis needs. Running an untuned program produces unpredictable results.

## Customizing Server Log Analysis Programs

SLTOOL0.SAS is a sample driver program that you can use to automate the execution of your set of programs at the same time. Alternatively, you can run them individually in consecutive order. For example:

```
FILENAME INLOG 'TESTLOG.LOG';
LIBNAME  SLOGDATA 'SLOGDATA';
%INCLUDE(SLTOOLM);
%INCLUDE(SLTOOL1);
%INCLUDE(SLTOOL2);
%INCLUDE(SLTOOL3);
%INCLUDE(SLTOOL4);
```

The first line associates the fileref INLOG with an operating environment-specific name for a file that contains the server log. In this example, 'TESTLOG.LOG' is the name of the file in a UNIX operating environment that contains the server log. In the second line, the LIBNAME statement associates the libref SLOGDATA with the operating environment-specific SAS library 'SLOGDATA'. The first two lines identify the external file as input and specify a SAS library to write SAS data files to.

The remaining lines in the driver program are INCLUDE macro statements in SAS, which read and execute each named program consecutively.

Modify the FILENAME and LIBNAME statements in SLTOOL0.SAS to specify your server's log and the repository for the SAS data files, respectively.

SLTOOLM.SAS, SLTOOL1.SAS, and SLTOOL2.SAS do not require modification. They can be run as provided to produce SAS data files that contain information about the server's session. Those data files are the input to the analysis phase, which is performed by the programs in SLTOOL3.SAS and SLTOOL4.SAS.

Usually, you customize the SLTOOL3.SAS and SLTOOL4.SAS programs to produce the analyses of your server that are most relevant to your needs.

## Executing the Driver Program (SAS/SHARE)

How you execute the driver program depends on your operating environment. The following example runs the driver program on a UNIX operating environment:

```
sas pathname/sltool0.sas -log
/pathname/logfile
```

**sas** invokes the SAS System. *pathname* specifies the location of SLTOOL0 and the location of the log file.

## SLTOOL1 Sample Program (SAS/SHARE)

Before you use the SLTOOL1 program, the server log must be in a file that can be read by the INFILE and the INPUT statements in the DATA step. The FILENAME statement points to the physical location that contains the server's log.

Because SLTOOL1 produces a compressed SAS data file that contains the server log and some additional data, it is recommended that you reserve an amount of space that is twice the size of the server log file.

SLTOOL1 produces a data file that is stored in the library WORK. Because the library WORK is temporary, it exists only for the duration of the SAS session and is deleted at termination. To keep the data file that is produced by SLTOOL1 for use after the session terminates, save it in a permanent library by specifying a valid, two-level name. For example:

```
DATA SLOGDATA.CVTLOG (DROP=SERVREL ANALREL
                      COMPRESS=YES
                      LABEL='Server Log');
```

# SLTOOL2 Sample Program (SAS/SHARE)

## *Overview of the SLTOOL2 Sample Program*

SLTOOL2 reads the SAS data file produced by SLTOOL1 and creates a group of SAS data sets. Usually, SLTOOL2 is executed during the same SAS session as the SLTOOL1 program.

The libref SLOGDATA is associated with a SAS library by the LIBNAME statement in SLTOOL0.SAS.

The data sets created by SLTOOL2 are stored in the SAS library and are associated with the libref SLOGDATA. It is most efficient to run SLTOOL0.SAS one time to create the data files in the library SLOGDATA, and then run multiple analysis programs that access the library SLOGDATA. With large server logs, creating the data sets in the library SLOGDATA can take quite a long time and should be done only one time for each server log.

SLTOOL2 creates data sets with names that include either INFO or SUM. INFO files contain observations that record specific SAS/SHARE activities, such as each time a server is started and stopped. SUM files present the total number of times a specific activity occurred, such as the total number of connections made to a server.

The following sections describe each data set that is created by SLTOOL2.

### *SLOGDATA.SERVINFO*

The SERVINFO data set records the server name and the times at which it was started and stopped. You can use this information to write a descriptive header on a report that relates to that server.

### *SLOGDATA.CONNINFO*

The CONNINFO data set contains one observation for each time a user connects to the server and one observation for each time a user disconnects from the server.

From this data set, you can obtain a list that shows who connected to a server, how long each user remained connected, or how many times each user connected to the server. You can also chart the simultaneous number of connections to a server over a period of time, which shows peaks and valleys in the number of users who access data through a specific server.

### SLOGINFO.CONNSUM

The CONNSUM data set contains only one observation with one variable that stores the total number of connections to this server.

### SLOGDATA.TASKINFO

The TASKINFO data set contains one observation for each creation of a mirror resource environment and one observation for each termination of a mirror resource environment.

The name of a resource environment in this data set corresponds to the name of a SAS procedure or a window that is used to access data through the server. From this data set, you can obtain a list of those SAS procedures and windows and the length of time each procedure or each window remained active.

### SLOGDATA.LIBINFO

The LIBINFO data set contains one observation for each time a user accesses a SAS library and one observation for each time a user releases a SAS library.

From this file, you can determine how many times each library was accessed through the server and the length of time that each library was accessed. You should use the physical name for the library because each library can be referred to by different librefs at various times. To obtain a list of the libraries accessed through a server, use SLOGDATA.PHYSINFO.

### SLOGDATA.PHYSINFO

The PHYSINFO data set contains a list of the physical names that correspond to the libraries that were accessed through the server.

### SLOGDATA.ENGSUM1

The ENGSUM1 data set contains a list of the engines that were used to access SAS libraries through the server.

### SLOGDATA.MEMINFO

The MEMINFO data set contains an observation for each time a SAS library member is opened, reopened, closed, renamed, repaired, or deleted.

From this data set, which usually is very large, you can derive a list of members for each library that has been accessed through the server; the length of time and how many times each member was accessed; whether each member was created, read, or updated; and the number of simultaneous users of each member over a period of time.

You should use the physical name for the library because each library can be referred to by different librefs at various times.

### SLOGDATA.OBJINFO

The OBJINFO data set contains an observation for each time a SAS catalog entry is opened, closed, renamed, deleted, aliased, or has its directory information or options changed.

This data set is similar to SLOGDATA.MEMINFO, but it contains information for catalog entries instead of members of SAS libraries.

### SLOGDATA.IDXINFO

The IDXINFO data set creates an observation for each time a user creates or deletes an index through the server.

Because creating an index tends to be expensive, this data set is probably most useful as a warning signal. Also, because indexes can be very helpful for SAS data sets that are accessed concurrently, having a list of indexes that were deleted during a server's session can also be a warning signal.

### SLOGDATA.DIRINFO

The DIRINFO data set contains an observation for each time the directory of a SAS library or the directory of a SAS catalog is opened or closed through the server.

### SLOGDATA.IDXSUM

The IDXSUM data set contains only one observation with two variables. One variable counts how many indexes were created through the server; the other variable counts how many indexes were deleted through the server.

### SLOGDATA.ACCTINFO

The ACCTINFO data set contains one observation for each accounting message that is written to the server's log after a user disconnects. You must specify the LOG= option in the PROC SERVER statement to collect this data.

## SLTOOL3 and SLTOOL4 Sample Programs

You can examine the source code of these sample programs making the following menu selections: **Help ⇨ SAS Help and Documentation ⇨ Contents ⇨ SAS System Documentation ⇨ Learning to Use SAS ⇨ Sample SAS Programs ⇨ SAS/SHARE**

# Part 2

# Reference

*Chapter 9*

# The SERVER Procedure



## Overview of the SERVER Procedure

The SERVER procedure is the core of SAS/SHARE. It is the component that enables two or more clients to write concurrently to the same SAS file. To start a SAS/SHARE server, invoke the SERVER procedure. Specify an ID for that server with a set of optional parameters that define the server behavior.

You can use any SAS method of processing to invoke PROC SERVER: non-interactive mode, interactive line mode, batch mode, or the SAS windowing environments. For production, SAS/SHARE servers are usually run in batch mode; for interactive testing, they are usually run in interactive line mode.

The SERVER procedure is interactive, which means that the parser processes statements (that are called commands in PROC SERVER) as they are encountered. For details about how to invoke the SERVER procedure and for an example of a SAS log, see "Managing a SAS/SHARE Server (Server Administrators)" on page 29.

*Note:* Although the parser is accepting and processing commands, the SAS/SHARE server is not available to users until the RUN statement (or another program step) is executed.

## Syntax: The SERVER Procedure

**PROC SERVER** <*options*>;

**ALLOCATE SASFILE** *SAS-data-set1* <(*data-set-options*) >
<*SAS-data-set2*> <(*data-set-options*) ... *SAS-data-set8*<(*data-set-options*)>>;

**ALLOCATE LIBRARY** *libref* <*engine*> '*SAS-data-library*' <LIBTYPE=*library-type*>
<CATCACHELIMIT=*n*><*engine/system-options*>;

| Statement | Task |
|---|---|
| ALLOCATE SASFILE Statement | Specify data sets for the SAS/SHARE server to open and hold in memory. |
| ALLOCATE LIBRARY Statement | Defines a SAS library to a SAS/SHARE server. |

## PROC SERVER Statement

Starts a SAS/SHARE server session.

### Syntax

**PROC SERVER** <*options*>;

#### *Summary of Optional Arguments*

ACCTLVL=*value* | (*value 1* < …*value n* >)
: specifies the level of accounting.

ADMINLIBREF=*value* | _NONE_
: specifies the libref associated with the library of administrative data for the server.

ALLOC | NOALLOC
: determines whether clients can define additional SAS libraries to a server after a server has started.

AUTHENTICATE=REQUIRED | OPTIONAL
: controls whether a server requires connecting users to provide a valid user ID and password when they connect to the server.

CLIENTID=SESSION | SECURITY
: specifies the source of the client user ID to use in log messages that are written to the server log.

DTFORMAT=*SAS-datetime-format* | _NODTS_
: specifies the format for the date-and-time stamp at the beginning of each message that is written to the server log.

LOG=*value* | (*value 1* <... *value n*>)
: causes the server to log specific usage statistics about client/server transactions and SQL queries that it receives through the remote SQL pass-through facility.

LRPYIELD=*value*
: indicates, to a long-running process in the server's SAS session, how frequently the process should yield so that others can use the server.

MSGNUMBER
: associates a unique message number (represented in hexadecimal notation) with each type of operation for which a message is recorded in a server log.

NORMTVIEW
: disables the ability of a server to interpret SAS data views.

OAPW=*password* | "*encoded-password* "

specifies a password that you must supply (by using the OPERATE procedure) to connect to the server.

PT2DBPW=*password* | "*encoded-password*"
specifies the client password for controlling pass-through access to a remote DBMS.

SERVERID=*server-ID* | _ _*port-number*
specifies the name of a server.

TBUFSIZE=*value*
specifies the suggested size of a buffer that the server uses to transmit potentially large pieces of information to a client or receive such information from a client.

THREADEDTCP | NOTHREADEDTCP
specifies whether the threaded version of the TCP access method and associated threaded infrastructure should be used when TCP/IP communication is specified.

UAPW=*password* | *"encoded-password"*
specifies a password that a client must supply in the LIBNAME statement to establish communication with the server.

WORKTASKS = *initial* | ( <*initial*> <, *maximum*> )
specifies the initial and maximum number of work tasks for the SAS/SHARE server to execute.

## Optional Arguments

**ACCTLVL=*value* | (*value 1* < *…value n* >)**
specifies the level of accounting. Use this option to specify the aggregation (or detail) used in reporting server usage statistics. This option is useful for tuning an application because it enables you to examine, in more detail, how various parts of the application use the server resources. The usage statistics also enable you to charge users for the amount of server resources they consume.

The ACCTLVL= and LOG= options are closely related. The LOG= option specifies which usage statistic is written to the log; the ACCTLVL= option specifies the level of aggregation for that statistic (user, file, and resource environment). See the LOG= option for more information.

You can specify multiple values for the ACCTLVL= option by enclosing the values in parentheses and separating them with a space or a comma—for example, ACCTLVL=(USER DATA). Here are the valid values for ACCTLVL=:

ALL
causes the server to log statistics for all accounting levels (DATA, USER, and RESOURCE_ENVIRONMENT).

DATA
causes the server to log statistics every time a SAS file is closed by a user. These statistics show the usage of server resources for each file by that user.

RESOURCE_ENVIRONMENT
causes the server to log statistics every time a user terminates a *resource environment*. Examples of resource environments include a SAS procedure, a SAS window, a DATA step, a SAS process, or other internal SAS activity. These statistics show the usage of server resources by requests that are made in the context of that resource environment by that user.

USER
causes the server to log statistics every time a user disconnects from the server. These statistics show the usage of server resources for that client session.

**Default:** USER

**ADMINLIBREF=*value* | _NONE_**
specifies the libref associated with the library of administrative data for the server. This option enables a client to handle a server's administrative data in the same way it handles any other SAS file, which enables the client to process that data programmatically. To turn off this type of access to a server's administrative data, specify ADMINLIBREF=_NONE_.

```
/* This program segment creates the SAS data file WORK.A,
which is a list of all the libraries that clients of the
server are accessing. */
   libname sasadmin server=shr1;
   data a;
      set sasadmin.library;
      where users > '0';
run;
```

**Default:** SASADMIN

**ALLOC | NOALLOC**
determines whether clients can define additional SAS libraries to a server after a server has started. The ALLOC option enables clients to define libraries to a server. The NOALLOC option prevents clients from defining additional libraries to the server and restricts clients to accessing libraries that are defined by a server administrator.

**Default:** ALLOC

**AUTHENTICATE=REQUIRED | OPTIONAL**
controls whether a server requires connecting users to provide a valid user ID and password when they connect to the server. You can use this option with a communications access method security option (for example, use the TCPSEC option for TCP/IP).

AUTHENTICATE=REQUIRED means that the security options of all the access methods that are used by the server must be set to _SECURE_. Also, PROC SERVER will not start unless all of the access methods guarantee that they require a valid user ID and password combination before establishing a connection between a client and the server.

AUTHENTICATE=OPTIONAL means that the communications access method security options might be set to _SECURE_. Each access method can have a different security option. This can be useful for a site that trusts the IDs of users that connect through one access method but does not trust the IDs of users that connect through another access method. In this instance, only the access method used by the non-trusted clients would have its security option set to _SECURE_. Using the parameter OPTIONAL allows trusted users to connect without requiring validation.

When an access method supplies a validated user ID (usually, by requiring a connecting client to specify a valid user ID and password combination), the server uses that validated ID to verify the user's authorization to access SAS files. When AUTHENTICATE=OPTIONAL and some access methods do not supply validated user IDs, those clients are allowed to access only those SAS files that the server is allowed to access. In this instance, you should run the server under a user ID that does not have open access to the files on its computer.

**Alias:**

REQ

OPT

**Default:** REQUIRED

**CLIENTID=SESSION | SECURITY**

specifies the source of the client user ID to use in log messages that are written to the server log. Your choice determines the search order that is used to locate the client user ID. Choose the source of the client's name to be used in the server log, as follows:

SESSION

specifies that the source of the client user ID is either the logon name or the batch account name of the client session. The server uses the following search order:

- If a session name is located, the session name is used.
- If a session name is not located, but a secured name is located, the secured name is used. (For details, see the SECURITY value.)
- If neither a session name nor a secured name is located, the string _U_ is used as a generic identifier in the log messages.

SECURITY

specifies that the source of the user ID is the secured name that the client used to connect to the server. The server uses the following search order:

- If a client's secured name is located, the secured name is used.

  *Note:* A secured name is the user ID that is specified using the USERNAME= and PASSWORD= options in the LIBNAME statement. For example:

```
libname test userid=tbass password=_PROMPT_;
```

  For details about the LIBNAME statement, see "LIBNAME Statement" on page 123.

- If a secured name is not located, but a session name is located, the session name is used. (For details, see the SESSION value).
- If neither a session name nor a secured name is located, the string _U_ is used as a generic identifier in the log messages.

**DTFORMAT=*SAS-datetime-format* | _NODTS_**

specifies the format for the date-and-time stamp at the beginning of each message that is written to the server log. You can specify any SAS date, time, or date-and-time format; or you can specify your own date-and-time format. For details about specifying these formats, see *SAS Formats and Informats: Reference*.

If you specify your own date-and-time format in this option, a SAS datetime value is supplied to the formatting routine. For an example of the date-and-time format in the server log, see "Usage Statistics in the Server Log" on page 88.

Specifying the value _NODTS_ suppresses the date-and-time stamp.

**Alias:** DTF

**Default:** DATETIME22.3

**LOG=*value* | (*value 1 <... value n>*)**

causes the server to log specific usage statistics about client/server transactions and SQL queries that it receives through the remote SQL pass-through facility. This is useful for application tuning and for charging clients for the amount of server resources they consume. The server writes one line to its log for each resource statistic that is specified.

When this option is used with the ACTIVETIME, BYTECOUNT, ELAPSEDTIME, and MESSAGE values, the LOG= option is related to the ACCTLVL= option. The LOG= option specifies which usage statistic is written to the log; the ACCTLVL= option specifies the level of aggregation for that statistic (user, file, and resource environment). See the ACCTLVL= option for more information.

The LOG=QUERY option allows you to track SQL queries that are submitted through the remote SQL pass-through facility. This is useful to applications programmers for debugging and to server administrators to learn how the server is being accessed.

You can specify multiple values for the LOG= option by enclosing the values in parentheses and using a space or a comma to separate them—for example, LOG=(MESSAGE BYTECOUNT).

*Note:* The log values IO and CPU are no longer valid. The data that was reported by these options in SAS 6 offered limited accuracy, and the changes in resource tracking in SAS 8 reduced the potential accuracy even further. If you previously set the SAS options STIMER or FULLSTIMER because you had specified LOG=IO or LOG=CPU, you can reset or delete these SAS options in order to improve your server performance.

Here are the valid values for LOG=:

ACTIVETIME
causes the server to log the cumulative elapsed time of server processing for the specified event in the ACCTLVL= option. For example, if ACCTLVL=USER and LOG=ACTIVE, the server logs the cumulative elapsed time of processing for that client session when that client session ends. This value is printed using the SAS format TIME15.3. Here is an example of a log message that is recorded for this statistic:

```
30Apr2008:10:42:44.060 Usage statistics for user TIM(1):
                         Active time:              0:00:01.0394
```

**Alias:** ACTIVE

ALL
causes the server to log all the usage statistics (ACTIVETIME, BYTECOUNT, ELAPSEDTIME, and MESSAGE) for the corresponding accounting level. For example, if ACCTLVL=USER and LOG=ALL, the server logs statistics for ACTIVETIME, BYTECOUNT, ELAPSEDTIME, and MESSAGE for the client session when that client session ends.

If you specify LOG=ALL, the server will also log SQL queries that it receives through the remote SQL pass-through facility. See the QUERY option for an example of the SQL messages that are logged.

Here is an example of a log message that is recorded when ALL is specified:

```
30Apr2008:16:02:44.060 Usage statistics for user BILL(1):
                           Messages processed:                 47
                           Bytes transferred:                104 K
                           Active time:              0:00:05.0394
                           Elapsed time:             0:22:57.6912
```

BYTECOUNT
causes the server to log the cumulative number of bytes that are transferred between the client and the server for the event that is being logged. For example, if ACCTLVL=USER and LOG=BYTE, the server logs the cumulative number of bytes transferred between the client and the server for that client session when that client session ends. The value for BYTE is automatically scaled to make it

easier to read and the appropriate character is appended to the data. K=Kilobytes (1,024 bytes), M=Megabytes (1,048,576 bytes) and G=Gigabytes (1,073,741,824 bytes). The values are printed using the SAS format COMMA10.0. Here is an example of a log message that is recorded for this statistic:

```
30Apr2008:10:52:31.040 Usage statistics for user MIKE(2):
                              Bytes transferred:            40,052 K
```

**Alias:** BYTE

ELAPSEDTIME
causes the server to log the elapsed time of the recorded event. For example, if ACCTLVL=DATA and LOG=ELAPSED, the server logs the length of time that a file was open. If ACCTLVL=USER and LOG=ELAPSED, the server logs the length of time that the user was connected to the server. This value is printed using the SAS format TIME15.3. Here is an example of a log message that is recorded for this statistic:

```
30Apr2008:11:15:44.020 Usage statistics for user JOE(3):
                              Elapsed time:             0:22:57.6912
```

**Alias:** ELAPSED

MESSAGE
causes the server to log the number of client requests that are processed by the server for the recorded event. For example, if ACCTLVL=USER and LOG=MESSAGE, the server logs the number of requests processed for the client when that client session ends. This value is printed using the SAS format COMMA15.0. Here is an example of a log message that is recorded for this statistic:

```
30Apr2008:13:22:04.060 Usage statistics for user STEPHEN(4):
                              Messages processed:                  15
```

*Note:* Do not confuse LOG=MESSAGE with the MSGNUMBER option.

**Alias:** MSG

QUERY
causes the server to log each SQL query that it receives through the remote SQL pass-through facility from a SAS session or other client. By default, the server logs only update and output SQL statements, not queries. If LOG= QUERY, you will see messages similar to the following in your server log:

```
30Apr2008:15:14:12.898 GISELLE(14) in "SQL"(13) has issued select
                       flight, date, depart from home.chicago where
                       flight='202' to SQLVIEW.
```

REM
is an acronym for REMOTE engine emulation; when the server terminates, this value causes the server to log the total number of connections and maximum concurrent number of connections from thin clients that access the SAS/SHARE server. These clients include Share JDBC, ODBC, OLE DB, and SAS SQL.

**LRPYIELD=*value***
indicates, to a long-running process in the server's SAS session, how frequently the process should yield so that others can use the server. The default is 10000; this value has no units. Increase the value to have a long-running process yield more frequently; decrease the value to have it yield less frequently. LRPYIELD=0 does not yield at all. With a 0 value, the server can process other requests only after the long-running process has finished.

Here are two examples of long-running processes:

- A client accesses a PROC SQL view that joins two large data sets, which the server is required to sort.
- A client issues a WHERE clause that requires the server to search millions of observations, sequentially, to find the first observation that satisfies the WHERE clause.

**Default:** 10000

**MSGNUMBER**
associates a unique message number (represented in hexadecimal notation) with each type of operation for which a message is recorded in a server log.

***CAUTION:***
**Avoid hardcoding message numbers in your applications. Use macros instead.** Message numbers can change from one software release to another.

Message numbers are useful for server log analysis applications, which can count the number of instances of an operation that occur in a specific client/server session. Collection and analysis of these statistics might help with server load balancing. SAS provides a set of server log analysis program prototypes that you can customize for your needs. Among these prototypes is a file that maps message numbers to operations. See "Starting the Server Log" on page 97 for information about the server log analysis tools.

In the following example of a typical message written to a server log, the message number **`043131`** identifies a PROC SERVER start-up operation.

```
30Apr2008:08:28:20.911 043131 SAS server SHR1 started
```

*Note:* Do not confuse the MSGNUMBER option with the LOG=MESSAGE option.

**Alias:** MSGN

**NORMTVIEW**
disables the ability of a server to interpret SAS data views. By default, a SAS data view is interpreted in the server SAS session, and the data that is produced by the view is transmitted to a client SAS session.

Occasionally, it is preferable to transmit the view (the instructions for producing the data) to a client SAS session and to have the view interpreted and the data assembled in the server SAS session. You can use the RMTVIEW option in a LIBNAME statement to request this action on a library-by-library basis.

The NORMTVIEW option enforces transmission of the view (instead of the data) for all users of the server, regardless of whether the RMTVIEW option is specified in a LIBNAME statement.

*Note:* The NORMTVIEW option was developed for specific needs and is only rarely appropriate for use. To use this option for a server, contact SAS Technical Support to review the circumstances.

**OAPW=*password* | "*encoded-password* "**
specifies a password that you must supply (by using the OPERATE procedure) to connect to the server.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**PT2DBPW=*password* | "*encoded-password*"**
specifies the client password for controlling pass-through access to a remote DBMS. This password allows a connection between the SAS/SHARE server and another data server (either a DBMS or another SAS/SHARE server) that contains the target database.

Pass-through access to a remote database by means of a SAS/SHARE server requires that you also run the CONNECT TO REMOTE statement in PROC SQL with the DBMS= option and the PT2DBPW= option.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**SERVERID=*server-ID* | _ _*port-number***

specifies the name of a server.

*server-ID*

must be a valid SAS name that is 8 characters or less in length.

Server naming is affected by the operating environment and the access method that you specify for communication between a server and a client session. For example, if you use the TCP/IP communications access method, *server-ID* must be a valid TCP/IP service as defined in the TCP/IP SERVICES file.

For complete information about how to name servers by operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. For details about the SERVICES file, see Chapter 10, "TCP/IP SERVICES File," in *Communications Access Methods for SAS/CONNECT and SAS/SHARE* .

_ _*port-number*

If you are using the TCP/IP access method, you can specify the server's port number that corresponds to the server ID in the TCP/IP SERVICES file. Precede the port number with two consecutive underscores. For details, see the topic on the Chapter 10, "TCP/IP SERVICES File," in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

*Note:* Do not space after the first underscore or the second underscore. Example:

```
_ _1025
```

*Note:* Specifying a server by using a port number is not supported for ODBC clients.

**Alias:**

ID

SERVER

**TBUFSIZE=*value***

specifies the suggested size of a buffer that the server uses to transmit potentially large pieces of information to a client or receive such information from a client. When this option is not specified in the PROC SERVER statement, the value of the TBUFSIZE= SAS system option, if specified, is used.

One of the uses of these transmission buffers is for transmitting observations. The server uses the value of the TBUFSIZE= option when computing the number of observations to transmit in each multi-observation transfer between the server and the client sessions. If the observation size, plus overhead, exceeds the value of the TBUFSIZE= option, only single-observation transfers are executed.

You cannot calculate the number of observations per transfer by dividing the observation length into the value that you specify for the TBUFSIZE= option. To determine the effect of this option on your data sets, use the PROC SERVER options LOG=MESSAGE and ACCTLVL=DATA and compare the number of messages exchanged between the server and the client sessions as a function of the value of the TBUFSIZE= option and the number of observations in the data set.

**Default:** 128k

**THREADEDTCP | NOTHREADEDTCP**
specifies whether the threaded version of the TCP access method and associated threaded infrastructure should be used when TCP/IP communication is specified. The default is NOTHREADEDTCP.

When THREADEDTCP is specified, communication activity to and from the server is processed primarily in a threaded context. Communication activity refers to the work the server does to receive a request and to reply to the request, including any necessary data representation conversion. Threading enables multiple, concurrent reception and transmission activity when the server runs on SMP hardware.

Also, a portion of the main request processing, which occurs between the time the request is received and the reply time, is performed in the threaded context for SAS 9.0 and later clients.

Threaded TCP is compatible with the collection of active-time usage statistics. You specify the collection of statistics by setting either the LOG=ALL or LOG=ACTIVETIME option in the PROC SERVER statement. When THREADEDTCP is specified, active time can still be collected.

**Alias:**
TTCP

NOTTCP

**Default:** NOTHREADEDTCP

**UAPW=*password* | *"encoded-password"***
specifies a password that a client must supply in the LIBNAME statement to establish communication with the server.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **`srvmach`** is specified in the PROC PWENCODE statement. The output is generated in the form **`{key}encoded-password`**. **`sas001`** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**WORKTASKS = *initial* | ( <*initial*> <, *maximum*> )**
specifies the initial and maximum number of work tasks for the SAS/SHARE server to execute. A work task is typically a pair of lightweight threads that service requests from SAS/SHARE clients. When the server receives a request for service and no work task is available, the server spawns a new work task to service the request, up to the maximum number specified.

More work tasks enable the SAS/SHARE server to service more asynchronous requests with some of the work done in parallel.. However, the majority of work will happen synchronously as work tasks time-slice against a single processor. A larger number of work tasks does facilitate fair processor sharing when the number of concurrent requests is high.

*initial*
specifies the initial number of work tasks for the SAS/SHARE server to execute. The default is 3.

*maximum*
specifies the maximum number of work tasks for the SAS/SHARE server to execute. The default is 16.

**See:** THREADEDTCP | NOTHREADEDTCP on page 115

## Details

### Interaction between PROC SERVER and Other File-Locking Processes

Under the UNIX and Windows operating environments, if the FILELOCKWAITMAX= system option is not specified at SAS invocation, the FILELOCKWAITMAX= system option is set to zero for the duration of a server session that is started with PROC SERVER. This system option is used to specify the maximum time for a SAS session to wait to access a SAS file that is currently locked by another process.

*Note:* When using SAS/SHARE, you should not perform other processes that lock SAS files or use the FILELOCKWAITMAX= system option and the FILELOCKWAIT= option in the LIBNAME statement.

For details about the FILELOCKWAITMAX= system option and the FILELOCKWAIT= option in the LIBNAME statement, see the *SAS Companion for UNIX Environments* or the *SAS Companion for Windows*, as appropriate.

### PROC SERVER and the Default NOXCMD System Option

The XCMD and NOXCMD system options can be used to specify whether to enable host-system mode in order to submit an operating system command without ending the SAS session.

Using the XCMD option, an unauthorized user in a SAS/SHARE client session could access privileged data through a SAS/SHARE server session.

For SAS 9.2 and later releases, to prevent unauthorized access, if the XCMD system option is not specified at SAS invocation, the NOXCMD system option is set for the duration of a server session that is started with PROC SERVER.

*Note:* For SAS releases before SAS 9.2, XCMD is the default. Therefore, you must explicitly specify the NOXCMD system option in order to disable host-system mode.

For details, see the XCMD system option in the *SAS Companion for UNIX Environments*, the *SAS Companion for Windows*, or the *SAS Companion for z/OS*, as appropriate.

## Examples

### Example 1

The following statements start the server SHARE1:

```
proc server id=share1;
run;
```

### Example 2

The following statements start the server SHARE1 and prevent clients from defining SAS libraries to the server:

```
proc server id=share1 noalloc;
run;
```

### Example 3

The following statements start the server SHARE1 and report all message counts to the server log:

```
proc server id=share1 log=msg;
run;
```

# ALLOCATE SASFILE Statement

Specifies SAS data sets to open and keep in memory for the duration of time that a SAS/SHARE server session.

**Note:** You must define all librefs before using them in an ALLOCATE SASFILE command.

**Tip:** Keeping SAS data sets open can improve server performance by reducing the overhead that normally occurs when users open and close the data sets during application processing. A file remains open until the program or SAS session ends.

## Syntax

**ALLOCATE SASFILE** *SAS-data-set1* <(*data-set-options*) > <*SAS-data-set2* <(*data-set-options*)>…*SAS-data-set8*<(*data-set-options*)>>;

### Required Arguments

***SAS-data-set***
contains descriptor information and its related data values organized as a table of observations and variables that can be processed by SAS.

***data-set-options***
specify actions that apply only to the SAS data set with which they appear. For complete details about data set options, see *SAS Data Set Options: Reference*.

## Details

### ALLOCATE SASFILE Command Considerations

Consider the following when using the ALLOCATE SASFILE command with the PROC SERVER statement:

- A maximum of 8 SAS data set names can be specified in each ALLOCATE SASFILE command.
- The SAS data sets that you specify must exist before the ALLOCATE SASFILE command is issued.
- Only SAS data sets can be specified. Other types of SAS files (for example, catalogs) cannot be specified in an ALLOCATE SASFILE command.
- When you open a SAS data set by using the ALLOCATE SASFILE command, the file is opened for input processing and can be used for subsequent input or update processing. However, the file cannot be used for subsequent utility or output processing, because utility and output processing require exclusive access to the file (member-level locking). For example, you cannot replace the file or rename its variables.
- The ALLOCATE SASFILE command can execute only in the server session.
- After the ALLOCATE SASFILE command executes, all users who subsequently open the file will access the data that is held in memory instead of the data that is stored on the disk.
- After the ALLOCATE SASFILE command executes, the file is closed and the buffers are freed only after the SAS/SHARE server is terminated.
- Do not specify the same data set in both a SASFILE statement and an ALLOCATE SASFILE command.
- You must execute the SASFILE statement before you execute the PROC SERVER statement.

### *Comparison: ALLOCATE SASFILE Command and SASFILE Statement*

The ALLOCATE SASFILE command is similar and complementary to the SASFILE statement in Base SAS. The SASFILE statement can be used in a server session as well as in a single-user session. Both statements used in a server session achieve performance gains by providing in-memory processing using buffers.

A buffer is a reserved area of memory that holds a segment of data while it is processed. The number of allocated buffers determines how much data can be held in memory at one time.

The ALLOCATE SASFILE command offers limited buffering. The SASFILE statement in Base SAS provides maximum buffering, and therefore, the best performance. You can specify any of the following four levels of file buffering (shown in the following table) for each data set.

***Table 9.1*** *Levels of File Buffering*

| Level of File Buffering | Memory Consumed | Condition for Using |
|---|---|---|
| Neither the ALLOCATE SASFILE command nor the SASFILE statement is used. | The least amount of memory is consumed. Each client that accesses a file duplicates the same overhead that is required for a file open. | File access is limited and memory is constrained. |

| Level of File Buffering | Memory Consumed | Condition for Using |
|---|---|---|
| Use the ALLOCATE SASFILE command, and accept the default number of buffers that are pre-allocated. | The SAS/SHARE server opens the file and keeps it open for all client access, which eliminates the duplicate overhead that is required for a file open. | File access is frequent and memory is constrained. |
| Use the ALLOCATE SASFILE command, and use the BUFNO= data set option to specify the number of buffers to pre-allocate. | Specify buffers according to available memory and the usage pattern of the file. | The file being accessed is large, but only certain pages of the data are accessed frequently. |
| Use the SASFILE statement to read the entire data set into memory. | The entire file resides in memory. | File access is frequent and there is sufficient memory for reading in the entire file. |

For details about the SASFILE statement in Base SAS, see the following topics:

- "BUFNO= System Option" in *SAS System Options: Reference*
- "BUFNO= Data Set Option" in *SAS Data Set Options: Reference*
- "SASFILE Statement" in *SAS Statements: Reference*

.

### Example

In the following example, server SHARE1 is started and eight data sets are specified to be stored in memory for client access. The number of buffers that are used is determined by the default value of the BUFNO= system option.

```
proc server id=share1;
allocate sasfile sas-dataset1 sas-dataset2 ... sas-dataset8;
run;
```

## ALLOCATE LIBRARY Statement

Defines a SAS library to a SAS/SHARE server.

**Restriction:** The SERVER procedure

**Tip:** provides the same functionality as the ALLOCATE LIBRARY command in the OPERATE procedure. In addition, it enables you to specify the library and a libref to use for caching catalog files that have been accessed by using the option LIBTYPE=CATCACHE, which is a Cross-Architecture Catalog Access feature.

**See:** "Cross-Architecture Catalog Access in the SAS/SHARE Server" on page 188

### Syntax

**ALLOCATE LIBRARY** *libref* <*engine*> *'SAS-data-library'* <LIBTYPE=*library-type*> <CATCACHELIMIT=*n*> <*engine/system options*>;

### Required Arguments

***libref***
is a valid SAS name that is temporarily associated with a SAS library.

**'*SAS-data-library*'**
is the physical name of the SAS library that is being defined to the server. The name is specific to your operating environment and must be enclosed in either single or double quotation marks.

### Optional Arguments

***engine***
specifies the engine to be used to process the SAS library when the server executes SAS.

*Note:* Usually, you do not have to specify this option because the server automatically determines which engine to use for processing a data library. However, you can specify this option to reduce the time that is used by the server when determining which engine to use to access a specific SAS library.

**LIBTYPE=*library-type***
specifies the type of library to be allocated. The following library types are supported:

CATCACHE
specifies that this library will be used to cache catalogs that have been accessed by means of the cross-architecture catalog access feature. This option gives you the flexibility to specify a libref and a location for these files. Only one catalog cache might be allocated per SAS/SHARE server invocation.

By default, the SAS/SHARE server uses its WORK library as the location for the catalog cache and assigns the libref SASCATCA to it. In most instances, the default is suitable.

STANDARD
specifies a standard SAS data file library. This is the default.

**CATCACHELIMIT=*n***
is a numeric value that specifies the maximum number of catalog files that are held in the cache. The files are accessed by means of the cross-architecture catalog access feature. The following values can be used:

0 specifies that the catalog files are not cached.

1 specifies that all catalog files are saved in the catalog cache.

*n* specifies that the cache is limited in size to *n* files. The cache is maintained, and the files are discarded by using a least-recently-used algorithm. The *n* argument is valid only when specifying a library type of CATCACHE. By default, the server limits the number of catalogs cached to 3.

***engine/system-options***
are options that apply to the SAS library. This argument is not required. You can specify any option that is valid in the LIBNAME statement for a specific operating environment and engine. Using the form *keyword=value*, you can specify as many options as you need. Use a blank space to separate options. See the SAS documentation for your operating environment for a complete list of the options that are available for your operating environment and engine.

## Example

- The following example allocates the UNIX server library **/data/sales** that has the libref SALES. It specifies that the server use the V9 engine to process this library.

  ```
  ALLOC LIBRARY SALES V9 '/data/sales';
  ```

- The following example allocates the z/OS server library **SAS.CATALOG** (as a catalog cache) that has the libref SRVCATS.

  ```
  AL LIB SRVCATS 'SAS.CATALOG' LIBTYPE=CATCACHE;
  ```

*Chapter 10*

# Remote Library Services



## Overview of Remote Library Services

The LIBNAME statement implements the SAS/SHARE Remote Library Services (RLS), which provides transparent access to remote data libraries to move data through the network as it is requested by the local SAS session.

A LIBNAME statement associates a SAS library reference (libref) with a permanent SAS library. In SAS/SHARE software, the SAS library is accessed through a SAS server and is called a *server library*.

## Dictionary

### LIBNAME Statement

In a client session, associates a libref (a shortcut name) with a SAS library that is located on the server for client access. In a server session, predefines a server library that clients are permitted to access.

**Valid in:** client and server sessions

**Category:** Data Access

**Operating environment:** “LIBNAME Statement: UNIX” in *SAS Companion for UNIX Environments*, “LIBNAME Statement: Windows” in *SAS Companion for Windows*, and “LIBNAME Statement: z/OS” in *SAS Companion for z/OS*.

**See:** Base SAS “LIBNAME Statement” in *SAS Statements: Reference*.

#### Syntax

**LIBNAME** *libref* <*engine*> < '*SAS-data-library*'> SERVER=<*server-node.*> *server-name* | *__port-number* <*options*> ;

### Required Arguments

***libref***

For a server, specifies the name of a library reference (predefines a library) for client access. One or more LIBNAME statements may be issued before the server is started. If you predefine server libraries and want to limit client access to only the predefined server libraries, use the NOALLOC option in the PROC SERVER statement. Specifying the ALLOC option in the PROC SERVER statement permits clients to assign server libraries for use after a server is started.

For a client, specifies the name of a server library for client access.

***CAUTION:***

**A client's ability to access a SAS/SHARE server library depends on the permissions granted at server start-up.** If PROC SERVER ALLOC is specified, clients can access both predefined libraries and libraries that have not already been allocated at server start-up. If PROC SERVER on page 105 NOALLOC is specified, client access is limited to predefined server libraries. For details about the PROC SERVER statement, see Chapter 9, "The SERVER Procedure," on page 105.

The libref that you specify is presumed to be the server libref for an existing server library unless you specify the SLIBREF= option or the physical name of the data library.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

***engine***

specifies the name of a valid SAS engine for a client to use to access the server library. Usually, you should not use this option because the client automatically determines which engine to use for accessing a SAS/SHARE server. Specify this option only to override the SAS default for a specific server, or to reduce the time that is needed to determine which engine to use to access a specific server.

For example, if the server library is located on a SAS/SHARE server that is running SAS 9.3, you could specify REMOTE9. Specifying an explicit engine might improve performance slightly.

Examples of engines include REMOTE, REMOTE8, and REMOTE9. For a list of valid engines, see the SAS documentation for your operating environment. For background information about engines, see *SAS Language Reference: Concepts*.

The *engine* parameter is positional. If you use it, it must follow the libref.

**Note:** Do not confuse the *engine* positional parameter with the RENGINE= option. An engine is used by a client to access a server. An RENGINE is used by the server to access its SAS library.

***'SAS-library'***

specifies the physical name for the SAS library (on the server) that the client will access. If you specify a server library either as the libref or as the value for the SLIBREF= option, you must omit the physical name.

If you specify *'SAS-library'*, the name must be a valid SAS name, and it must be enclosed in single or double quotation marks. For details about specifying a SAS library, see the documentation that is appropriate to your operating environment.

**SERVER=<*server-node*.>*server-name* | _ _*port-number***

specifies the location and identity of the SAS/SHARE server session that administers the SAS library.

*server-node*
specifies the node name (machine name or IP address) of the server where the SAS library is located. The node name must be a valid SAS name that is 8 or less characters in length. Specifying *server-node* is necessary when using the TCP/IP communications access method and the client session is on a different machine from the server.

If the node name exceeds eight characters, you can either specify the node name by defining it as a macro variable or you can use the HOSTNAME= option to specify the node name as a quoted string. The following example shows how to use the macro statement %LET to assign the node name MYHOST.MY.NET.WORK to the macro variable, MYRLIB, and then use that variable in place of the invalid node name in the LIBNAME statement.

```
%let myrlib=myhost.my.net.work;
libname refserv server=myrlib._2323;
```

See "Example 3: Assigning a Server Host to a Macro Variable" on page 132 for another example of how to use a macro variable with an invalid SAS name.

Another way to specify the node name when it is not a valid SAS name is to use the HOSTNAME= option (see HOSTNAME=node-name on page 126 ). In the example below, the server library, TEST, is defined in the server session SHR1, which runs on the node STONES.UNX.APEX.COM. The node name is specified as a quoted string that has a valid length.

```
Libname test hostname='stones.unx.apex.com' server=shr1;
```

Server naming is also affected by the operating environment and the access method that you specify for communication between server and client session.

For complete information about how to name servers by operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. For details about the SERVICES file, see the topic on the Chapter 10, "TCP/IP SERVICES File," in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. To get the value for the *server-ID* in your environment, consult your server administrator.

*server-name*
specifies the particular server session that administers the SAS library. The name must be a valid SAS name, 8 or less characters in length.

- For the TCP/IP communications access method, this must be the name of a valid TCP/IP service as defined in the TCP/IP Services file.
- For the XMS communications access method on z/OS, this name must be a unique identifier with respect to your cross-memory subsystem anchor. For details, see "SAS/SHARE SUBSYSID= Option " in Chapter 5 of *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

*_ _port-number*
If you are using the TCP/IP access method, you can specify the server's port number that corresponds to the server ID in the TCP/IP SERVICES file. Precede the port number with two consecutive underscores. (For details about the SERVICES file, see the topic on the Chapter 10, "TCP/IP SERVICES File," in *Communications Access Methods for SAS/CONNECT and SAS/SHARE* ).

*Note:* Do not space after the first underscore or the second underscore.

Example:

```
_ _1025
```

### Optional Arguments

**ACCESS=READONLY**
restricts a client's access to a SAS library via a multi-user SAS/SHARE server. If ACCESS=READONLY is specified in the client session, the client can read but not update data in the library. However, other clients might have read/write access to the library via the server.

If ACCESS=READONLY is specified in the server session, all clients are limited to read-only access to the library via the server. No clients will have update access.

**AUTHDOMAIN=*auth-domain* | "*auth-domain*"**
specifies the name of an authentication domain, which is a metadata object that manages the credentials (user ID and password) that are associated with the specified domain. Specifying the authentication domain is a convenient way to obtain the metadata-based user credentials rather than having to explicitly supply them during server sign-on.

An administrator can define an authentication domain using the **User Manager** in SAS Management Console. See *SAS Management Console: Guide to Users and Permissions* for information about adding an authentication domain.

Examples:

```
authdomain=DefaultAuth
authdomain="SAS/SHARE Auth Domain"
```

**Requirements:**
The authentication domain and the associated credentials must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Enclose domain names that are not valid SAS names in double or single quotation marks.

**Interaction:** If you specify AUTHDOMAIN=, do not also specify USERNAME= and PASSWORD=.

**See:**
For complete details about creating and using authentication domains, see *SAS Intelligence Platform: Security Administration Guide*.

*SAS Management Console: Guide to Users and Permissions* and SAS Management Console online Help

**HOSTNAME=*node-name***
is used to specify the name of the node that the SAS/SHARE server runs on. The value for *node-name* can be specified as a quoted string that does not exceed 256 characters or as an unquoted SAS name that does not exceed 32 characters.

HOSTNAME= is used in conjunction with the SERVER= option, which specifies the name of the SAS/SHARE server that runs on the node. If a two-level node name (*node.server-ID*) is assigned to the SERVER= option, and the HOSTNAME= option is also specified, duplicate node names would result and seem to be ambiguous. However, to resolve the ambiguity, the value that is specified as the final option in the LIBNAME statement takes precedence.

***Table 10.1*** *HOSTNAME= Option Examples*

| Syntax | Description |
|---|---|

| | |
|---|---|
| `libname test hostname=sirlancelot server=shr1;` | A server library is defined in the server session SHR1 that runs on the node SIRLANCELOT. The node name is specified as an unquoted SAS name that has a valid length. |
| `libname test hostname='stones.unx.apex.com' server=shr1;` | A server library is defined in the server session SHR1 that runs on the node STONES.UNX.APEX.COM. The node name is specified as a quoted string that has a valid length. |
| `libname test server=d8433.shr1 hostname=defiant;` | A server library is defined in the server session SHR1 that runs on the node DEFIANT. The node name is specified as an unquoted string that has a valid length.<br><br>If both the SERVER= option and the HOSTNAME= option specify the node name, the option that is specified last takes precedence. In this example, the value DEFIANT, which is assigned to the option HOSTNAME=, takes precedence over the value d8433 in the two-level name that is assigned to the option SERVER=. |
| `libname test hostname=notused server=d8433.shr1;` | A server library is defined in the server session SHR1 that runs on the node d8433. The node name is specified as an unquoted string that has a valid length. |

If both the SERVER= option and the HOSTNAME= option specify the node name, the option that is specified last takes precedence. In this example, the value d8433, in the two-level name that is assigned to the option SERVER=, takes precedence over the value NOTUSED, which is assigned to the option HOSTNAME=.

**PASSWORD=*password* | "*encoded-password*" | _PROMPT_**
executed in the client session, specifies a password that is valid on the server. This parameter is used by the server to validate the client on the server's operating environment (if authentication is enabled). For details about valid passwords, see Details on page 131 .

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

Here is an example of using an encoded password in a LIBNAME statement:

```
libname mylib server=shr1 user=jward password="{sas001}c2Vydm1hY2g=";
```

**Alias:**
PASSWD
PASS
PWD
PW

**Interaction:** If you specify PASSWORD=, do not also specify AUTHDOMAIN=.

**See:**
AUTHDOMAIN=auth-domain | "auth-domain" on page 126.
USERNAME=user-ID | _PROMPT_ on page 130

**RENGINE=*engine-name***
executed in the server session, specifies the engine to be used to process the SAS library. Using this option is usually unnecessary because the server automatically determines which engine to use to process the data library. Specify this option only to override the SAS default for a specific library, or to reduce the time that is used by the server to determine which engine to use.

**Note:** Do not confuse the RENGINE= option with the *engine* positional parameter. An RENGINE is used by the server to access its SAS library. An engine is used by a client to access a server. Do not use the SPD engine as a remote engine.

**RMTVIEW=YES | NO**
specified in a client session, determines whether SAS data views are interpreted in the server or in the client SAS session. Where a data view is interpreted determines where the view engine is loaded and used. The default is YES.

SAS data views include DATA step views and PROC SQL views, which are created by using the SQL procedure and the ACCESS procedure (in SAS/ACCESS software). SAS data views are accessed through an engine just as other SAS data sets are. DATA step views use the SASDSV engine. PROC SQL views use the SQLVIEW engine. SAS/ACCESS views use a product-specific engine (supplied by SAS Institute) for each SAS/ACCESS interface product.

RMTVIEW= YES (the default) causes views to be interpreted in the server's SAS execution. This uses more processing time and might increase the amount of memory used. However, the amount of data transferred to the client SAS sessions might be reduced.

RMTVIEW=NO causes views to be interpreted in the client SAS session. This minimizes the processing time on the server but might increase the amount of data transferred between the server and client SAS sessions. Also, if you specify RMTVIEW=NO, there might be version incompatibilities when the client and server are running different versions of SAS. For example, SAS 6 and SAS 9.3 views are not always compatible. For details about view interpretation, see “Interpreting SAS Data Views” on page 54.

**Default:** YES

**ROPTIONS="*option=value <option=value>…* "**
executed in the server session, specifies remote options and options that are specific to an operating environment that the client passes to the engine on the server that will process the SAS library. Specify as many options as you need by using the form *keyword=value*. Use a blank space to separate options. You can specify options for either the default engine or an alternative engine that you specify by using the RENGINE= option. You can use the option ROPTIONS= to provide any valid option for the targeted engine. For information about the options that are supported by a specific engine, see the documentation for the engine that you will use. For details about options that are specific to an operating environment, see the documentation that is appropriate for the operating environment that you use.

***CAUTION:***
**If allocating a library to a SAS/SHARE server, the DISP= host option is not allowed with ROPTIONS= for security reasons.**

**Restrictions:**
If one or more client sessions that run under UNIX or Windows use the FILELOCKWAIT= option in the ROPTIONS= statement to set the maximum time limit that SAS will wait for a file to be unlocked (available for use), the effect could cause the server session to stall. The SAS/SHARE server memory can become inadvertently consumed by multiple tasks that are waiting for the release of one or more locked files.

To prevent the server session from stalling, the SAS/SHARE server administrator, when invoking the SAS session from which the server session will run, can use the FILELOCKWAITMAX= system option to explicitly set the client wait time to zero. Negating the client's specified wait time prevents the server from stalling.

**See:**
For the client session FILELOCKWAIT= option in the LIBNAME statement, see *SAS Companion for UNIX Environments* and *SAS Companion for Windows*.

For the server session FILELOCKWAITMAX= system option, see *SAS Companion for UNIX Environments* and *SAS Companion for Windows*.

"Interaction between PROC SERVER and Other File-Locking Processes" on page 116

**SAPW=*server-access-password* | "*encoded-password*"**
executed in the client session, specifies a server access password. This option is needed to access a SAS/SHARE server that is executing with the UAPW= option in PROC SERVER in effect. SAPW= establishes communication with the server that is used to access the library. Although this option is specified in the LIBNAME statement, it does not control access to the server library itself. For details about valid passwords, see Details on page 131.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**SLIBREF=*server-libref***
executed in the client session, specifies an existing libref that was defined in the server session that you want to reference (or copy) in the client session. Use this option when you want to reference an existing server libref, but you want to use a different name for that libref in the client session.

**Restriction:** Do not assign the physical name for the SAS library on the server to SLIBREF=. SLIBREF=*server-libref* and *'SAS-data-library'* are mutually exclusive.

**Interaction:** In a client session, if you define a user library whose libref duplicates a libref that has been predefined in the server session, the user libref overrides the server's predefined libref for the duration of the client session only. The server inherits the client's user libref.

**See:** "Methods for Predefining a Server Library" on page 32

**Examples:**

"Example 7: Associating a User Libref with a Server Libref" on page 132

"Example 8: Duplicating a Server Libref" on page 132

"Example 9: Server Inheritance of User Librefs That Are Associated with Server Librefs" on page 133

**CAUTION: Avoid using duplicated librefs.** Even though duplicated librefs are valid and the resolution of duplicated librefs is logical, the practice can be confusing.

**USERNAME=*user-ID* | _PROMPT_**
executed in the client session, specifies a user ID that is valid on the server. This parameter is used in two ways. The server uses it to validate the client on the server operating environment (if authentication is enabled). The server also uses it to verify access permission when the client accesses files on the server. For details about valid user IDs, see Details on page 131.

If USERNAME=_PROMPT_, a dialog box appears that contains a message that prompts the user to enter a valid user ID. This enables you to specify the value at program execution instead of coding it into the program. Using _PROMPT_ is a way to enforce security.

**Alias:** USERID, USER, UID

**Interaction:** If you specify USERNAME=, do not also specify AUTHDOMAIN=.

**See:**

AUTHDOMAIN=auth-domain | "auth-domain" on page 126
PASSWORD=password | "encoded-password" | _PROMPT_ on page 127

## Details

Here are the general rules for creating user IDs and passwords:

- Mixed case is allowed (for example, **user=JoeBlack**).
- Quotation marks must enclose values that have the following characteristics:
  - contain one or more spaces (for example, **user='joe black'**).
  - contain one or more special characters (for example, **user='joe?black'**).
  - contain one or more quotation marks (for example, **password="It's mine"**).
  - begin with a numeric value (for example, **password='2BorNot2B'**).
  - do not conform to rules for user-supplied SAS names.
  - are NULL values (for example, **user=''**).

    NULL values are sometimes used in a UNIX environment, when you want to use the local ID. See *Communications Access Methods for SAS/CONNECT and SAS/SHARE* for details.
  - are user names that contain domain information (for example, **user='apexdomain\joe'**).

    Domain information is sometimes included in the Windows environment.
- The SAPW= option requires that you specify a password that is 1 to 8 characters in length.
- Specify the value _PROMPT_ if you want SAS to prompt you for information (for example, **password=_prompt_**).

  Using _PROMPT_ increases security by causing SAS to prompt you for a password instead of coding the password in the LIBNAME statement.

SAS limits each user name and password to 256 characters. The operating environment in which SAS is running might also impose restrictions on user names and passwords. For details, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

## Examples

### *Example 1: Client Using the Server's Libref*

**The client uses the existing server libref SALES to point to a server library that is located on SERVER1.**

```
libname sales server=server1;
```

### *Example 2: Client Assigning a Libref to a Physical Path*

**The client assigns the libref SQLDSLIB to the SAS library SASXYZ.VIEWLIB.SASDATA that is located in the SAS/SHARE session SERVER7.**
In this example, the client is permitted to access a server library that has not been predefined at the server.

```
libname sqldslib 'sasxyz.viewlib.sasdata' server=server7;
```

### Example 3: Assigning a Server Host to a Macro Variable

**The client assigns the libref REMNAME to the SAS server MYHOST.MY.NET.WORK** The LIBNAME statement establishes the libref SCORCARD to point to a library on the server that is defined by REMNAME.

```
options comamid=tcp;
%let remnode=myhost.my.net.work;
libname scorcard remote server=remnode._2323;
```

### Example 4: Using a Server Password to Access a Server Library

**The client associates the libref EDUCLIB with the SAS library SASDEMO.EDUCCATS.SCREENS that is located on the server ABCSERV.** Users must specify the password DEMOPW in order to access this server.

```
libname educlib 'sasdemo.educcats.screens' server=abcserv sapw=demopw;
```

### Example 5: Specifying a Prompt for a Password to Access a Server Library

**Rather than hardcoding the user's password to access a server library, the client specifies that a prompt for password be displayed.**

```
libname mygrade slibref=grades server=shr1 user=bass password=_prompt_;
```

### Example 6: Specifying an Encoded Password in a LIBNAME Statement

**To prevent the risk of exposing clear-text passwords in stored SAS programs, the client specifies an encoded password.**

```
libname sales server=server1 userid="myuserid" password="{sas001}c2Vydm1hY2g=;
```

### Example 7: Associating a User Libref with a Server Libref

**The client associates the new user libref MKTDATA with an existing server libref MARKETD in the SAS/SHARE session SERVER1.** The user libref MKDATA is a copy of the server libref MARKETD. For this client, the server inherits the client's user libref MKTDATA.

```
libname MKTDATA slibref=MARKETD server=server1;
```

### Example 8: Duplicating a Server Libref

**Two predefined libraries are created in a server session before the server session is started.** These libraries are available to any client to access through the SAS/SHARE server session.

```
libname shoe 'C:\myshoe';
libname blue 'C:\myblue';
proc server id=shr1;
run;
```

**In the client session, the user defines a new libref BLUE, which is a copy of the server libref SHOE, which points to C:\MYSHOE.** The client's user libref BLUE overrides the server libref BLUE in this client session only. For this client , the server has the server-defined libref SHOE, and the server has inherited the client's user libref BLUE. The user libref BLUE will override the server libref BLUE for this client only.

```
libname blue slibref=shoe server=shr1;
```

**As another example, if the user defined a new user libref GLUE that referred to the server libref BLUE, would libref BLUE refer to the client's user libref BLUE or to the original server libref BLUE?** In this example, because the server has already inherited the client's user libref BLUE, GLUE will refer to the client's user libref BLUE in this client session only. The order in which the library assignments occurs is important. If libref GLUE were assigned before libref BLUE, then libref GLUE would be assigned to C:\MYBLUE rather than to C:\MYSHOE.

```
libname glue slibref=blue server=shr1;
```

## *Example 9: Server Inheritance of User Librefs That Are Associated with Server Librefs*

**Here are two predefined libraries that are created in a server session:**

```
libname zoo 'C:\blue';
libname zoo 'C:\glue';
proc server id=shr1;
run;
```

**In this example, the user libref MYLIB is assigned to the server libref ZOO.**

```
libname mylib slibref=zoo server=hrhost.shr1;
```

**Because the server performs work on behalf of the client, the server inherits the user libref MYLIB.** The SAS logs, which can be viewed in the client and server sessions, indicate that the librefs of the client and the server are identical. Libref MYLIB points to two physical files C:\blue and C:\glue, which are concatenated.

```
34   libname mylib slibref=zoo server=hrhost.shr1;
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:        REMOTE
      Physical Name: ( 'C:\blue' 'C:\glue' )
35   proc datasets lib=mylib;
                                          Directory
Libref                                    MYLIB
Engine                                    REMOTE
Physical Name                             ( 'C:\blue' 'C:\glue' )
Accessed through server                   HRHOST.SHR1
Server's libref                           MYLIB
Server's engine                           V9
Server's SAS release                      9.02
Server's host type                        NET_SRV -
Server's versus user's data representation SAME
Views interpreted in server's execution   YES
File Name                                 C:\ blue
```

*Chapter 11*

# The OPERATE Procedure



## Overview of the OPERATE Procedure

You can use the OPERATE procedure in any SAS method of processing (noninteractive mode, interactive-line mode, batch mode, or windowing environment) to manage a server, the server libraries, and the server users. Using PROC OPERATE, you can do the following tasks:

- define a SAS library to a server after the server has started

- display information about assigned libraries
- release libraries from assignment
- terminate access to a library
- display IDs of users who are connected to the current server
- manage the server from a session other than the server session

The OPERATE procedure is interactive; that is, its statements are executed as they are encountered. For this reason, statements used in the OPERATE procedure are called commands. The OPERATE procedure executes until it is terminated by a QUIT or a RUN command. The syntax for these commands is discussed later in this section. Here is the syntax for the PROC OPERATE statement.

# Syntax: The OPERATE Procedure

**PROC OPERATE** <*options*>
*Library Management Commands*
*Server Management Commands*
*User Management Commands*
;

# OPERATE Statement

## Syntax

**PROC OPERATE** <*options*;>

### *Optional Arguments*

**SERVER=*server-ID* | _ _*port-number***
identifies the default server session to be managed. If this option is not specified, you must identify the server in the SET SERVER command or in those PROC OPERATE commands that allow you to identify the server to be managed. For details, see "Specifying a Server" on page 150.

If you are using the TCP/IP access method, you can specify the server's port number that corresponds to the server ID in the TCP/IP SERVICES file. Precede the port number with two consecutive underscores. (For details about the SERVICES file, see the topic on the Chapter 10, "TCP/IP SERVICES File," in *Communications Access Methods for SAS/CONNECT and SAS/SHARE* in the *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.)

*Note:* Do not space after the first underscore or the second underscore.

Example:

```
_ _1025
```

**Alias:**
ID
SERVERID

**PRINTFILE=LOG | PRINT**
directs the output from the OPERATE procedure. PRINTFILE=LOG directs the output to the SAS log. PRINTFILE=PRINT directs the output to the procedure output file or Output window.

**Alias:** PF

**Default:** LOG

**SAPW=*password* | "*encoded-password*"**
specifies a server-access password. This password is required to access a SAS/SHARE server that is executing with the OAPW= option in PROC SERVER in effect.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**USER=*user-ID*| _PROMPT_**
specifies the user ID of the accessing client on the server. The operating environment on which the client runs can also affect user ID conventions. For details about user ID conventions that are imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Here are the valid values for this option:

*user-ID*
For details about a valid user ID, see Details on page 131.

_PROMPT_
specifies that SAS prompt the user for a valid user ID. Using _PROMPT_ is a way to enforce security.

**Alias:**
USERNAME
USERID

UID

**PASSWORD=*password* | "*encoded-password*" | _PROMPT_**
specifies the password of the accessing client on the server. The operating environment on which the client runs can also affect password naming conventions. For details about password naming conventions that are imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**Alias:**
PASSWD
PASS
PWD
PW

**SETSASRC=(YES | NO)**
causes the return code of PROC OPERATE to be surfaced to the operating environment when SAS execution terminates. For example, when submitting PROC OPERATE commands under batch mode under z/OS, SETSASRC=YES specifies that a nonzero return code from PROC OPERATE cause a condition code to be set in the JES message log for the batch job that invoked PROC OPERATE.

If SETSASRC=NO, the condition code for the batch job will be 0 regardless of whether the PROC OPERATE commands executed successfully.

**Default:** NO (disabled)

# Library Management Commands

## *Overview of Library Management Commands*

Beginning with SAS 8, multiple users can define different SAS libraries with the same libref name. Enhancements to SAS enable the server engine to differentiate among multiple user sessions that might use the same libref and to manage them appropriately.

## *Defining a Library to a Server That Is Running*

You use the ALLOCATE LIBRARY command to define a SAS library to a server that is already running. Each library that you define requires a separate ALLOCATE LIBRARY command. Here is the syntax for this command:

**ALLOCATE LIBRARY** *libref <'SAS-data-lib'>* <RENGINE=*engine-name* > <*engine/environment options*>;

*libref*
identifies the SAS library that is specified in the *'SAS-lib'* argument.

*'SAS-lib'*
specifies the physical name of the SAS library that is being defined to the server. This name is specific to the operating environment and must be enclosed in single or double quotation marks. See "Specifying a SAS Library" on page 142 for examples by operating environment.

RENGINE=*engine-name*
specifies the engine to be used to process the SAS library in the server's SAS execution. Usually, this option is not used because the server determines which engine to use to process the data library. Specify this option only if you want to override the SAS default or to reduce the search time that is required by the server to determine which engine to use to access a specific SAS library.

*engine/environment options*
are options that apply to the SAS library. You can specify any option that is valid in the LIBNAME statement for a specific operating environment and engine. You can specify one or as many options as you need by using the form *keyword=value*. Use a blank space to separate options. This argument is not required.

These options are effective in the server SAS session, not in the user SAS session. For a complete list of options that are available for your operating environment and engine, see the SAS documentation for your operating environment.

Alias ALLOC, AL

## *Displaying Information about a Library*

You use the DISPLAY LIBRARY command to display information about one or more server libraries that are defined to the current server. The information includes the server libref, the physical name of the data library, its status, and the number of users that are accessing it. This information for a server library that is defined by an operating environment-specific external method is reported only after the library becomes active.

A library becomes active after the user issues a LIBNAME statement to access a server library. For information about methods to define libraries to the server, see "Predefining SAS Libraries to the Server" on page 32. Here is the syntax for this command:

**DISPLAY LIBRARY** *libid-1*<*…libid-n*>;

**DISPLAY LIBRARY** _ALL_ ;

*libid*
specifies either a libref for a predefined server library or a physical name for a server library. Detailed information is displayed for the specified *libid*. For example, this command produces detailed information, as follows:

```
display library 'SAS-lib';
USER              USER LIBREF
------------------------------
IAN(5)            FEES
```

The column USER LIBREF contains the libref that is specified by the user in the LIBNAME statement. The user's libref is provided only to help communicate with the user, if necessary.

The following data set in library '*SAS-lib*' is active:

```
MEMBER   TYPE     STATUS   USER    OPEN MODE   USER
                                                LIBREF
--------------------------------------------------
PROD     DATA     ACTIVE   IAN(5)   INPUT     FEES
```

The column STATUS is always ACTIVE. The column OPEN MODE indicates whether the user is currently executing a SAS program step to read, update, or create the member.

_ALL_
provides summary information about each server library that is currently defined to the server. For example, this command produces this summary table, as follows:

```
proc operate serverid=share1;
    display library _all_;


                            NUMBER
 LIBREF          STATUS     OF USERS  LIBRARY NAME
-----------------------------------------------------
DATALIB         QUIESCED       1       SAS-lib
POINT           ACTIVE         6       SAS-lib
POINTS          ACTIVE         4       SAS-lib
MAIN            STOPPED        0       SAS-lib
MAPS            INACTIVE       0       SAS-lib
```

The column LIBREF in the preceding example contains the server libref for a SAS server. The server libref is the name that a server administrator assigns to the library by using one of the following:

- a LIBNAME statement specified before the PROC SERVER statement
- an ALLOCATE LIBRARY command in a PROC OPERATE statement
- an operating environment-dependent external allocation

A library that is not defined by using one of the preceding methods does not have a server libref. Therefore, administrative commands that subsequently refer to that library must use a library name that is specific to the operating environment, such as a UNIX pathname.

Alias DISP LIBRARY, D LIBRARY

### *Freeing a Library*

You use the FREE LIBRARY command to free (or release) one or more server-defined libraries. When you issue the FREE LIBRARY command, a library that is not in use is freed immediately; a library that is in use is freed after it is no longer in use. Here is the syntax for this command:

**FREE LIBRARY** *libid-1* <…*libid-n*>;

**FREE LIBRARY** _ALL_;

*libid*
specifies a libref for a predefined server library or a physical name for a server library.

_ALL_
frees all SAS libraries that were defined to the SAS/SHARE server by using an ALLOCATE LIBRARY command or the LIBNAME statement.

*Note:* To bring a library to a stopped status gradually, issue the QUIESCE LIBRARY command. To bring the library to a stopped status immediately, issue the STOP LIBRARY command. Descriptions of these commands are given later in this section.

Alias FR LIBRARY

### *Quiescing a Library*

You use the QUIESCE LIBRARY command to move a library that is defined to the current server from an active status to a stopped status. This command gradually terminates access to a library by denying new requests to use the library. It immediately stops libraries that do not currently have members open. If the library is user-defined, after all users are released, the library is stopped and is no longer defined to the server. Here is the syntax for the QUIESCE LIBRARY command:

**QUIESCE LIBRARY** *libid-1* <…*libid-n*>;

**QUIESCE LIBRARY** _ALL_;

*libid*
specifies a libref for a predefined server library or a physical name for a server library.

_ALL_
quiesces all the libraries that are defined to the server.

Alias QUI LIBRARY, Q LIBRARY

### *Restarting a Library*

You use the START LIBRARY command to restart one or more server libraries that have been stopped or quiesced. Because server libraries are available by default, this command is necessary to undo the effect of a STOP LIBRARY or a QUIESCE LIBRARY command. Here is the syntax for this command:

**START LIBRARY** *libid-1* <…*libid-n*>;

**START LIBRARY** _ALL_;

*libid*
specifies a libref for a predefined server library or a physical name for a server library.

_ALL_
restarts all server libraries that are quiesced or stopped.

Alias ST LIBRARY

If a library that was user-defined is stopped and then restarted with the START LIBRARY command, the library is no longer defined to the server.

### Stopping a Library

You use the STOP LIBRARY command to immediately terminate user access to one or more server libraries and bring the libraries to a stopped status. Here is the syntax for this command:

**STOP LIBRARY** *libid-1* <…*libid-n*>;
**STOP LIBRARY** _ALL_;

*libid*
specifies a libref for a predefined server library or a physical name for a server library.

_ALL_
stops all libraries that are defined to the server.

If users are in the process of updating a data set, updates might be lost. Subsequent attempts to access a stopped library are denied.

### Specifying a SAS Library

The SAS-library argument is specified according to operating environment. For SAS-library in the library management command examples throughout this section, see the following list of examples for specific operating environments:

*z/OS Specifics*
```
'DISK1:[AREA2.WEATHER.STATS]'
```

*UNIX Specifics*
```
'/area2/weather/stats'
```

*Windows Specifics*
```
'G:\AREA2\WEATHER\STATS'
```

## Server Management Commands

### Displaying Information about a Server

You use the DISPLAY SERVER command to display summary information about the current server. Here is the syntax for this command:

**DISPLAY SERVER**;

**DISPLAY SERVER** *server-ID* <(SAPW=*password*)>;

**DISPLAY SERVER** *server-ID* </ SAPW=*password*>;

*server-ID*
displays summary information about a specific server. For more information, see "Specifying a Server" on page 150.

SAPW= *password* | "*encoded-password*"
*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

In the following example, the DISPLAY SERVER command displays information about the server SHARE1.

```
proc operate;
    display server share1;
```

Alternatively, you can use the SERVERID= option in the PROC OPERATE statement to identify the default server. The default server is also the current server unless you use the SET SERVER command and specify a different current server. In the following example, the statements display information about the default server SHARE1, which is also the current server.

```
proc operate serverid=share1;
    display server;
```

Alias DISP SERVER, D SERVER

### Quiescing a Server

You use the QUIESCE SERVER command to move a server from an active status to a stopped status by gradually releasing libraries and users, and denying new requests to access libraries through the server. Here is the syntax for this command:

**QUIESCE SERVER**;

**QUIESCE SERVER** *server-ID* <(SAPW=*password*)>;

**QUIESCE SERVER** *server-ID* <| SAPW=*password*>;

*server-ID*
specifies the server to be terminated. If you do not specify a server ID, this command gradually terminates the current server. For information about specifying a server ID, see "Specifying a Server" on page 150.

SAPW= *password* | "*encoded-password*"
*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

A quiesced server accepts only PROC OPERATE commands. Current DATA and PROC steps can continue to access files that are already open but cannot open new files or members. After a user closes all members in a server library, the server releases the library. If this is the only server library that the user has accessed, that user is disconnected from it.

The QUIESCE SERVER command does not affect server administrators. When a QUIESCE SERVER command is issued, server administrators who are executing PROC OPERATE remain connected, and the server continues to accept connections. Because the server terminates only after all users are disconnected, a server administrator can keep the server quiesced indefinitely by remaining connected to it. While a server is quiesced, an administrator can issue the START SERVER command to change the status of the server back to active.

Alias QUI SERVER, Q SERVER

### *Setting the Current Server*

You use the SET SERVER command to specify the current server and override the server that was specified in a previous SET SERVER command. This specified server also overrides the default server that was specified in a SERVERID= option in the PROC OPERATE statement. The effect of a SET SERVER command is limited to the current execution of PROC OPERATE. Here is the syntax for this command:

**SET SERVER**;

**SET SERVER** *server-ID* <(SAPW=*password*)>;

**SET SERVER** *server-ID* <| SAPW=*password*>;

*server-ID*
specifies the current server. For information, see "Specifying a Server" on page 150.

SAPW= *password* | "*encoded-password*"
*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

If no server is specified, the SET SERVER command resets the current server to the default server that is specified in the SERVERID= option in the PROC OPERATE statement. However, if the SERVERID= option was not specified in the PROC OPERATE statement, SET SERVER retains the current server value.

In the following example, the SET SERVER command establishes the current server and displays information about that server without your having to specify a server ID.

```
proc operate;
   set server share1;
   display server;
```

In the following example, PROC OPERATE displays information about the server libraries that are identified by LIB1 and LIB2 in the DISPLAY LIBRARY command. These libraries are defined to the current server MYSHR. Next, the current server is reset to SHARE, and information is displayed about the library LIBALPHA, which is defined to the server SHARE.

```
proc operate serverid=share;
   set server myshr;
   display library lib1 lib2;
   set server;
   display library libalpha;
```

If you do not identify a server before you issue a command that acts on the current server, PROC OPERATE issues the following message:

```
ERROR: PROC OPERATE is not currently set to any
       server, so this command will be ignored.  Use
       the 'SET SERVER serverid;' command to establish
       communication with a server.
```

## Restarting a Quiesced Server

You use the START SERVER command to restart a server from a quiesced state only. If server-ID is not specified, this command starts the current server. Here is the syntax for this command:

**START SERVER**;

**START SERVER** *server-ID* <(SAPW=*password*)>;

**START SERVER** *server-ID* </ SAPW=*password*>;

*server-ID*
specifies the name of the quiesced server. For more information, see "Specifying a Server" on page 150.

SAPW= *password* | "*encoded-password*"
*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

You cannot use the START SERVER command to restart a stopped server; instead, you must execute the SERVER procedure.

Alias ST SERVER

### Stopping a Server

You use the STOP SERVER command to terminate a server immediately. If users are currently reading from or writing to members in the server library, the server closes the members and updates might be lost. The server releases the libraries held by each user and disconnects each user. Here is the syntax for this command:

**STOP SERVER**;

**STOP SERVER** *server-ID* <(SAPW=*password*)>;

**STOP SERVER** *server-ID* </ SAPW=*password*>;

*server-ID*
specifies the name of the server to be terminated. If *server-ID* is not specified, this command terminates the current server. For more information, see "Specifying a Server" on page 150.

SAPW= *password* | "*encoded-password*"
*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement .

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

*Windows Specifics*

For Windows NT and Windows 2000 operating environments, the SAS/SHARE server also responds to console STOP commands. This means that you do not have to use PROC OPERATE to terminate a SAS/SHARE server. For more information, see "SAS/SHARE Server Can Run as a Windows Service" on page 210.

*z/OS Specifics*

For z/OS operating environments, the SAS/SHARE server also responds to console STOP commands. This means that you do not have to use PROC OPERATE to terminate a SAS/SHARE server. For more information, see "SAS/SHARE Server Can Run as a Windows Service" on page 210.

# User Management Commands

## *Displaying Information about a User*

You use the DISPLAY USER command to display information about one or more users who are accessing the current server. Summary information is followed by detailed information for each user who has one or more libraries assigned. Here is the syntax for this command:

**DISPLAY USER** *user-ID-1 <…user-ID-n>*;

**DISPLAY USER** _ALL_;

*user-ID-1*

specifies one user ID. *User-ID-n* specifies multiple user IDs. For more information, see "Specifying a User" on page 151.

_ALL_

displays summary information for other administrators and for all users who were connected and have been explicitly stopped.

The next example contains two types of librefs.

*user libref*

is a user-defined name for referring to a library. It is provided only as an aid for communicating with the user, if necessary.

*server libref*

is a user-defined name that a server administrator assigns to the library by using one of these methods:

- a LIBNAME statement specified before the PROC SERVER statement
- an ALLOCATE LIBRARY command in PROC OPERATE
- an operating environment-dependent external allocation

In SAS 9.3, a library that is not defined by using one of these methods does not have a server libref. Therefore, administrative commands that subsequently refer to that library must use the library name that is specific to the operating environment (for example, a UNIX pathname).

```
DISPLAY USER 15;
           USER ID       STATUS      LIBRARIES
          ----------------------------------
          MIKE(15)      ACTIVE         2
```

The preceding summary information is followed by detailed information. For example, user MIKE(15) is accessing the following libraries:

```
USER LIBREF   SERVER LIBREF    LIBRARY NAME
 ---------------------------------------------
 USAGE         USAGE           SAS-lib
 MEM                           SAS-lib
```

User MIKE(15) is accessing the following data sets:

```
USER    SERVER    MEMBER     TYPE       OPEN MODE
 LIBREF LIBREF
 ---------------------------------------------------
 USAGE  USAGE     USAGE      CATALOG    UPDATE
 USAGE  USAGE     MODULE     DATA       INPUT
 MEM              MEMOBY     DATA       INPUT
```

In this example, MIKE(15) currently has files open in both of the libraries to which he currently has access. If user MIKE(15) had no files opened when the DISPLAY USER command was issued, only the first two parts of the output would be displayed.

The SERVER LIBREF that is missing in the preceding output indicates that the USER LIBREF MEM is not server-defined.

Alias DISP USER, D USER

## *Quiescing User Access to a Server*

You use the QUIESCE USER command to gradually terminate a user's access to a SAS/SHARE server and deny new user requests for resources. This command moves the user from an active status to a stopped status. When a quiesced user closes all files in a server library, the server releases that user's access to the library. If the user has no open files in an accessed server library, the server terminates that user's access to the library immediately. When the user has released all server libraries, the user ID is assigned a stopped status and is disconnected from the server. While a user is quiesced or stopped, the START USER command can be issued to change the user's status back to active. Here is the syntax for this command:

**QUIESCE USER** *user-ID-1* <*…user-ID-n*>;

**QUIESCE USER** _ALL_;

*user-ID-1*
specifies the ID of a user whose access to the server will be terminated. user-ID-n specifies the IDs of multiple users whose access will be terminated. For more information, see "Specifying a User" on page 151.

_ALL_
quiesces all users who are connected to the current server except the administrator who issues the command. You cannot quiesce yourself. However, you can quiesce other server administrators by name. When PROC OPERATE terminates and its server session is quiesced, that administrator is assigned a stopped status.

Alias QUI USER, Q USER

### Restarting a Quiesced or a Stopped User

You use the START USER command to restart any users who have been stopped or quiesced. Because users are allowed access to a SAS/SHARE server by default, this command is necessary only to undo the effect of a previous STOP USER or a QUIESCE USER command. Here is the syntax for this command:

**START USER** *user-ID-1* <*…user-ID-n*>;

**START USER** _ALL_;

*user-ID-1*
specifies the ID of a user whose access to the server was terminated. user-ID-n specifies the IDs of multiple users whose access to the server was terminated. When a stopped user ID is restarted, that user ID becomes unknown to the server. For information, see "Specifying a User" on page 151.

_ALL_
restarts all users who are quiesced or stopped.

Alias ST USER

### Terminating User Connections to a Server

You use the STOP USER command to immediately terminate user connections to a server. The server closes library members that the user has open, terminates the user's access to libraries that are accessed through the server, and terminates the user's communication path to the server. If the user is updating a data set when the command is issued, updates might be lost. Because users are allowed access to a SAS/SHARE server by default, this command can be useful as a security tool. Here is the syntax for this command:

**STOP USER** *user-ID-1* <*…user-ID-n*>;

**STOP USER** _ALL_;

*user-id-1*
specifies the user ID of a user whose status is currently active or quiesced, and stops users who are not currently using the server. *User-ID-n* specifies the IDs of multiple users who currently have access to the server. For more information, see "Specifying a User" on page 151.

_ALL_
stops all users who are currently active or quiesced, except the administrator who issues the command. You cannot stop yourself. However, you can stop other server administrators.

## Specifying a Server

A server-ID specifies a one- or two-level name for the server that you want to manage. If you started the server, you should already know its name.

The server name must meet the criteria for a valid SAS name, but it can also include the following special characters: dollar sign ($), at sign (@) and pound sign (#).

The operating environment and the access method that you specify for communication between a server session and a user session might also impose server-naming criteria. For complete server-naming details by operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

## Specifying a Server-Access Password

A server-access password (SAPW) is required if you are not already connected to the server and the OAPW= option in the SERVER procedure is in effect. You can specify the server-access password either as a resource ID option (SAPW=password) or as a PROC OPERATE command option (/SAPW=password).

## Specifying a User

A user ID identifies a specific user or a specific connection to a server. A user ID can be specified as a number, an identifying connection, or a case-sensitive name. A user ID name must meet the criteria for a valid SAS name, but the ID can also include the following special characters: dollar sign ($), at sign (@), and pound sign (#). The operating environment on which the client runs can also impose user-naming criteria. For details, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. The following are examples of user IDs:

maria

3

Each time a user accesses a SAS/SHARE server, the new connection is assigned a number. A user is identified in the server SAS log and in PROC OPERATE output by a combination of that number and the applicable user ID in the form *user-ID(nnnn)*.

A USER command in which you specify a user connection number is restricted to that user's specific connection. For example, if Maria accesses the same server three times, she is identified by the server as MARIA(3). To display information about that connection only, you issue the following command:

```
display user 3;
```

A USER command in which you specify a user ID operates on all current connections for that user. In addition, the QUIESCE, STOP, and START commands act on that user's future connections. For example, if Maria connects to the same server three times and accesses the server a fifth time, the following command provides information about both MARIA(3) and MARIA(5):

```
display user MARIA;
```

The following command terminates the maria(3) and maria(5) connections to the server and prevents Maria from reconnecting to the server.

```
stop user maria;
```

*Chapter 12*

# Remote SQL Pass-Through (RSPT) Facility



## Overview of the RSPT Facility

The remote SQL pass-through (RSPT) facility enables you to pass PROC SQL statements from the client to a server for processing. RSPT can be used to process SAS data or DBMS data. When you access external databases through a server, you must reference a SAS/SHARE server that has access to the database.

## Syntax: Remote SQL Pass-Through (RSPT) Facility

**Requirement:** SAS/ACCESS software

**PROC SQL;**
**CONNECT TO REMOTE** <AS *alias*> (<*options*>);
**SELECT** *object-item* FROM CONNECTION TO *server* (*dbms-select-expression*);
**EXECUTE** *(SQL-statement)* BY *server* ;
**DISCONNECT FROM** *server* ;

## PROC SQL Statement

Initiates the SQL procedure.

### Syntax

**PROC SQL**;

## CONNECT TO REMOTE Statement

Establishes a connection to a DBMS or to SAS data through a SAS server.

**Requirement:** SAS/ACCESS software

### Syntax

**CONNECT TO REMOTE** <AS *alias*> (<*options*>);

#### *Optional Arguments*

**AS *alias***
specifies an alias for the server.

**SERVER=*server-ID* | __*port-number***
specifies the name of the server. If the server is a multi-user server, *server-ID* is the name specified in the ID= option in the PROC SERVER statement. If the server is a single-user server that runs on a SAS/CONNECT server, *server-ID* is the name of the SAS/CONNECT server. In either case, *server-ID* is the same server name that is specified in the SERVER= option in a LIBNAME statement.

The TCP/IP access method enables you to specify syntax that uses two consecutive underscores with a port number, in place of a server ID that has been defined in the client TCP/IP SERVICES file.

*Note:* Do not space after the first underscore or the second underscore.

For *port-number*, specify a number that is greater than 1024 and that is not already used in the TCP/IP SERVICES file.

**SAPW=*password* | "*encoded-password*" | _PROMPT_**
specifies the password for controlling user access to a multi-user server. The password must be a valid SAS name and must be 8 characters or less in length. This password is in the UAPW= option in the PROC SERVER statement. If the UAPW= option is specified when the server is started, you must specify the SAPW= option in a CONNECT TO REMOTE statement that specifies the same server.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For information about PROC PWENCODE, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

_PROMPT_
specifies that SAS prompt the user for a valid password. A dialog box appears that prompts the user to enter a valid password. This enables you to specify the value at program execution instead of coding it into the program. Using _PROMPT_ is a way to enforce security.

**USER=*user-name* | _PROMPT_**
specifies the user ID of the accessing client on the server. The operating environment in which the client runs can also affect user-naming conventions. For details about user-naming conventions imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Here are the valid values for the USER= option:

*user-name*
For details about specifying a valid user name, see Details on page 131.

_PROMPT_
specifies that SAS prompt the user for a valid user name. Using _PROMPT_ is a way to enforce security.

**Alias:**
USERNAME
USERID
UID

**PASSWORD=*password* | "encoded-password" | _PROMPT_**
specifies the password of the accessing client on the server. The operating environment in which the client runs can also affect password-naming conventions. For details about password-naming conventions imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Here are the valid values for the PASSWORD= option:

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed. For information about PROC PWENCODE, see the PWENCODE Procedure in the *Base SAS Procedures Guide*.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

_PROMPT_
specifies that SAS prompt the user for a valid password. A dialog box appears that prompts the user to enter a valid password. This enables you to specify the value at program execution instead of coding it into the program. Using _PROMPT_ is a way to enforce security.

**Alias:**
PASSWD
PASS
PWD
PW

**DBMS=*dbms-name***
specifies the name of the server DBMS that you want to connect to. This is the same name that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS. Use this option if you want to connect to a server DBMS instead of the SAS SQL server.

**PT2DBPW=*password* | *"encoded-password"***
specifies the password for controlling pass-through access to server DBMS databases that are specified in the PT2DBPW= option in the PROC SERVER statement. If PT2DBPW= is specified when the server is started, you must specify the PT2DBPW= option in a CONNECT TO REMOTE statement that specifies the same server and also specifies the DBMS= option.

*password*
must be a valid SAS name that is 1 to 8 characters in length. The value for this option is replaced by Xs in the log. To protect this password, you should use the

security software at your site to limit access to the SAS program statements that create the server.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PROC PWENCODE statement. For details, see the *Base SAS Procedures Guide*.

Here is an example of code for obtaining an encoded password:

```
proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form **{key}encoded-password**. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

*Note:* The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

**DBMSARG=(*dbms-argument-1=value …<dbms-argument-n=value>*)**
specifies the arguments that are required by the server DBMS to establish the connection. These are the same arguments that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS.

## SELECT... FROM CONNECTION TO Statement

Specifies which data will be used.

**See:** SELECT statement in the *SAS SQL Procedure User's Guide*

### Syntax

**SELECT** *object-item* FROM CONNECTION TO *server* (*dbms-select-expression*);

#### *Required Arguments*

***object-item***
specifies one or more columns or expressions. For more information, see the SELECT statement in the *SAS SQL Procedure User's Guide*.

***server***
specifies the SAS SQL server or server DBMS where the data is stored. Here are the valid values for *server*:

REMOTE
the server that is specified in the most recent CONNECT TO REMOTE statement will be used.

*alias*
the server that you assigned the alias to (in the AS=*alias* option in the CONNECT TO REMOTE statement) will be used. Specifying *alias* is useful if you are connected to several SQL servers at the same time.

***dbms-select-expression***
specifies a SELECT expression that will be processed before the data is transmitted. For server data that is accessed through the PROC SQL view engine, *dbms-select-expression* is any valid PROC SQL SELECT statement. For a server DBMS, *dbms-select-expression* is the same SQL query that you would specify if you were connected directly to the DBMS. For more information about the PROC SQL SELECT statement, see the SELECT statement in the *SAS SQL Procedure User's Guide*.

## EXECUTE... BY Statement

Submits SQL statements for server processing.

### Syntax

**EXECUTE** *(SQL-statement)* BY *server* ;

#### *Required Arguments*

***SQL-statement***
specifies an SQL statement for server processing. *SQL-statement* can be any valid SAS SQL statement except SELECT. For a server DBMS that is accessed through a single-user server in a SAS/CONNECT session, *SQL-statement* is the same SQL statement that you would specify if you were connected directly to the DBMS.

***server***
specifies the server where the SQL statement will be processed. Here are the valid values for *server*:

REMOTE
the server that is specified in the most recent CONNECT TO REMOTE statement will be used.

*alias*
the server that you assigned the alias to (in the AS=*alias* option in the CONNECT TO REMOTE statement) will be used. Specifying *alias* is useful if you are connected to several SQL servers at the same time.

## DISCONNECT FROM Statement

Closes the connection to the server.

### Syntax

**DISCONNECT FROM** *server*;

### Required Argument

***server***
specifies which server to disconnect from. Here are the valid values for *server*:

REMOTE
the server that is specified in the most recent CONNECT TO REMOTE statement will be used.

*alias*
the server that you assigned the alias to (in the AS=*alias* option in the CONNECT TO REMOTE statement) will be used. Specifying *alias* is useful if you are connected to several SQL servers at the same time.

# Examples: Remote SQL Pass-Through (RSPT) Facility

## Example 1: Processing Data Using RSPT

Here are examples of processing data by using RSPT.

The following program joins two server data sets (RSPT through a server).

```
proc sql;
   connect to remote(server=sdcmvs.prx6xhsrv);
   select *
   from connection to remote
      (select p.idnum   label='ID Number'
              p.jobcode label='Job Code'
              s.city    label='City'
       from rmtshr.staff s,
            rmtshr.payroll p
       where s.idnum=p.idnum
       orderby jobcode);
```

## Example 2: Accessing a DBMS Server

The following program uses RSPT to connect to a remote server to read data from a DB2 table (accessing a server DBMS with RSPT).

```
proc sql;
   connect to remote(server=sdcmvs.mktserv
                     dbms=db2 dbmsarg=(ssid=db2));
   select *
   from connection to remote
      (select flight#, orig, destination, delay
      from educ.db2delay
      where delay > 11);
```

*Chapter 13*

# The LOCK Statement and Command



## Overview of the LOCK Statement and the LOCK Command

The LOCK statement and the LOCK command enable you to acquire, list, or release locks on SAS data objects, which include SAS libraries, SAS data sets, SAS catalogs, and SAS catalog entries.

*Note:* If you want to lock a SAS library or any object in it by using the LOCK statement, you must first access the library through a SAS/SHARE server.

Using a LOCK statement to lock a data object prevents other users from reading or writing to that data object. However, you can open a locked data object as many times as you want to and in any mode (for example, create, update, replace, or read) if your PROC or DATA step does not conflict with what is allowed by the engine that was used by the SAS/SHARE server to access the data object.

For more information about locking, see "Locking SAS Data Objects" on page 59.

## Dictionary

### LOCK Statement

Places an exclusive lock on a specified data object.

**See:** "Locking SAS Data Objects" on page 59

## Syntax

**LOCK**

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

### Syntax Description

***libref***
specifies the name of a SAS library that is currently accessed through a SAS/SHARE server.

***member-name***
specifies the name of a member in the library *libref* that is to be locked.

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

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

***entry-name***
specifies the name of the catalog entry to be locked.

***entry-type***
specifies 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.

**Alias:**
QUERY
SHOW

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

For more information about how and when a lock is released, see "Locking SAS Data Objects" on page 59.

## Examples

### Example 1

In the following example, the first LOCK statement acquires implicit locks on the SAS library EDUCLIB and on the SAS catalog EDUCLIB.MYCAT. It then acquires an explicit lock on the catalog entry EDUCLIB.MYCAT.CHOICE1.MENU. The second LOCK statement acquires an explicit lock on the catalog entry EDUCLIB.MYCAT.CHOICE2.MENU.

```
lock educlib.mycat.choice1.menu;
lock educlib.mycat.choice2.menu;
```

### Example 2

In the following example, the first LOCK statement that contains the argument CLEAR releases the explicit lock on the catalog entry CHOICE1.MENU, but it does not release the implicit locks because an entry in the catalog is still locked. The second LOCK

statement that contains the argument CLEAR releases the explicit lock on the catalog entry CHOICE2.MENU. Because no catalog entries remain locked, the second statement that contains the argument CLEAR also releases the implicit lock on the SAS catalog EDUCLIB.MYCAT. Because no other members of the library are locked, it also releases the implicit lock on the SAS library EDUCLIB.

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

# LOCK Command

Places an exclusive lock on a specified data object.

**See:** "Locking SAS Data Objects" on page 59

## Syntax

**LOCK**

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

### Syntax Description

***libref***
specifies the name of a SAS library that is currently accessed through a SAS/SHARE server.

***member-name***
specifies the name of a member of the library *libref* that is to be locked.

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

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

***entry-name***
specifies the name of the catalog entry to be locked.

***entry-type***
specifies 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.

**Alias:**
QUERY
SHOW

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

For more information about how and when a lock is released, see "Locking SAS Data Objects" on page 59.

## Examples

### *Example 1*

The following LOCK command locks SAS catalog entries of type CMAP. The SAS log will show that one catalog entry has already been locked.

```
lock mapslib.mapscat.euromap.cmap
```

### *Example 2*

The following LOCK command releases the lock on the catalog entry EUROMAP.CMAP. The SAS log will show that the lock on this catalog entry has been released and by whom.

```
lock mapslib.mapscat.euromap.cmap clear
```

*Chapter 14*

# SAS/SHARE Macros



## Dictionary

### SHRMACS Macro

Compiles other SAS/SHARE macros.

**Category:** None

**Note:** The SHRMACS macro must be invoked before invoking any other SAS/SHARE macro.

#### Syntax

%**SHRMACS** (*category*,<*log-info*>,<APPLSYS=*app-sys-lib-tab*…>, <SASSAML=*app-sys-lib-tab*…> );

##### *Syntax Description*

***category***
specifies the category of macros to be compiled. For details, see "Macros Generated by the SHRMACS Macro" on page 77.

Valid values: USER, SERVER, OPER, ALL

***log-info***
specifies whether descriptive information about each macro is written to the SAS log.

| | |
|---|---|
| MSG (default) | displays the SAS/SHARE macros that are generated and the function of each macro. |
| NOMSG | specifies that no information is displayed. |
| HELP | displays detailed information about the SAS/SHARE macros that are generated. This includes the syntax, a brief description, and an example of each macro that is generated by the first argument. |

**APPLSYS=*app-sys-lib-tab***
specifies which applications systems tables should be loaded. You can specify one or more tables. It is recommended that you use this argument to save initialization time. If the APPLSYS= argument is not specified, the default applications systems table is loaded. Using APPLSYS= is optional. For more information, see "The APPLSYS Macro Library" on page 79.

**SASSAML=*app-sys-lib-tab***
specifies an alternate APPLSYS macro library. If an alternate library is specified, application systems tables are loaded from it instead of from the default library. The value of SASSAML= can be a physical pathname or the string _DEFINED_, which indicates that the fileref SASSAML is already assigned to the alternate APPLSYS macro library. Using SASSAML= is optional. For more information, see "The APPLSYS Macro Library" on page 79.

## Details

The SHRMACS macro also loads the applications systems tables that associate aliases with server names and associate libraries with aliases. These tables are used to generate the server name for the PROC SERVER, PROC OPERATE, and LIBNAME statements. Based on what is specified in the first argument, the server-alias and library-alias tables can be written to the SAS log if you specify the MSG or the HELP argument in the SHRMACS macro. For example,

```
%shrmacs(user);
%shrmacs(user,help);
%shrmacs(oper,help);
%shrmacs(server,msg);
%shrmacs(all,msg);
```

Early in its execution, the SHRMACS macro invokes PROC SQL to obtain the current settings of the SAS options NOTE, SOURCE2, and LINESIZE= and saves them in macro variables that are named _NOTES_, _SRC2_, and _LS_, respectively. The original values of these options are restored after the settings have been changed by SHRMACS or other SAS/SHARE macros. You can avoid the overhead of this PROC SQL step by explicitly setting the macro variables to the values that you want. For example:

```
%let _notes_=notes;
%let _src2_=nosource2;
%let _ls_=70;
```

# LIBDEF Macro

Generates a LIBNAME statement.

**Category:** User

## Syntax

**%LIBDEF** (*libref*,<*SAS-library*> ,<READONLY> ,<RETRY> ,<ENGINE=*local-engine*> ,<RENGINE=*r*

### *Syntax Description*

**READONLY**
specifies that the server library can be accessed in read-only mode.

**RETRY**
If a LIBNAME statement that specifies the SERVER= option fails, %LIBDEF generates a LIBNAME statement without the SERVER= option.

**ENGINE=*local-engine***
specifies the local engine to be used in the user's session to access the server library. Omit this parameter unless you need to override the default engine.

**Default:** REMOTE

**RENGINE=*remote-engine***
specifies the remote engine to be used in the server session to access the library. SAS chooses an appropriate engine. Omit this parameter unless you need to override the engine that is chosen by SAS.

**Default:** No default

**RMTVIEW=*remote-engine-RMTVIEW=-option***
specifies the value of the REMOTE engine's RMTVIEW= option in the LIBNAME statement that is generated by the LIBDEF macro. You can use the RMTVIEW parameter to override the default value of the RMTVIEW option for a specific library. This parameter overrides the RMTVIEW= parameter in the SERVINFO and SERVLIB macros.

**SLIBREF=*server-library***
specifies the libref that references the specified library in the server session. If this parameter is specified, the generated LIBNAME statement will include the SLIBREF= option. This parameter overrides any value that is specified for the *SAS-data-library* argument in the LIBDEF macro statement.

***uapw***
specifies that a user can access the server only by supplying a password.

**APPLSYS=*appl-sys-lib-table***
specifies a new application system library table to be loaded. This argument is ignored if the table has already been specified in the SHRMACS call or in a previous LIBDEF call.

## Details

The LIBDEF macro generates a LIBNAME statement to define a SAS library that will be accessed locally or through a server. The server administrator can specify a physical name or a reserved libref in the APPLSYS macro library. If a physical name is specified,

the server administrator must specify that name as the second argument in this macro; the physical name should not be enclosed in quotation marks. If a libref is specified, the second argument (the physical name) is omitted.

If you specify a physical name that is not specified in the APPLSYS macro library, a LIBNAME statement is generated without the SERVER= option.

After executing the LIBDEF macro, the automatic macro variable SYSLIBRC contains the return code from the LIBNAME statement. For more information, see the section about Chapter 14, "Automatic Macro Variables," in *SAS Macro Language: Reference* in *SAS Macro Language: Reference*.

### Example

Here are three examples using the LIBDEF macro:

```
%libdef(mylib,SAS-data-library,applsys=qa);
%libdef(perm,SAS-data-library,readonly,retry,myuserpw);
%libdef(datalib);
```

## LISTLIB Macro

Lists the current library-alias table.

**Categories:** Operator
Server
User

### Syntax

**%LISTLIB** <FULL> ;

#### *Syntax Description*

**FULL**
an optional argument that writes the values that are specified for the SERVLIB macro parameters to the SAS log. For more information about using the SERVLIB macro parameters, see "Associating SAS Libraries with Server Aliases (SERVLIB)" on page 81.

### Details

The LISTLIB macro writes the library-alias table that is currently in use to the SAS log. It shows the server alias that is associated with each SAS library.

## LISTSRV Macro

Lists the server-alias table.

**Categories:** Operator
Server
User

### Syntax

**%LISTSRV**;

### Details

The LISTSRV macro writes the server-alias table to the SAS log. It shows the server ID that is associated with each defined alias.

## LISTSRVI Macro

Lists the server information table.

**Categories:** Operator
Server
User

### Syntax

**%LISTSRVI**;

### Details

The LISTSRVI macro writes the server information table to the SAS log. It shows the REMOTE engine's LIBNAME statement option RMTVIEW= and the network node name, by default.

## OPERATE Macro

Generates a PROC OPERATE statement.

**Category:** Operator

### Syntax

**%OPERATE**(*server-name*,<*oapw*>);

#### *Syntax Description*

***server-name***
specifies the server name, which can be an alias or an actual server ID. This value identifies the server to be controlled.

***oapw***
specifies the administrator password if one is required by the server. This value is mapped to the SAPW= option in the PROC OPERATE statement.

### Details

The OPERATE macro invokes the OPERATE procedure for a server that is identified by the *server-name* argument.

## SERVERID Macro

Converts a server alias to a server ID.

**Categories:** Operator
Server
User

### Syntax

**%SERVERID**(*server-alias*, <NEQ>);

#### *Syntax Description*

***server-alias***
The SERVERID macro converts the *server-alias* to an actual server ID in the SERVER= option in the SERVER and OPERATE procedures and LIBNAME statements.

**NEQ**
supplies only the server ID value (without the SERVER= option).

### Details

Additionally, the SERVERID macro generates a %LET statement for a macro variable whose name is the high-level qualifier in a two-level server name in the following form:

%**LET** *high-level-qualifier=network-node*;

### Examples

#### *Example 1*

The server name must be listed in the server information table and have a network node name associated with it, as shown in the following examples:

```
libname mylib 'SAS-data-library' %serverid(devserv);
```

The first example generates the LIBNAME statement, which supplies the SERVER=*server-ID* parameter.

#### *Example 2*

```
set server %serverid(serv1,neq);
```

The second example generates a SET SERVER statement, which supplies only the *server-ID* without the SERVER= parameter keyword.

## SERVIIDX Macro

Returns the index of the entry for the specified server in the server identification table.

**Categories:** Operator
Server

User

## Syntax

**%SERVIIDX**(*server-name*);

## Details

The SERVIIDX macro requires a server name and returns the index for that server entry in the server information table. You can use this index to access the fields in the table entry.

*Note: server-name* cannot be specified as an alias.

# SERVINFO Macro

Adds server attributes to the server information table.

**Category:** None

## Syntax

**%SERVINFO** (*two-level-server-name*,<RMTVIEW= *REMOTE-engine-RMTVIEW=-option*> , <NETNODE=*network-node-name*)> ;

### *Syntax Description*

**RMTVIEW=**
specifies a default value for the REMOTE engine's RMTVIEW= option in the LIBNAME statement. If you specify this parameter, the LIBDEF macro, by default, generates the RMTVIEW=*value* for any LIBNAME statement that specifies this server. This parameter is overridden by the RMTVIEW= parameter in the SERVLIB and LIBDEF macros.

**NETNODE=**
specifies a network node name that is represented by the high-level qualifier in a two-level server name. When a two-level server name is specified in a PROC OPERATE or a LIBNAME statement and the high-level qualifier cannot be found as a network node, the server name is treated as the name of a macro variable whose value is the node name. This substitution is useful when the node name is not a valid SAS name. If you specify NETNODE= in the server information table, the first time that it translates an alias for that server ID %SERVERID generates the following code in an application:

**%LET** *high-level-qualifier=network-node*;

*High-level-qualifier* is the high-level qualifier in the server ID that is specified in the positional parameter and *network-node* is the value of NETNODE=.

If the high-level qualifier in the server ID is also the high-level qualifier in the full network node name, you can omit it from the value of NETNODE= by using a period (.) at the beginning of the NETNODE= value. For example, if the server SHRSERV runs on HP103.DOM2.ACME.COM, you would specify the following:

```
%servinfo (hp.shrserv,netnode=hp103.dom2.acme.com);
```

The SERVERID macro generates the following:

```
%let hp=hp103.dom2.acme.com;
```

## Details

Usually, the SERVINFO macro is used in the member SERVERID in the APPLSYS macro library.

A server information table is created to contain information about the servers at your site. You can use this information in a program, or you can display it. By default, the table contains the following type of information:

- a default value for the REMOTE engine's RMTVIEW= option in the LIBNAME statement
- a network node name that is represented by a fully-qualified node name (for example, HP103.DOM2.ACME.COM).

You can also use the server information table to specify other characteristics of a server, its users, or its administrators, such as server access passwords, PROC SERVER statement options, and which release of SAS the server is running under.

# SERVLIB Macro

Adds server-library pairs to the library table.

**Category:** None

## Syntax

**%SERVLIB**(*SAS-library-name, server-name*
<RMTVIEW=*REMOTE-engine-RMTVIEW=-option*>,
<PHYSNAME=*physical-name-of-library*>,<SLIBREF=*server-libref*>,
<ENGINE=*engine-in-user-session*>, <RENGINE=*engine-in-server-session*>);

### *Syntax Description*

The SERVLIB macro adds new libraries to the library table in the APPLSYS macro library in the form *SAS-library-name*, *server-name*. How you specify the SAS library name is based on your operating environment. Here are the optional arguments to %SERVLIB:

**RMTVIEW=**
specifies the value of the REMOTE engine's RMTVIEW= option in the LIBNAME statement that is generated by the LIBDEF macro. You can use the RMTVIEW= parameter in the SERVLIB macro to specify the default value of the RMTVIEW= option for a specific library. This parameter overrides the RMTVIEW= parameter in the SERVINFO macro; but, is overridden by the RMTVIEW= parameter in the LIBDEF macro.

**PHYSNAME=**
specifies the physical name of a library. This parameter is used by the STRTSRV macro to generate a LIBNAME statement in the server session. If this parameter is used and the SLIBREF= parameter is not used, the first positional parameter in %SERVLIB is assumed to be the server libref. If both PHYSNAME= and SLIBREF= are specified, the first positional parameter is not used for generating the LIBNAME statement. Instead, the first positional parameter can be used as a

description of the library that is specified in place of the physical name when the LIBDEF macro is invoked in the application.

**SLIBREF=**
specifies the library's libref in the server session. This parameter is used by the STRTSRV macro to generate a LIBNAME statement in the server session. If this parameter is used and the PHYSNAME= parameter is not used, the first positional parameter in %SERVLIB is assumed to be the physical name of the library. If both PHYSNAME= and SLIBREF= are specified, the first positional parameter is not used for generating the LIBNAME statement. Instead, the first positional parameter can be used as a description of the library that is specified in place of the physical name when the LIBDEF macro is invoked in the application.

**ENGINE=**
specifies the engine to be used in the user session to access the library. The default is ENGINE=REMOTE. Omit this parameter unless you need to override the engine that is chosen by SAS.

**RENGINE=**
specifies the engine to be used in the server session to access the library. There is no default for this option. SAS chooses an appropriate engine. Omit this parameter unless you need to override the engine that is chosen by SAS.

## See Also

For more information about the SERVLIB macro, see "Associating SAS Libraries with Server Aliases (SERVLIB)" on page 81 and comments in the APPLSYS macro library.

# SETSRV Macro

Generates a SET SERVER statement.

**Category:** Operator

## Syntax

**%SETSRV** (*server-name*, <*oapw*>);

### *Syntax Description*

***server-name***
specifies the name of a server. The server name can be an alias or an actual server ID.

***oapw***
specifies the operator-access password to be specified in the SAPW= option in the SET statement.

## Details

The SETSRV macro generates a SET SERVER statement in an OPERATE procedure for a SAS server that is specified by using the *server-name* argument.

## SHUTSRV Macro

Stops a server.

**Category:** Operator

### Syntax

**%SHUTSRV**(*server-name*, <*oapw*>);

#### Syntax Description

***server-name***
specifies the server name. The name can be an alias or an actual server ID.

***oapw***
specifies the operator-access password to be mapped to the SAPW= option in the PROC OPERATE statement.

### Details

The SHUTSRV macro invokes the OPERATE procedure to terminate the server that is specified in the *server-name* argument.

## STRTSRV Macro

Starts a server.

**Category:** Server

### Syntax

**%STRTSRV**(*server-name*, <*options*> , <*uapw*> , <*oapw*> );

#### Syntax Description

***server-name***
specifies the server name. The name can be an alias or an actual server ID.

***options***
specifies any PROC SERVER statement options. Use blank spaces to separate options. For information about the PROC SERVER options, see Chapter 9, "The SERVER Procedure," on page 105.

***uapw***
specifies the user-access password.

***oapw***
specifies the operator-access password to be mapped to the UAPW= and OAPW= options, respectively, in the PROC SERVER statement.

## Details

The STRTSRV macro invokes the SERVER procedure. %STRTSRV generates a LIBNAME statement for each library in the current library table for the server that is being started (either directly or through an alias) for which either the SLIBREF= or the PHYSNAME= parameters in %SERVLIB have been specified. For information about %SERVLIB, see "SERVLIB Macro" on page 172. Using %STRTSRV to define a library to the server at start-up automates the change to server start-up when the library is moved to another server.

The STRTSRV macro also generates a LIBNAME statement for each library that is listed in the current library-alias table for the server that is being started.

*Chapter 15*

# SAS/SHARE General SAS System Options



## Dictionary

### COMAMID= System Option

Identifies the communications access method to connect a SAS/SHARE client and server SAS session.

**Client:** required

**Server:** required

**Valid in:** Client: Configuration file, OPTIONS statement, SAS invocation, Server: Configuration file, OPTIONS statement, SAS invocation

**Category:** Communications: Networking and encryption

**PROC OPTIONS GROUP=** COMMUNICATIONS

#### Syntax

COMAMID=*access-method-ID*

##### *Syntax Description*

***access-method-ID***
specifies the name of the communications access method that is used by a client to access a server.

#### Details

The COMAMID= system option specifies a communications access method that is used by a SAS/SHARE client to connect to a SAS/SHARE server.

For find out about the supported access methods by operating environment (for example, to connect a Windows client to a UNIX server, use the TCP/IP access method), see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

## COMAUX1= System Option

Specifies the first alternate communication access method.

**Client:** optional

**Server:** optional

**Valid in:** configuration file, SAS invocation

**Category:** Communications: Networking and encryption

**PROC OPTIONS GROUP=** COMMUNICATIONS

### Syntax

COMAUX1=*name*

### Details

The COMAUX1= option specifies the first auxiliary communication access method. For example, you can specify COMAMID=XMS and COMAUX1=TCP. These specifications indicate that the primary method of communication is cross-memory services. If this access method is unable to establish a connection, TCP/IP communication is attempted.

If the COMAUX1= option is specified in a destination (server) session, it defines additional communication support to be initialized. In an originating (user) session, it specifies that the communication access method should try to connect to the destination session if the initial COMAMID-based attempt is unsuccessful. Because only z/OS has available an alternate communications access method in Version 9 SAS, this option is likely not useful on other host types.

## TBUFSIZE= System Option

Specifies the value of the default buffer size that the server uses for transferring data.

**Client:** optional

**Server:** optional

**Valid in:** Client: configuration file, OPTIONS statement, SAS invocation, Server: OPTIONS statement

**Category:** Communications: Networking and encryption

**PROC OPTIONS GROUP=** COMMUNICATIONS

**Default:** 128K

## Syntax

TBUFSIZE=*value*

### *Syntax Description*

***value***
specifies the suggested size of a buffer that the server uses for transmitting information to or receiving information from a client.

## Details

The TBUFSIZE= option specifies the suggested size of a buffer that the server uses for transmitting information to or receiving information from a client. When this option is not specified in the PROC SERVER statement, the value of the TBUFSIZE= SAS system option, if specified, will be used.

# SHARESESSIONCNTL= System Option

In a SAS Intelligence Platform environment, specifies the number of connections between SAS clients and a SAS/SHARE server via a SAS server. There can be a single connection for all SAS clients or a separate connection for each SAS client.

**Server:** optional

**Valid in:** configuration file, SAS invocation, OPTIONS statement, System Options window

**Category:** Communications: Networking and encryption

**PROC OPTIONS GROUP=** COMMUNICATIONS

## Syntax

SHARESESSIONCNTL= SERVER | ENV

### *Syntax Description*

**SERVER**
creates a single connection between the SAS/SHARE client and the SAS/SHARE server. A typical connection occurs when a client uses a LIBNAME statement to access data from a library on the computer that the SAS/SHARE server runs on. This is the default.

**ENV**
in a SAS Intelligence Platform environment, creates a connection between each SAS client that accesses data from SAS/SHARE server library via another SAS server. An example of a SAS/SHARE server connection in the SAS Intelligence Platform environment might involve several SAS Enterprise Guide clients that request data sources from a SAS/SHARE server via a SAS Workspace Server. Each SAS Enterprise Guide client (and other SAS clients) will establish a separate connection to the SAS/SHARE server via the workspace server. Maintaining separate client connections to a SAS/SHARE server ensures security and simplifies client administration.

If ENV is not specified in a SAS Intelligence Platform environment, a single connection is maintained between the SAS server (such as the workspace server) and

the SAS/SHARE server. All SAS clients that request data from the SAS/SHARE server via a SAS server will communicate using a single connection. Not using ENV reduces security and obscures the identities of the SAS clients that request access to the SAS/SHARE server.

This option and value must be specified at the SAS server, which communicates on behalf of the SAS clients to the SAS/SHARE server.

## Details

This option is useful in a SAS Intelligence Platform environment in which the administrator wants to maintain a separate security context for each SAS client (such as a SAS Enterprise Guide client) that connects to the SAS/SHARE server.

The SHARESESSIONCNTL= option should be specified only in the SAS session of the SAS server that will connect to the SAS/SHARE server. The option is effective when it is specified in the SAS configuration file or in an OPTIONS statement in the SAS server session and when it is executed before the SAS server completes its initialization and connection to the SAS/SHARE server. The SHARESESSIONCNTL= option should not be specified while the SAS server is already connected to a SAS/SHARE server. Otherwise, this note is written to the SAS log:

```
The Remote engine is active.
The SHARESESSIONCNTL setting is referenced only if the engine is reloaded.
```

The REMOTE engine is active for the duration of the SAS client connections to the SAS/SHARE server. The REMOTE engine is unloaded after all client connections to a SAS/SHARE server are terminated. The REMOTE engine is reloaded when subsequent SAS clients connect to the SAS/SHARE server.

# VALIDMEMNAME= System Option

Specifies the rules for naming SAS data sets and data views. When used In SAS/SHARE or SAS/CONNECT, the value specified on the client takes precedence over the value set on the server.

**Valid in:** configuration file, SAS invocation, OPTIONS statement, SAS System Options window

**Category:** Files: SAS Files

**PROC OPTIONS GROUP=** SASFILES

**Applies to:** Base SAS engine and SPDS engine

**Restriction:** The VALIDMEMNAME= option is not supported by the tape engines V9TAPE, V8TAPE, V7TAPE, and V6TAPE

## Syntax

**VALIDMEMNAME**=COMPATIBLE | EXTEND

### *Syntax Description*

**COMPATIBLE**
specifies that a SAS data set name, a view name, or an item store must follow these rules:

- The length of the names can be up to 32 characters.
- Names must begin with a letter of the Latin alphabet (A–Z, a–z) or the underscore. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- Names cannot contain blanks or special characters except for the underscore.
- Names can contain mixed-case letters. SAS internally converts the member name to uppercase. You cannot, therefore, use the same member name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`customer`**, **`Customer`**, and **`CUSTOMER`** all represent the same member name. How the name is saved on disk is determined by the operating environment.

This is the default.

**Alias:** COMPAT

**EXTEND**

specifies that a SAS data set name, a SAS view name, or an item store must follow these rules:

- Names can include national characters.
- The name can include special characters, except for the characters / \ * ? " < > |: -.

  *Note:* The SPD engine does not allow '.' (the period) anywhere in the member name.
- The name must contain at least one character.
- The length of the name can be up to 32 bytes.
- Null bytes are not allowed.
- Names cannot begin with a blank or a '.' (the period).

  *Note:* The SPD engine does not allow '$' as the first character of the member name.
- Leading and trailing blanks are deleted when the member is created.
- Names can contain mixed-case letters. SAS internally converts the member name to uppercase. You cannot, therefore, use the same member name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`customer`**, **`Customer`**, and **`CUSTOMER`** all represent the same member name. How the name appears is determined by the operating environment.

**Restriction:** The windowing environment supports the extended rules in the Editor, Log, and Output windows when VALIDMEMNAME=EXTEND is set. In most SAS windows, these extended rules are not supported. For example, these rules are not supported in SAS Explorer, the VIEWTABLE window, and windows that you open using the Solutions menu.

**Requirement:** When VALIDMEMNAME=EXTEND, SAS data set names, view names, and item store names must be written as a SAS name literal. If you use either the percent sign (%) or the ampersand (&), then you must use single quotation marks in the name literal in order to avoid interaction with the SAS Macro Facility.For more information, see "SAS Name Literals" in Chapter 3 of *SAS Language Reference: Concepts*.

**Tip:** The name displays in uppercase letters.

**See:** "How Many Characters Can I Use When I Measure SAS Name Lengths in Bytes?" in Chapter 3 of *SAS Language Reference: Concepts*

**Examples:**

data "August Purchases"n;

data 'Años de empleo'n.;

**CAUTION: Throughout SAS, using the name literal syntax with SAS member names that exceed the 32-byte limit or have excessive embedded quotation marks might cause unexpected results.** The intent of the VALIDMEMNAME=EXTEND system option is to enable compatibility with other DBMS member naming conventions, such as allowing embedded blanks and national characters.

**CAUTION: Using the special character # when VALIDMEMNAME=EXTEND could cause a SAS data set to be overwritten by a generation data set.** When VALIDMEMNAME= is set to EXTEND, it is possible to name a SAS data set name that uses the naming conventions for generation data sets, which append the special character # and a three-digit number to its member name. To avoid conflict, do not name SAS data sets similar to archived SAS data sets. For example, for a data set named A, generation data sets would automatically be named A#001, A#002, and so on. If you name a SAS data set A#003, the SAS data set could be deleted by SAS in the process of adding to a generation group.

## Details

In SAS/SHARE 9.3, the value that you specify for VALIDMEMNAME on the client will be honored on the server. For example, if you set the global option VALIDMEMNAME=COMPAT in the server session before starting the server, and then connect with a 9.3 client, you will not be subject to the value of the option in the server session. The value that you set on the client (COMPAT or EXTEND) will be used in your server session.

*Note:* For 9.2 or earlier clients connecting to 9.3 servers, the value COMPAT will be used.

When VALIDMEMNAME= EXTEND valid characters that are allowed in a SAS data set name, view name, and item store name are extended to these characters:

- international characters
- characters supported by third-party databases
- characters that are commonly used in a filename

Only the DATA, VIEW, and ITEMSTOR SAS member types support the extension of characters. The other member types, such as CATALOG and PROGRAM do not support the extended characters. INDEX and AUDIT types that exist only with the associated DATA member do support extended characters.

## See Also

- Chapter 3, "Rules for Words and Names in the SAS Language," in *SAS Language Reference: Concepts*

### System Option:

- "VALIDVARNAME= System Option" in *SAS System Options: Reference*

# Part 3

# Appendixes

*Appendix 1*

# Cross-Architecture Access

## Audience for Cross-Architecture Access

The information in this section is of interest to the following SAS/SHARE users:

- end users who use SAS/SHARE to update shared data across operating environments that have different architectures
- programmers who develop SAS applications that access shared data when the applications execute in one operating environment, and the shared data resides in another operating environment that has a different architecture
- server administrators who create and maintain SAS servers that are accessible from operating environments that have different architectures

## Cross-Architecture Access: Overview

Cross-architecture access is a feature of SAS/SHARE software that enables a SAS/SHARE server session and its client sessions to execute on machines that have different architectures. For example, a server and its clients can execute on machines that have different internal representations of data such as the IBM System/390 and Intel Pentium, or Digital Equipment Alpha VMS and Hewlett-Packard Precision Architecture.

Cross-architecture access enables you to move data or applications from one type of operating environment to another. For example, a UNIX application that uses a SAS library on z/OS issues a LIBNAME statement in the same way that a z/OS application does—that is, by specifying the z/OS physical name for the SAS library and the name of the z/OS server. Using cross-architecture access, you can do the following:

- move a SAS application from z/OS to UNIX, leave the data on z/OS, and continue to access the data without changing the application
- move an application's data from UNIX to z/OS, leave the application on UNIX, and change only the LIBNAME statement that accesses the data
- duplicate the application on both operating environments and simultaneously access the data on z/OS

Cross-architecture access enables users to read and write SAS data across architecture boundaries. It enables applications that run in one type of operating environment to read DBMS data that is accessed through server-managed SAS/ACCESS views when that DBMS is available only under another operating environment. For example, a SAS session on a Sun workstation can use a SAS/ACCESS view to read the contents of a DB2 table on a machine that runs the z/OS operating environment. For more information about using views under SAS/SHARE, see “SAS Data View Programming Considerations” on page 54.

SAS/SHARE software is especially well-suited to the following types of applications:

- those that require access to a single record at a time
- those that use a WHERE clause to subset large data sets
- those that execute procedures against small data sets

An application that processes large quantities of data, especially through multiple passes, might benefit from moving a copy of the data to the computer on which it executes, or

from using SAS/CONNECT software to remotely execute SAS on the computer on which the data is stored.

SAS/SHARE 9.1 supports access to some other types of SAS files, such as SAS catalogs, when the architecture of the server machine differs from the architecture of the client machine. For details, see "Cross-Architectural Differences" on page 187 and "Cross-Architecture Restrictions and Limitations" on page 187.

# Cross-Architectural Differences

- The internal data representations are different.

  Cross-architecture access is used when the client session and the server session are running on machines that internally represent data differently due to hardware differences between two machines. For example, IBM mainframe machines represent floating-point numbers differently than computers that use Intel CPUs. The code pages that are used to represent character data also vary. For example, EBCDIC and ASCII are two major character-encoding methods.

- The C-language compilers that are used are different.

  Different operating environments and C-language compilers also cause differences in data representation due to the varied alignment requirements of aggregate data types, such as the inter-element padding in a specific C structure. Also, two compilers for the same type of CPU might implement simple data types that have different lengths.

- Operating environments are the same, but the machine architectures are different.

  It might not always be obvious when the cross-architecture feature of SAS/SHARE is required. z/OS to CMS access is not cross-architecture because the underlying representation of data in the two operating environments is the same. However, sharing data between OpenVMS for VAX and OpenVMS for AXP uses cross-architecture access because data is represented differently on the Digital VAX and Alpha AXP architectures, even when the same operating environment is used. For complete details about architectural compatibility, see "Identical Architectural Groups" on page 194.

*Note:* Although neither CMS nor OpenVMS VAX are supported in later versions of SAS, they are included in the preceding example for backward compatibility.

# Cross-Architecture Restrictions and Limitations

## *Cross-Architecture Catalog Access in the Client SAS Session*

In cross-architecture catalog access, clients have read-only access to the SAS catalogs, but they cannot create, replace, or update SAS catalogs.

The client application specifies a catalog by using the libref and catalog names. This causes the entire catalog to be stored in a transport file on the server and imported by the client. The catalog is stored in the client's WORK library under an automatically generated name, such as the following:

```
WORK._SASXHST_00000000000001023144088.CATALOG
```

A maximum of three imported catalogs at a time might be stored in the client's WORK library. A subsequent read of the original catalog by using the libref and catalog name might read from the imported copy of the catalog in the WORK library, depending on whether the catalog on the server has been modified since the last read by the cross-architecture client. If more than three cross-architecture catalogs are imported, the least recently used catalog is deleted to make room for the most recently used catalog.

The CONTENTS and the CATALOG procedures can be used to examine the imported catalogs in the WORK library. The DATASETS procedure can be used to delete these catalogs.

### Cross-Architecture Catalog Access in the SAS/SHARE Server

By default, the SAS/SHARE server uses the library WORK to cache the transport files on the server in the SAS library SASCATCA. However, you can use the SERVER procedure ALLOCATE LIBRARY command to assign the cache to a different SAS data library. The files in this library are stored in the transport format of SAS catalogs. For example:

```
proc server id=shr1;
allocate library mycache '/catalog/cache' libtype=catcache;
run;
```

You might want to assign the cache to a different SAS library in order to accomplish the following:

- minimize activity in the library WORK on the server so that other scratch uses of that library (such as temporary sort files) are not competing for space.
- dedicate a specific amount of space to the cache. The allotted space might be especially large or limited to a modest size by site considerations.
- save the SAS catalog files in transport format when the SAS/SHARE server is stopped, so that you can avoid having the server repeatedly translate catalogs into transport format.

Although the server caches a maximum of three transport files, an administrator can change the default. The ALLOCATE LIBRARY command, issued in the SERVER procedure, has a CATCACHELIMIT option in which you can specify the number of catalogs that are stored in the library SASCATCA. For details, see ALLOCATE LIBRARY Statement on page 119.

### Concatenating Cross-Architecture Catalogs

You can concatenate catalogs to minimize changes to an application and to increase efficiency when accessing cross-architecture catalogs. Set up a catalog concatenation in the client session that uses the same librefs and catalog names that the application already uses.

Two new catalogs are used in a concatenation:

- One catalog, which is accessed through the SAS/SHARE server, contains catalog entries that change frequently.
- Another catalog contains the entries that do not change frequently. These entries can be built for and distributed to each client. Relocating catalog entries that do not change frequently on each client maintains a low overhead.

Using catalog concatenation in the server session is not recommended, because a client that reads an entry in the concatenation will retrieve the entire concatenation as a single

catalog, which could be large. Large catalogs can take a long time to read or retrieve entries from.

Catalog concatenation on the server can be an advantage if you have many small catalogs that will be accessed simultaneously. Having all of the catalogs concatenated under one name enables the client to keep that catalog in the library WORK. Having a single concatenated catalog reduces the possibility of thrashing, which can result if the client needs more than three catalogs in the library WORK. Thrashing occurs when catalogs are deleted and then retrieved.

### Other SAS File Access

Direct access to PROC SQL views is provided, but SAS view files (type VIEW) cannot be directly accessed across architectures. A DATA step or SAS/ACCESS view can be read cross-architecture as long as it is interpreted in the server session. In this case, you should not specify the RMTVIEW=NO option in the client LIBNAME statement because that option requests interpretation in the client SAS session and requires the SAS view file itself to be transferred to the client session. For more information, see "SAS Data View Programming Considerations" on page 54.

Access descriptor files (type ACCESS) cannot be accessed across architectures. Access descriptors are special files that are produced and used by SAS/ACCESS products to describe data in other vendors' databases, such as DB/2 or Oracle. Although cross-architecture interpretation of a SAS/ACCESS view is supported, direct access to the descriptor file is not. Therefore, you cannot use PROC ACCESS to create a SAS access descriptor file on a computer that has a different architecture.

SAS files of type PROGRAM cannot be accessed across architectures. These files contain compiled DATA step code. You cannot execute such a DATA step in your local SAS session by using the PGM= option in the DATA step, nor can you write a DATA step PROGRAM entry to a cross-architecture server. A DATA step PROGRAM entry can be executed in a cross-architecture server session if it is referenced by a DATA step view that is interpreted there.

### Short Numerics and Mixed-Type Variables

In order for SAS data sets to be accessed across architectures, they should not include two-byte numeric variables. This length is allowed on IBM mainframe machines, but other operating environments that SAS runs on have a minimum numeric variable length of three. As a result, a data set that contains a two-byte numeric cannot be accessed across architectures from other types of operating environments.

With clever programming in the DATA step, it is possible to store numeric values in character variables and character values in numeric variables. However, you should not create data sets this way if you want to access them across architectures. SAS/SHARE performs appropriate character translation of character variables and numeric translation of numeric variables when crossing architectures. However, storing numeric values in character variables and character values in numeric variables will not be preserved. SAS/SHARE has no means to detect such usage.

# Implications of Data Translation

## Translation of Data at the Client and the Server

In SAS/SHARE, translation of numeric variables occurs when the server machine and the client machine represent floating-point numbers differently. For character variables, translation occurs when their character representations differ. Values are dynamically translated directly from the source representation to the target representation; they do not pass through transport format. Translation occurs both when data flows from the server to the client and when it flows from the client to the server. Therefore, data that flows across architectures from a server to a client and is then sent back to the server is translated twice.

For all operating environments that SAS/SHARE runs on, the REMOTE engine performs all data translations that are necessary in order to converse with the server. The REMOTE engine translates outgoing data to the server format, and translates incoming data from the server to its own format. The administrative procedure, PROC OPERATE, works in the same way.

*Note:* For all SAS/SHARE clients other than SAS sessions, such as the SAS ODBC driver, data translation occurs on the server, not on the client.

## Translation of Floating-Point Numbers between Computers

### Loss of Numeric Precision and Magnitude

If you move SAS data between a client and a server session that run on computers that have different architectures, numeric precision or magnitude can be lost. Precision can be lost when the data value in the source representation contains more significant digits than the target representation can store. A loss of magnitude results when data values exceed the range of values that an operating environment can store.

For complete details about how SAS stores numeric values, see *SAS Language Reference: Concepts*.

### Avoiding Loss of Precision

To avoid loss of precision, do not store numeric values in short variables. Instead, store numeric values using longer numeric variables (up to 8 bytes) according to the number of significant digits that the target representation can store.

### Significance of Loss of Magnitude

When you lose magnitude, SAS produces the following warning:

```
WARNING:  The magnitude of at least one numeric value
was decreased to the maximum the target representation allows,
due to representation conversion.
```

A loss of magnitude is unlikely in many applications, but if you have data with extremely large values or extremely small fractions, you might experience a loss of magnitude during cross-architecture access. When you lose magnitude, SAS changes the values that are out of range to the maximum or minimum value that the operating environment can represent.

***Table A1.1*** *Approximate Value Ranges by Operating Environment*

| Operating Environment | Minimum Value | Maximum Value |
|---|---|---|
| UNIX | 2.3E-308 | 1.8E+308 |
| Windows | 2.3E-308 | 1.8E+308 |
| z/OS | 5.4E-79 | 7.2E+75 |

### *Example*

You create a data set under UNIX that contains the value **`8.93323E+105`**. If you copy the file to a z/OS operating environment, magnitude is lost and the value changes to **`7.23701E+75`**, which is the maximum value that z/OS can represent.

## *Character-Translation Tables*

*Note:* The use of translation tables is relevant only when you use the following:

- SAS 8 and SAS 9.2 and later cross-release access
- thin client and SAS 9.2 and later server access

The tables that are used for character translation in SAS/SHARE are stored in SAS catalog entries of type TRANTAB. Each of these catalog entries contains two translation tables. The first table is for import translation, and the second table is for export conversion. For example, the EBCDIC/ASCII-OEM translation entry under z/OS contains an import table for ASCII-OEM to EBCDIC translation and an export table for EBCDIC to ASCII-OEM translation.

***Table A1.2*** *Translation Tables and Catalog Entry Names*

| Translation Table Set | Catalog Entry Name |
|---|---|
| EBCDIC/ASCII-ISO | _0000030 |
| EBCDIC/ASCII-ANSI | _0000060 |
| EBCDIC/ASCII-OEM | _00000A0 |
| EBCDIC/ASCII-MAC | _0000120 |
| ASCII-ISO/ASCII-ANSI | _0000050 |
| ASCII-ISO/ASCII-OEM | _0000090 |
| ASCII-ISO/ASCII-MAC | _0000110 |
| ASCII-ANSI/ASCII-OEM | _00000C0 |
| ASCII-ANSI/ASCII-MAC | _0000140 |
| ASCII-OEM/ASCII-MAC | _0000180 |

Character-translation catalog entries are stored in the SASUSER.PROFILE and SASHELP.HOST catalogs. The translation process locates a specific translation entry by first searching the SASUSER.PROFILE catalog and then searching the SASHELP.HOST catalog.

The client's execution of the REMOTE engine and the client's translation tables are responsible for all data translations that occur between a SAS/SHARE client and a server. However, the SAS/SHARE server is responsible for the data translations that occur between that server and all SAS/SHARE clients other than SAS sessions.

SAS site administrators can use the TRANTAB procedure to replace or update the translation tables. For details, see the Chapter 17, "TRANTAB Procedure" in *SAS National Language Support (NLS): Reference Guide* and "TRANTAB= System Option" in *SAS National Language Support (NLS): Reference Guide*.

***CAUTION:***

**Do not attempt to update a translation table in a client session while accessing the SAS/SHARE server that the translation table will be applied against.** You cannot ensure that the new version of the table will be used for subsequent translations.

## Data Translation Considerations

Data translation in SAS/SHARE has some implications that users need to consider. For example, suppose you assign two SAS libraries, ROO and ZOO, through a server on an operating environment that has an architecture that is different from your machine. You copy the data sets that are contained in ROO to ZOO:

```
proc copy in=roo out=zoo mt=data;
run;
```

The contents of the copied data sets in ZOO are not guaranteed to be identical to the contents of the original data sets in ROO because the data sets in ZOO have been translated twice. First, the data is translated from server representation to client representation, then from client representation back to server representation.

As another example, suppose you are using the FSEDIT procedure to edit a data set across architectures. You enter a DUP command and then modify the variable X before saving the new record. You might find that, other than the value of the variable X, the new record is not identical to the old record. The original values of the duplicated record have been translated twice, from server-machine format to client-machine format and back to server-machine format. The new value that was entered for the variable X has been translated only one time, from user-machine format to server-machine format.

*Note:* When editing or updating a data set across architectures by using the FSEDIT procedure, the FSVIEW procedure, or the MODIFY statement in the DATA step, any variables that are not updated in an updated observation are exempt from translation and will be unaltered.

## Problems Parsing Numeric Data in a Cross-Architecture Environment

SAS code (keywords and data) are submitted for execution as text. During parsing, numeric data is converted to the floating-point representation that is used by the operating environment and computer. For UNIX and Windows, numeric data is represented in IEEE floating-point format. For z/OS, numeric data is represented in IBM floating-point format.

When using SAS/CONNECT and Remote Library Services to process numeric data in a cross-architecture client/server session, you might get unexpected, but logical, results.

Consider a scenario in which a SAS/CONNECT client session under Windows accesses a server session under z/OS. In this code example, the value -6.14, which is stored in a data set, is used in a WHERE clause. This number cannot be represented exactly in either IEEE or IBM floating-point formats because -6.14 is a repeating decimal, in binary. Regardless of whether -6.14 is stored under Windows or z/OS, its representation is imprecise.

```
 /* Start a server session on z/OS */
options comamid=tcp remote=zos;
filename rlink "c:\sas\v9\connect\saslink\tcptso.scr";
signon;

/* The following code is remotely submitted to the server session  */
/* under z/OS. The text is passed to z/OS and is parsed on z/OS. */
/* When the text "-6.14" is parsed on z/OS, the resulting value  */
/* is stored on z/OS in IBM floating-point format.               */
rsubmit; 1
data sasuser.test_4;
   format RUE_H comma18.2;
   RUE_H= -6.14; output;
   RUE_H= -6.14; output;
   RUE_H= -6.14; output;
   RUE_H= -6.14; output;
   run;
endrsubmit;

/* The following code assigns a server library on z/OS.         */

libname test1 server=sdczos slibref=sasuser;

/* The following statements are parsed in the client session    */
/* on Windows, and the results are stored in IEEE format.       */
/* in the client session in IEEE format.                        */

/* You might expect the following WHERE clause to return zero   */
/* records because the expression resolves to 0.                */
/* However, this code execution returns 4.                      */
proc sql; 2
   create table xx2 as
   select RUH_H
   from test1.test_4
   where RUE_H < -6.14 3;
quit;
```

1 Parsing the string "-6.14" on z/OS produces a binary IBM representation of -6.14.

2 Parsing the string "-6.14" on Windows produces a binary IEEE representation of -6.14.

3 The WHERE clause, which contains the Windows binary IEEE value, is sent to z/OS via RLS, the IEEE representation of -6.14 is converted to its closest binary IBM equivalent. However, when an IEEE binary representation of -6.14 is converted into IBM format, the result is different from the IBM binary value that was obtained by parsing and converting the string on z/OS.

The clause **where RUE_H < -6.14** finds four observations because the binary IBM value that is obtained by converting the binary IEEE value is slightly smaller than the binary IBM value that was stored in the data set when "-6.14" was originally parsed. Although you might expect the WHERE clause to return no observations, it returns four observations because of the lack of precision that occurs when converting data across operating environments.

To avoid cross-architecture problems, you could change the code so that PROC SQL is remotely submitted to execute in the server session on z/OS. The text value, "-6.14", in the WHERE clause would be parsed and converted on z/OS and would result in the same binary representation that was used in the original data set, which was also parsed and converted in the server session on z/OS. Therefore, the WHERE clause would return no observations, as expected.

```
rsubmit;
proc sql;
   create table xx2 as
   select RUH_H
   from test1.test_4
   where RUE_H < -6.14 ;
quit;
endrsubmit;
```

When using SAS/CONNECT and RLS to process floating-point numbers in a cross-architecture environment, ensure that all statements are parsed in the same session, either the client or the server.

# Identical Architectural Groups

## Overview of Identical Data Representation Groups

The following sections contain lists of cross-version architecture groups for SAS 8 and SAS 9.3. These architectural groups are valid only for access between the following:

- a SAS 8 client (or server) and a SAS 9.3 server (or client)
- a SAS 8 client and a SAS 8 server.

The lists are grouped on the basis of identical data representation. No data translation is required in communications between any two operating environments in the same group. When two operating environments that are not in the same group communicate, translation of at least one data type is required, and cross-architecture restrictions and limitations are applied. For details, see "Cross-Architecture Restrictions and Limitations" on page 187.

*Note:* For backward compatibility, the following lists include architectures and operating environments that were supported in earlier versions of SAS but are not supported in SAS 9 and later.

## IBM System/390 Architecture Operating Environments

- CMS
- z/OS

### *UNIX RISC Operating Environments*

- AIX
- HP-UX
- Solaris 2
- SGI

### *UNIX 64-bit Operating Environments*

- AIX 64
- HP 64
- HP-UX/Itanium
- Solaris 64

### *Windows 32-bit Operating Environments*

- Windows Vista
- Windows XP
- Windows NT
- Windows 2000
- Windows 98
- Windows 95

### *OpenVMS 64-bit Operating Environments*

- OpenVMS Alpha 64
- OpenVMS Intel 64

### *UNIX 64–bit Little Endian Operating Environments*

- OSF Alpha/Compaq Tru 64 UNIX
- Red Hat Linux/Itanium
- Red Hat Linux/Intel 64
- Solaris 64/Intel 64

### *Intel ABI+ Operating Environments*

- RedHat Linux/Intel
- IABI+ hosts

### Incompatible Operating Environments

Data representation in the following operating environments is incompatible with all other operating environments and each other:

- OpenVMS Alpha
- OpenVMS VAX
- OS/2
- Windows/Itanium

*Note:* None of the SAS 9.3 supported operating environments have identical non-character representations.

## Numeric Architectural Groups

### Overview of Numeric Architectural Groups

The following sections contain lists of cross-version architecture groups for SAS 8 and SAS 9.3. These architectural groups are valid only for access between the following:

- a SAS 8 client (or server) and a SAS 9.3 server (or client)
- a SAS 8 client and a SAS 8 server.

The following lists show the SAS 8 and SAS 9.3 operating environments that SAS/SHARE software runs on. The lists are grouped on the basis of similar numerical representation. No numeric (floating-point) translation is required in communications between any two operating environments in the same group. When two operating environments in different groups communicate, numeric translation is required, and restrictions and limitations are applied. For details, see "Cross-Architecture Restrictions and Limitations" on page 187 .

*Note:* For backward compatibility, the following lists include architectures and operating environments that were supported in earlier versions of SAS but are not supported in SAS 9 and later.

### Version 8 Numeric Architecture Groups

#### IBM System/390 Format Operating Environments

- CMS
- z/OS

#### IEEE Format Operating Environments

- AIX
- AIX 64
- Compaq Tru64 UNIX (formerly Compaq's DIGITAL UNIX)
- HP-UX

- HP 64
- HP-UX/Itanium
- Intel ABI+ hosts
- OpenVMS Alpha
- OpenVMS Alpha 64
- OS/2
- RedHat Linux/Intel
- SGI
- Solaris 2
- Solaris 64
- Windows XP
- Windows NT
- Windows 2000
- Windows 98
- Windows 95
- Windows/Itanium

Translation of numeric data is necessary between some of these operating environments because byte-ordering or alignment requirements are different. However, because they all use the same number of exponent and mantissa bits, loss of precision or range does not occur.

#### *OpenVMS VAX Operating Environment*

OpenVMS VAX uses a unique numeric representation.

### *SAS 9.3 Numeric Architecture Groups*

#### *IBM System/390 Format Operating Environment*

- z/OS

#### *IEEE Format Operating Environments*

- AIX 64
- Compaq Tru64 UNIX (formerly Compaq's DIGITAL UNIX)
- HP 64
- HP-UX/Itanium
- OSF Alpha
- RedHat Linux/Intel
- RedHat Linux/Itanium
- RedHat Linux/Intel 64
- Solaris 64
- Solaris 64/Intel 64

- Windows XP
- Windows NT
- Windows 2000
- Windows/Itanium

Translation of numeric data is necessary between some of these operating environments because byte-ordering or alignment requirements are different. However, because they all use the same number of exponent and mantissa bits, loss of precision or range does not occur.

# Character Architectural Groups

## Overview of Character Architectural Groups

The following sections contain lists of cross-version architecture groups for SAS 8 and SAS 9.3. These architectural groups are valid only for access between the following:

- a SAS 8 client (or server) and a SAS 9.3 server (or client)
- a SAS 8 client and a SAS 8 server.

The lists are grouped on the basis of similar character representation. No character translation is required when communicating between any two operating environments in the same group. When two operating environments in different groups communicate, character translation is required and restrictions and limitations are applied. For details, see "Cross-Architecture Restrictions and Limitations" on page 187.

*Note:* For backward compatibility, the following lists include architectures and operating environments that were supported in earlier versions of SAS but are not supported in SAS 9 and later.

## EBCDIC Format Operating Environments

- CMS
- z/OS

## ASCII-ISO Format Operating Environments

- AIX
- AIX 64
- Compaq Tru64 UNIX (formerly Compaq's DIGITAL UNIX)
- HP-UX
- HP-UX 64
- HP-UX/Itanium
- Intel ABI+ hosts
- OpenVMS Alpha
- OpenVMS Alpha 64

- OpenVMS VAX
- RedHat Linux/Intel
- SGI
- Solaris 2
- Solaris 64

### *ASCII-ANSI Format Operating Environments*

- Windows XP
- Windows NT
- Windows 2000
- Windows 98
- Windows 95
- Windows/Itanium

### *ASCII-OEM Format Operating Environment*

- OS/2

*Note:* Because SAS 9.3 uses execution-time session encoding, operating environments cannot be meaningfully grouped according to character representation for SAS 9.3.

*Appendix 2*

# Creating the SAS/SHARE Server Environment



## Audience for SAS/SHARE Server Start-Up

This information is designed for system administrators or server administrators who are responsible for preparing the operating environment to accommodate a SAS/SHARE server. Requirements for successful server start-up and operation vary according to operating environment type.

# All Operating Environments: Setting SAS System Performance and Logging Options

Several SAS system options can help you reduce the number of disk accesses that are needed for SAS files and, therefore, enhance system performance. The SAS system options that are listed here are valid for all operating environments. Options that are operating environment-specific are documented in the sections that follow.

From a SAS session, run PROC OPTIONS to find the default settings for SAS system options on your operating environment.

BUFNO=
specifies the number of buffers to use for SAS data sets.

BUFSIZE=
specifies the permanent buffer size for an output SAS data set.

CATCACHE=
specifies the number of SAS catalogs to keep open.

COMPRESS=
controls the compression of observations in output SAS data sets.

LOGPARM=
controls when SAS log files are opened and closed.

For details, see in *SAS System Options: Reference*.

# The Operating Environments



# OpenVMS: Creating the Server Environment

## *Overview of Tasks to Create the Server Environment*

You must perform the following tasks to create the server environment under an OpenVMS operating environment:

1. Set SAS performance options.
2. Create a command file for the server.
3. Run the command file for the server.
4. Run the SUBMIT command to create the server.

### Setting SAS System Performance and Logging Options

The following SAS system options can be used to tune server performance and logging:

BUFNO=
specifies the number of buffers to use for SAS data sets. The default is 1.

BUFSIZE=
specifies the permanent buffer size for an output SAS data set. The default is 0.

LOG=
specifies a destination to which the SAS log is written in batch mode. The default is SYS$PRINT (the default printer queue) or SYS$OUTPUT (the default output stream).

LOGPARM=
controls when SAS log files are opened and closed.

### Creating a Command File for the Server

The command file performs any necessary process setup and invokes SAS. SAS runs a program that contains any setup that is needed for the server environment and then runs the PROC SERVER statement. For details about how to write a SAS program to start a server, see "Managing a SAS/SHARE Server (Server Administrators)" on page 29 .

Use the following syntax to create a command file for a server:

```
$set noon
$!
$ SAS /ALTLOG=SYS$OUTPUT
      /ALTPRINT=SYS$OUTPUT
      /COMAMID=access-method
      sas-input-file
$!
$exit
```

ALTLOG=SYS$OUTPUT
ALTPRINT=SYS$OUTPUT
specifies the files to which SAS writes copies of the log and the procedure output, respectively. These copies of the log and the procedure output are in addition to the default .LOG and .LIS files. Specifying /ALTLOG=SYS$OUTPUT and /ALTPRINT=SYS$OUTPUT causes all SAS output from the server process to be written to the SYS$OUTPUT file, which produces a single file that contains the OpenVMS record of the process execution and the SAS record of the server execution.

How the logical name SYS$OUTPUT is defined depends on how the command file is executed. For this information, see "Executing the Command File for the Server" on page 204.

COMAMID=*access-method*
specifies the access method that the server uses to communicate with its clients. Assign TCP to the COMAMID= option.

*sas-input-file*
specifies the name of the file that contains the SAS statements to start the server. For details about writing a program to start a server, see "Managing a SAS/SHARE Server (Server Administrators)" on page 29.

### Executing the Command File for the Server

You can execute the command file for a server by using the SUBMIT command to submit a batch job. The batch job creates a detached process, which then executes the command file.

Use the SUBMIT command to start the server during start-up of your OpenVMS operating environment or start a server by executing a command.

Because of its nature, a server usually runs in a detached process. Instead of executing the RUN command directly during system start-up or at other times, you should execute the RUN command in a batch command file that you submit with the SUBMIT/USER= command. This ensures that the server is created with appropriate privileges and file access authority. The SUBMIT/USER= command requires the CMKRNL privilege.

Here is the syntax of the SUBMIT command:

$ SUBMIT/USER=*user-name batch-filename*

*user-name*
specifies the name of the user that executes the batch job that creates the process in which the server runs.

*batch-filename*
specifies the batch job to be executed. The purpose of the batch job is to create a detached process in which the server executes. Therefore, this batch job usually consists of one RUN command. For example:

```
$ RUN /DETACHED              -
      /AUTHORIZE             -

      /INPUT=command-input-file    -
      /OUTPUT=command-output-file  -
      /ERROR=error-file    -
      /PROCESS_NAME=process-name   -
      /SYS$SYSTEM:LOGINOUT.EXE
```

*command-input-file*
specifies the name of the file that contains the commands that are executed in the detached process. For details about the contents of this file, see "Creating a Command File for the Server" on page 203.

*Note:* This file must also contain device or directory specifications. If the file does not contain these specifications, then the detached process might fail.

*output-file*
specifies the name of the file to which the record of the execution of the detached process is written. This file should be accessible to any administrator of the server and to developers of applications that use the server. This file contains any information that is written to SYS$OUTPUT.

*Note:* This file must also contain device or directory specifications. If the file does not contain these specifications, then the detached process might fail.

*error-file*
specifies the file to which OpenVMS errors are written. This should be accessible to any administrator of the server and to developers of applications that use the server. This file contains information that is written to SYS$ERROR.

*Note:* This file must also contain device or directory specifications. If the file does not contain these specifications, then the detached process might fail.

*process-name*
specifies a descriptive name of the detached process in which the server executes. This value can be the same as the server name that you specify for the SERVERID= option in the PROC SERVER statement.

# z/OS: Creating the Server Environment

## Starting a Server Using a Started Task

You can invoke SAS from a TSO session, a batch job, or a started task. However, it is best to use a started task to invoke SAS in order to run the PROC SERVER statement.

*Note:* If you use the XMS access method, do not invoke SAS and create the server in a batch environment. Doing this might drain the batch initiator when the server execution ends. The address space (ASID) would not be usable.

1. To start the server, create a cataloged started task procedure that contains the JCL, as follows:

***Example Code A2.1*** *Example JCL in the z/OS Started Task Procedure Library*

```
//SHRSTART  PROC ENTRY=entry,ID=id,SERVOPT=' ',UAPW=,OAPW=,OPTIONS=
//SAS       EXEC PGM=&ENTRY,DYNAMNBR=50,REGION=40M,
//          PARM='IS="%SHRMACS(SERVER);%STRTSRV(&ID',
//          '%STR(&SERVOPT),%UAPW,%OAPW",&options')
//STEPLIB   DD DISP=SHR,DSN=&prodfix.LIBRARY
//CONFIG    DD DISP=SHR,DSN=&prodfix.CNTL(BATW0)
//          DD DISP=SHR,DSN=&prodfix.CNTL(SRVCNFG)
//SASAUTOS  DD DISP=SHR,DSN=&prodfix.AUTOLIB
//SASHELP   DD DISP=SHR,DSN=&prodfix.SASHELP
//SASMSG    DD DISP=SHR,DSN=&prodfix.SASMSG
//WORK      DD UNIT=SYSDA,SPACE=(6144,(500,200),,,ROUND)
//SASLOG    DD SYSOUT=*,DCB=(BLKSIZE=141,LRECL=137,RECFM=VBA)
//SASCLOG   DD SYSOUT=*
//SASSNAP   DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SYSIN     DD DUMMY
```

*entry*
is the value that is specified for the ENTRY= parameter in the cataloged procedure that is used to invoke SAS from a batch job. This procedure was created when Base SAS was installed.

ID=*id*
is the server name (default or otherwise) that is passed to the PROC statement in the STRTSRV macro.

SERVOPT=' '
can be any valid option in the PROC SERVER statement. For information about PROC SERVER options, see Chapter 9, "The SERVER Procedure," on page 105.

UAPW=*uapw*
is the user-access password for the server.

OAPW=*oapw*
is the operator (or server administrator) password for the server.

2. Notice that the PARM= parameter uses the macro STRTSRV to start a server. %STRTSRV is a standard SAS/SHARE autocall macro. For more information, see "Using Macros for Server Library Access" on page 75 and "STRTSRV Macro" on page 174. Alternatively, you can use the SERVER macro to start a server. %SERVER executes faster than %STRTSRV. For more information, see Chapter 9, "The SERVER Procedure," on page 105.

   To use %SERVER instead of %STRTSRV in the PARM= parameter, change the EXEC statement as follows:

   ```
   //   PARM='IS="%SERVER(&ID,&SERVOPT,&UAPW,&OAPW)"'
   ```

3. After the member SERVER that contains this JCL has been created, the console operator issues the following command to create the server as a started task:

   ```
   START SHRSTART
   ```

   The cataloged procedure name for the SAS/SHARE server is SHRSTART.

4. When this command executes, the procedure passes the appropriate parameters to the SAS macro, STRTSRV or SERVER, which invokes the PROC SERVER statement.

5. To create a new server (one whose name is different from the ID= parameter in the JCL), enter the following:

   ```
   START SHRSTART,ID=server-ID
   ```

6. To enter the default PROC SERVER options that are indicated in the SERVOPT= parameter in the JCL, enter the following:

   ```
   START SHRSTART,SERVOPT='options'
   ```

7. To override the user- and operator-access passwords in the START command and to override those that are specified in the UAPW= and OAPW= parameters in the JCL, enter the following:

   ```
   START SHRSTART,UAPW=uapw,OAPW=oapw
   ```

8. To enter all of these specifications in one START command and to override those in the JCL, enter the following:

   ```
   START SHRSTART,ID=server-ID,SERVOPT='options'
                UAPW=uapw,OAPW=oapw
   ```

## Starting a Server Automatically

### Using the Static Program Method

To use the static program method, store a SAS program that contains PROC SERVER in an external file. For information about writing a SAS program to start a server, see "Managing a SAS/SHARE Server (Server Administrators)" on page 29. Invoke SAS by specifying the program as the primary input data stream. To use the program in the started task, locate the following line in the previous JCL code example:

```
//SYSIN DD  DUMMY
```

Change it to read as follows:

```
//SYSIN DD DSN=data-set-name,DISP=SHR
```

This method creates a server in the same way each time the program runs.

### Using the Macro Method

Although it is recommended that you use the STRTSRV macro from the SAS macro autocall library (see "SAS/SHARE Macros for Server Access" on page 75), you can also create and use the SERVER macro, which executes faster than %STRTSRV.

Before you can use the SERVER macro, you must create a member named SERVER in the SAS macro autocall library and add the following statements:

```
%MACRO SERVER(id,servopt,uapw,oapw);
%*****************************************;
%* This macro invokes PROC SERVER to create *;
%* a server with the specified ID.          *;
%*****************************************;
  PROC SERVER ID=&id &servopt
     %if (&uapw NE) %then
          %do;
             UAPW=&uapw
          %end;
     %if (&oapw NE) %then
          %do;
             OAPW=&oapw
          %end;
     ;
  run;
  endsas;
%MEND;
```

## Setting SAS System Performance and Logging Options

The following options affect the operation of the server.

Default values for these options are set in the SAS/SHARE server configuration file.

BUFNO=
: specifies the number of buffers to use for SAS data sets. The default value is 1.

LOGPARM=
: controls when SAS log files are opened and closed.

LOG=
: specifies a file to which the SAS log is written when executing SAS programs outside the windowing environment.

MEMLEAVE=
: specifies the amount of memory that is reserved for cleanup in the event of an abnormal termination of SAS.

MEMSIZE=
: specifies the total amount of memory that SAS can use.

STAE
: prevents a system abend exit when recoverable errors occur. STAE is the default.

SVC0SVC=
SVC0R15=
: The SAS SVC Routine 0 is required for SAS/SHARE. You must specify the SAS system options SVC0SVC= and SVC0R15= to accurately reflect the way the SVC was installed. Get this information from the person who installed Base SAS.

SYNCHIO
: specifies whether synchronous IO is enabled.

***CAUTION:***

**Do not specify the SYNCHIO option.** The SYNCHIO option prevents SAS/SHARE servers from working properly. NOSYNCHIO is the default.

VMCTLISA
specifies the size of the initial storage allocation (ISA) for SAS memory management and control blocks.

VMPAISA
specifies the size of the ISA for permanent memory above the 16-Mb line.

VMPAOSA
specifies the size of the overflow storage allocation (OSA) for permanent memory above the 16-Mb line.

VMTAISA
specifies the size of the ISA for temporary memory above the 16-Mb line.

VMTAOSA
specifies the size of the OSA for temporary memory above the 16-Mb line.

VMPBISA
specifies the size of the ISA for permanent memory below the 16-Mb line.

VMPBOSA
specifies the size of the OSA for permanent memory below the 16-Mb line.

VMTBISA
specifies the size of the ISA for temporary memory below the 16-Mb line.

VMTBOSA
specifies the size of the OSA for temporary memory below the 16-Mb line.

For details, see information about system options in the *SAS Companion for z/OS*.

# UNIX: Creating the Server Environment

## *Assigning the Server a User Account*

Rather than run the server process under root, assign a specific user account to the server process. Identify all SAS libraries that will be shared. Consider the appropriate file permissions that should be granted based on both the server and user account access. At minimum, the server account will need read and write permissions to the SAS libraries that it shares. By default, the processes that are initiated from **`/etc/inittab`** and **`/etc/rc`** will start under root ownership.

## *Starting a Server Manually*

You can invoke SAS and start a server manually by using a UNIX script command. Here is the command line syntax used to invoke SAS and start a server:

```
 nohup sas -sysin sas-input-file -noterminal -logparm "write=immediate"
-log "/u/system/server%W.log" -logparm "rollover=auto" &
```

**`nohup`**
causes the associated SAS process to ignore HUP or HangUp signals that are sent from the operating system.

**sas**
specifies a site-specific path to the SAS executable that initiates the process.

**-sysin** *sas-input-file*
specifies the file that contains the SAS statements to start a server. For information about the content of this file, see "Managing a SAS/SHARE Server (Server Administrators)" on page 29.

**-noterminal**
specifies that no physical terminal is associated with this process.

**-logparm "write=immediate"**
writes server log messages to the server log immediately (without buffering).

**-log "/u/system/server%W.log"**
directs all log messages to the log file that is named by replacing **%W** with the current numeric week of the year.

For example, **server02.log** identifies a log that was generated during the second week of the year.

**-logparm "rollover=auto"**
if the pathname of the log file changes (for example, when **%W** is used to name weekly logs), the log is automatically saved in the current file, and a new file is opened to store the new content. For details, see the following topics.

- "LOG System Option: UNIX" in *SAS Companion for UNIX Environments*
- "SYSIN System Option: UNIX" in *SAS Companion for UNIX Environments*
- "LOGPARM= System Option" in *SAS System Options: Reference*
- "TERMINAL System Option" in *SAS System Options: Reference*

**&** (ampersand symbol)
allows the SAS process to run in background mode in the UNIX operating environment.

The following is an example of a SAS command that invokes SAS and starts a server:

```
 nohup /u/system/sas -sysin /u/system/startsrv.sas -noterminal -logparm
"write=immediate" -log "/u/system/server%W.log" -logparm "rollover=auto" &
```

### *Starting a Server Automatically*

You can configure a SAS/SHARE server to start automatically whenever the UNIX machine that the server runs on is re-started. The UNIX system administrator can insert the script command (outlined in Starting a Server Manually) into the last portion of either the **/etc/inittab** or **/etc/rc** UNIX system files. For a typical script command, see "Starting a Server Manually" on page 208.

*Note:* The **nohup** parameter should be omitted from the server's UNIX script command when it is part of the **/etc/inittab** or **/etc/rc** system file.

### *Setting SAS System Performance and Logging Options*

The following options affect the operation of the server:

BUFNO=
specifies the number of buffers to use for SAS data sets. The default is 1.

BUFSIZE=
specifies the permanent buffer size for an output SAS data set. The default is 0.

LOG=
specifies a destination to which the SAS log is written in batch mode.

MEMSIZE=
specifies a limit on the total amount of memory that SAS uses at any one time. The default is 32M.

For more information, see the chapter about system options in the *SAS Companion for UNIX Environments*.

For best performance, run the server on the machine where the shared SAS data resides. Do not run production servers in SAS foreground mode using the SAS windowing environment.

*Note:* The windowing environment supports a maximum of 32,767 lines that can be written to the SAS log.

# Windows: Creating the Server Environment

## *SAS/SHARE Server Can Run as a Windows Service*

You can configure a SAS/SHARE server as a Windows service, which enables the automatic start-up of the server when the Windows operating environment is started.

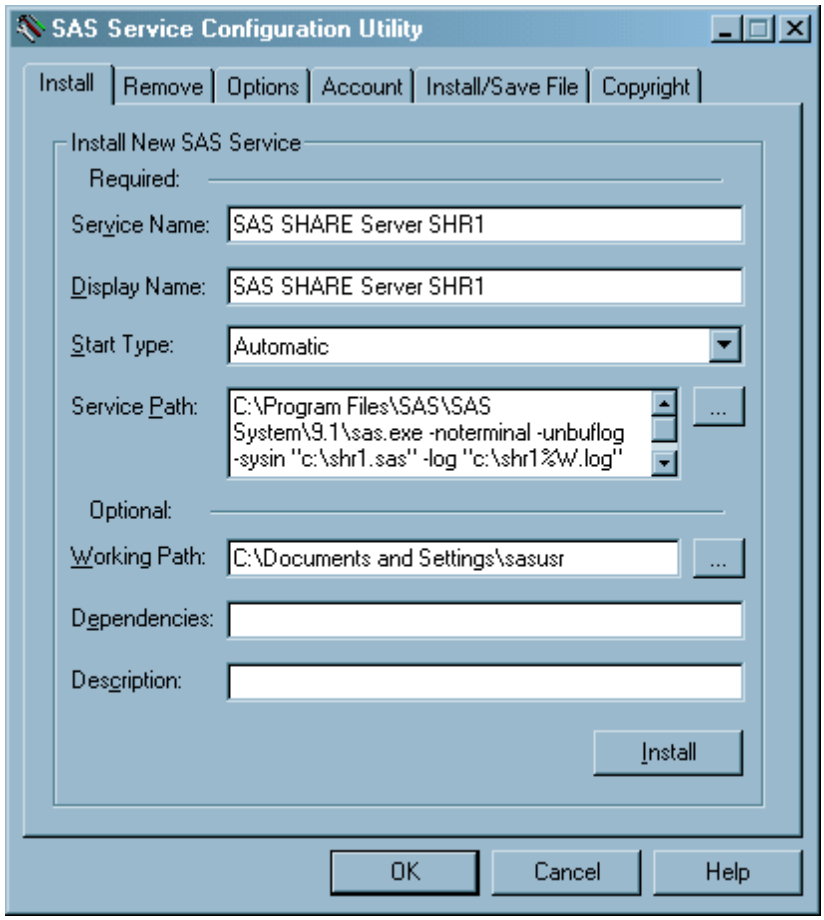
To configure a SAS/SHARE server as a Windows service, use the SAS Service Configuration Utility (SAS SCU), which is shipped with Base SAS for Windows. For information about using SAS SCU, see the online Help that is provided with the utility.

*Note:* The complete instructions for installing and using the SAS SCU are provided in the document "Installation Instructions for 9.3 of the SAS System for Microsoft Windows," which is included with the software.

## *Using SAS SCU to Install a SAS/SHARE Server as a SAS Service*

Configure the Server Service using these steps:

1. Start the SAS Service Configuration Utility (SAS SCU): **Start ⇨ Programs ⇨ your- SAS- System- folder ⇨ SAS Service Configuration Utility**
2. In the SAS SCU window, click the **Install** tab and complete the following fields:




**Service Name**
is used to create the service name as recorded in the Windows registry.

**Display Name**
is used to create the service name that is displayed in the Windows Service Control Manager. In the example, the server ID SHR1 is included to make it easier to identify among the multiple SAS/SHARE servers that might be configured.

**Start Type**
is used to specify whether the server is started automatically or manually. **`Automatic`** causes the SAS/SHARE server to start automatically when the Windows operating environment is started. Select **`Manual`** to enable the service to be started and stopped by an administrator using the Windows Service Control Manager.

**Service Path**
identifies the SAS invocation command line to be executed with corresponding system options for server-specific behavior. The following is an example of a service path specification:

```
C:\Program Files\SAS\SAS System\9.1\sas.exe
  -noterminal -logparm "write=immediate" -sysin "C:\shr1.sas"
  -log "C:\shr1%W.log" -logparm "rollover=auto"
```

**C:\Program Files\SAS\SAS System\9.1\sas.exe**
identifies the SAS executable.

**-noterminal**
prevents interaction with the console.

**-logparm "write=immediate"**
writes messages to the server log immediately (without buffering).

**-sysin "C:\shr1.sas"**
specifies a batch file to start the SAS/SHARE server. The file **shr1.sas** contains the following SAS program:

```
%let tcpsec=_secure_;
proc server id=shr1 authenticate=required;
run;
```

The PROC SERVER statement starts the SAS/SHARE server SHR1. The TCPSEC=_SECURE_ system option and the AUTHENTICATE=REQUIRED option specify that each client must be authenticated before accessing the server.

For more information about the TCP/IP access method, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE* .

**-log "C:\shr1%W.log"**
directs all log messages to the log file that is named by replacing **%W** with the current numeric week of the year. For example, **shr102.log** identifies a log that was generated during the second week of the year.

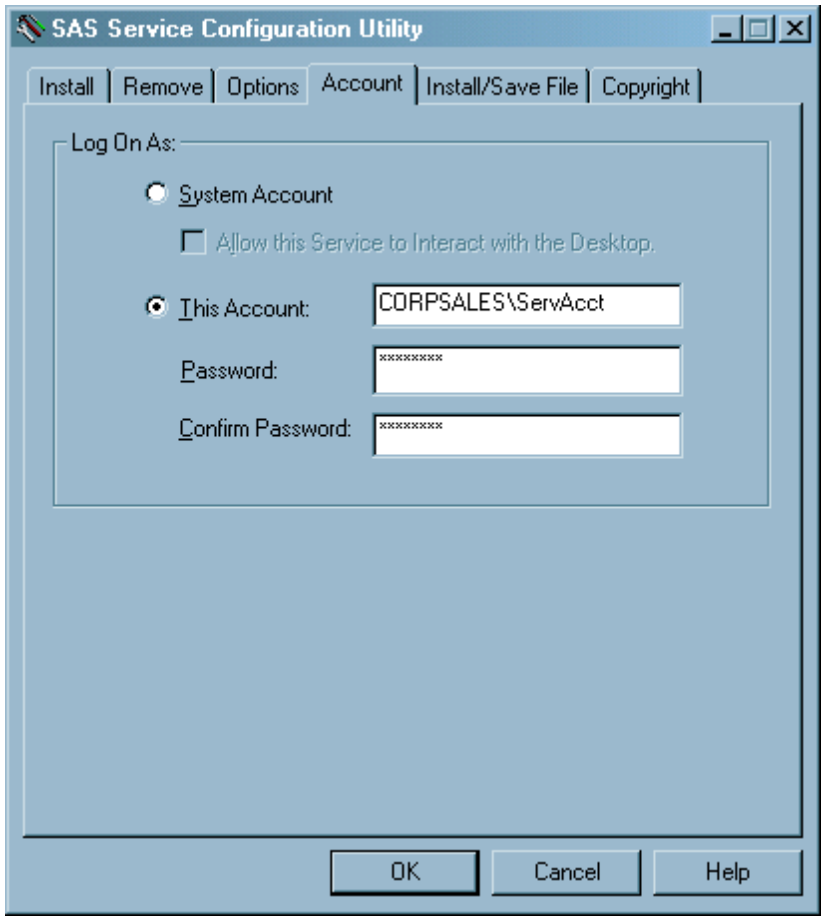
**-logparm "rollover=auto"**
if the pathname of the log file changes (for example, when **%W** is used to name weekly logs), the log is automatically saved in the current file, and a new file is opened to store the new content.

For details, see the following topics.

- "LOG System Option: Windows" in *SAS Companion for Windows*
- "SYSIN System Option: Windows" in *SAS Companion for Windows*
- "LOGPARM= System Option" in *SAS System Options: Reference*
- "TERMINAL System Option" in *SAS System Options: Reference*

**Dependencies**
specifies a list of Windows services, which will be started before the SHARE server service.

3. In the SAS SCU window, click the **Account** tab and complete the following fields:




**This Account**
specifies a valid Windows user account that the service will run under. The value for this field should be in the format *Domain\Account*. For example, **CORPSALES\ServAcct**.

**Password**
specifies a valid password for the user account that is specified in the **This Account** field.

**Confirm Password**
confirms the value that you entered in the **Password** field.

4. In the SAS SCU window, click the **Install** tab again and select **Install**.

   A message window appears and confirms that the service was installed successfully.

   Click **OK** to close the message window.

5. Click **OK** in the SAS Service Configuration Utility window to complete the configuration and installation of the service.

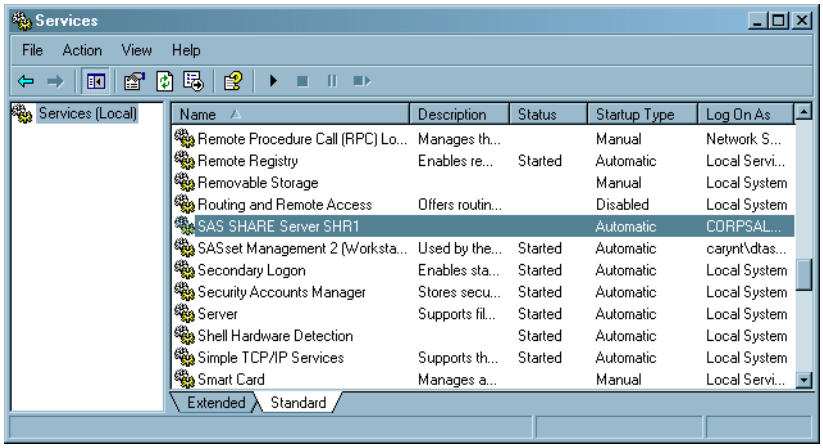
*Note:* To start the SAS/SHARE server service either manually or automatically, the account that you specified in the **This Account** field on the **Account** tab must be assigned the **Log on as a service** and **Act as part of the operating system** rights. The administrative interface that is used to assign user rights is dependent on the version of Windows that you are using. See the documentation for your Windows operating environment for instructions about assigning user rights.

## Starting and Stopping a Service

### Using the Windows Services Utility

From the Services utility in the Windows Control Panel, you can monitor Windows services, including the SAS/SHARE server service that you set up by using the SAS SCU.

To start or stop a service, click **Start** or **Stop**, as appropriate.




### Using DOS Commands

You can also start or stop a service by using commands at a DOS prompt.

To start a service, type **`NET START service-name`**, where *service-name* is the name of the service that you want to start.

To stop a service, type **`NET STOP service-name`**, where *service-name* is the name of the service that you want to stop.

## Removing or Changing an Installed SAS/SHARE Server Service

### Removing the Service

To remove the server service:

1. If the service is active, stop the service by using one of the methods described in "Starting and Stopping a Service" on page 214.
2. Start the SAS Service Configuration Utility (SAS SCU):**Start ⇨ Programs ⇨ *your-SAS-System-folder* ⇨ SAS Service Configuration Utility.**
3. In the SAS SCU window, click the **Remove** tab. Click the down arrow under **Choose the installed SAS service you want to remove** and select the service that you want to remove.
4. Click **Remove**.

### Changing the Service

To change the configuration for a service, remove the service and install it again.

*Appendix 3*

# Tuning Tips for Applications That Use SAS/SHARE Software



## Authors

This appendix is based on a paper by Steve Beatrous, Bill Brideson, Dan Squillace, and Jan Squillace. The original paper, which was published in the *Proceedings of the Twenty-First Annual SAS Users Group International Conference, Chicago, Illinois,*

*1996*, has been updated with new information about I/O resources and the TBUFSIZE= option in PROC SERVER.

# Introduction to Tuning Tips for Applications That Use SAS/SHARE Software

SAS has many tuning options, most of which are left at their default values. When an application accesses data through a SAS/SHARE server, sometimes the default values provide adequate performance and sometimes they do not.

SAS software is delivered to you properly tuned for a typical application that uses SAS/SHARE software. SAS makes some assumptions about the kind of processing that is going to take place in a typical application. Recognizing that your application might not be typical, SAS supplies tuning options that you can use to override default behavior.

This paper discusses programming techniques and option value adjustments that you can use to improve the performance of your applications that access data through servers. The information in this paper is the result of tuning several large applications that are in use at SAS.

# Overview of Tuning Tips for Applications That Use SAS/SHARE Software

This paper was originally presented at SUGI 18. Since that time, client/server applications have become more common. The paper is being updated and presented again because there is a growing audience interested in tuning their client/server applications.

This paper will give you some ideas to help you develop SAS applications that make the most efficient use of concurrent access to SAS files. Because this audience consists of people with different amounts of experience developing applications that use concurrent access to data, the first part of the paper will focus on overviewing the general model for accessing data in SAS. The later parts of the paper will draw on the general data model to describe how to tune a client/server application.

Since introducing SAS/SHARE software in 1987, SAS has compared two contrasting images to show the additional capability that SAS/SHARE software brings to SAS. One image shows a user's SAS session accessing files directly; the other image shows a user's SAS session connected to a server's session and the server's session accessing the files directly. The essential difference is that the data in a file that is accessed through a server travels through two SAS sessions whenever it is accessed. Of course, a server controls concurrent access to the data that is read and written through it, so its overhead has an important purpose. But it is important to remember that data accessed through a server requires more computing resources than data accessed directly.

One or more servers can execute at the same time on a single computer or in a network of computers. You can use different servers for different applications, or you can use a few servers to distribute the load of many applications.

When you use more than one server, each server performs only part of the work load. This allows each server to respond to requests more quickly. On the other hand, every process on a computer requires a certain amount of overhead simply to exist, and servers are no different from other processes in this regard. You must balance the performance

improvement that using multiple servers gives your users against the increased load on your system as more servers execute. The later parts of this paper will discuss measuring how much work a server is doing; you can use that information to determine when to add or delete servers.

# The SAS Library Model

You should make sure you thoroughly understand the material about the SAS library model in *SAS Language Reference: Concepts* before you attempt to tune SAS applications and servers at your installation. Here are some of the terms defined in that material that are most important to understanding this paper:

A SAS library can have five types of members, DATA, VIEW, CATALOG, PROGRAM, ACCESS, MDDB, and FDB. This paper will deal only with the types DATA, VIEW, and CATALOG.

A library member of type DATA is a SAS data file. Through SAS 5, SAS referred to such files as SAS data sets. A SAS data file can be compressed, and it can have zero or more indexes.

A SAS data view is a set of directions that tells a SAS view engine how to combine data from one or more sources into observations.

A SAS catalog is a file that contains smaller files; the files contained in a catalog are catalog entries. Some types of entries you might be familiar with are PROGRAM ( SAS/AF programs), SCREEN (PROC FSEDIT screens), and FORMAT (user-written formats).

# How Data Flows When You Use SAS Files

## *Introduction*

To tune applications that access data concurrently, it is to your advantage to understand how data is read and written in the different types of members of SAS libraries that can be accessed through a server.

It is important to remember that an application cannot run any faster when it accesses data through a server than it can when it accesses data directly. This might seem obvious, but it is surprisingly easy to simply blame an application's sluggish performance on the server without ever testing the application while accessing the data without going through a server. For many applications, the difference in performance between accessing the data directly versus accessing the data through a server will not be large. Whenever you develop a new application, verify that the application runs acceptably while accessing its data directly before you add a server to the application's data access.

## *SAS Data Files*

When a SAS session reads from a SAS data file that is accessed directly:

1. The procedure or DATA step requests an observation from the engine.

2. The engine requests the SAS host interface to read the page of the data file that contains the observation.
3. The engine extracts the observation from the page and returns it to the procedure.

When a SAS session updates or adds to a SAS data file that is accessed directly:

1. The procedure calls the engine to replace or add the observation.
2. The engine replaces or adds the observation in the page.
3. The engine calls the host interface to write the updated or new page to disk.

When a SAS session reads from a SAS data file that is accessed through a server:

1. The procedure or DATA step requests the observation from the REMOTE engine.
2. The REMOTE engine determines whether the requested observation is already available in its transmission buffer in the user's SAS session. If the observation is available, it is returned to the procedure.
3. If the observation is not already available in the user's SAS session, the REMOTE engine sends a message to the server to get a buffer full of observations, including the observation requested by the procedure.
4. The server fills the transmission buffer by requesting one or more observations from the engine that accesses the data file in the server's SAS session.
5. For each observation, the engine in the server's session requests the SAS host interface to read the page of the data file that contains the observation.
6. The engine in the server's SAS session extracts each observation from its page and returns it to the server.
7. After filling the transmission buffer, the server sends the buffer to the REMOTE engine.
8. The REMOTE engine extracts the selected observation from the transmission buffer and returns it to the procedure or DATA step.

When a SAS session updates or adds to a SAS data file that is accessed through a server:

1. The procedure calls the REMOTE engine to replace or add the observation.
2. The REMOTE engine replaces the observation in its transmission buffer or adds the observation to its transmission buffer.
3. If the data file is open for update access, the REMOTE engine sends a message to the server that carries the new or updated observation and requests that it be updated in or added to the data file.
4. If the data file is open for output access, the REMOTE engine adds observations to its transmission buffer until the buffer is full. After the transmission buffer is full, the REMOTE engine sends it to the server.
5. The server requests the engine that accesses the library in the server's SAS session to replace the observation in the data file or add the observation(s) to the data file.
6. The engine in the server's SAS session replaces or adds each observation by updating and creating pages in the data file.
7. The engine requests the SAS host interface to write each updated and new page to the data file.
8. The engine in the server's SAS session returns to the server.

9. The server replies to the REMOTE engine indicating that the updated or new observation has been stored in the data file.
10. The REMOTE engine returns to the procedure.

### *SAS Data Views*

The flow of data as a SAS data view is processed can be complex, because a view is a set of instructions that tells how to select and combine data from one or more sources.

A SAS data view can be interpreted in a user's SAS session or a server's SAS session. When a view is interpreted in a user's SAS session, the view file and none, some, or all of the data read by the view can be accessed through a server. When a view is interpreted in a server's SAS session, the view file and all of the data read by the view must be accessed by the server.

There are three types of SAS views:

- PROC SQL views, which are interpreted by the SQL engine
- SAS/ACCESS views, which are interpreted by SAS/ACCESS interface engines
- DATA step views, which are interpreted by the DATA step view engine

A view created by the SQL procedure can read SAS data sets (SAS data files and any kind of SAS data view).

When a SAS/ACCESS view engine is used in a multi-user server's session, the view engine can read only from the database; it cannot update the database. The flow of data is one-way: from the database to the interface engine to the server to the user.

A DATA step view can, like a PROC SQL view, combine data from SAS data files and SAS data views. In addition, DATA step views can include sophisticated calculations and read data from external files. A DATA step view can produce data exclusively by calculation, without reading any data.

### *SAS Catalogs*

SAS catalogs are containers for many different types of entries, and the data in each type of entry is accessed in a pattern unique to the entry type. Like the observations in SAS data sets, the REMOTE engine will combine records in a catalog entry into groups. The combination of records for catalog entries is done only for INPUT opens (OUTPUT and UPDATE opens transmit one record at a time).

## Concurrent Access: Update versus Read-only

Many applications use several SAS files. It is to your advantage if, while designing your application, you identify and divide the following:

- the set of files that must be updatable by more than one user at a time
- the files that will be updated by only one user, but while other users are reading the files
- the files that will be updated so infrequently that access to those files by all users is practically read-only.

The files in the first group are excellent candidates for access through a server. The files in the second group are often good candidates for access through a server, but for some applications the performance improvement from not accessing the files through a server might make it worthwhile to use a more complicated procedure to update those files while the users are not around. The files in the third group are almost always poor candidates for access through a server because all of the operating environments that SAS runs under provide shared read-only access to files, and that direct access is almost always faster than access through a server.

Here is a summary of the advantages and disadvantages of dividing files into read only and concurrently updated libraries:

- A SAS file that is accessed through a server usually costs more, in terms of computing resources, for users of the application to use than a SAS file that is stored in a library that is accessed directly by the users.
- Reduced traffic through a server optimizes response time for the users of the concurrently updated files.
- Simpler, more direct access to read-only copies of files reduces the cost of an application's query and reporting functions. Note that such a copy might be a subset instead of the entire file.
- A SAS file that is accessed through a server can be updated while it is being queried or reported on.
- Copies of files require disk space.
- A file in a SAS library that is accessed directly by users cannot be updated while a user executes the part of the application that uses that file.

# Computer Resources Used by a Server

## *Overview*

The information in this paper so far has been about SAS files and how they are used by an application. You will be a more effective application developer if, in addition to understanding how to make optimum use of SAS files, you also understand the computer resources that a server consumes. That understanding will allow you to design your applications to make optimum use of a server and optimum use of SAS files.

A server is an independently running SAS session that brokers requests for data from other SAS sessions. There are four types of computer resources that a server consumes:

- CPU (the computer's processor)
- I/O (input from and output to the computer's permanent storage)
- memory (the computer's working storage)
- messages (passing data between a server and its users)

CPU, I/O, and memory resources are consumed by every SAS session. Messages is a name for one measurable aspect of the complex area of communications resources; communications resources are consumed by SAS/SHARE software and SAS/CONNECT software because these two products enable SAS sessions to communicate with one another.

Any work done by a server consumes more than one kind of resource (if you are looking for simple uncomplicated truths, you might want to skip this section). A server can do several types of work and, as you might expect, not all types of work consume resources in the same relative amounts. For example, some work a server can do consumes much of the CPU resource but little of the other resources, while other work consumes much of the memory resource, less of the CPU resource, and very little of the other resources.

### *CPU*

A server creates processes as users connect to it and execute DATA steps, procedures, and windows. These processes (created on users' behalf) are assigned the work that is actually performed in the server's SAS session. This allows a process in a server's session to do work requested by one user and then yield control so that another process can do work for another user.

Most requests handled by the processes in a server require small bursts of CPU time. But there are several requests that can consume especially large amounts of CPU time:

- processing a WHERE clause
- interpreting a SAS DATA step view
- processing a compressed SAS data file

When a SAS data set is accessed through a server, every WHERE clause used to select observations from that data set is evaluated by a process in the server's SAS session. This increases the server's overall use of the CPU resource to reduce its use of the messages resource. Often, evaluation of a WHERE clause can be optimized by using an index to locate the selected observations. But when an index is not used, or selects more observations than satisfy the WHERE clause, the process in the server's session must search for observations that completely satisfy the WHERE clause. Searching can consume a significant amount of the CPU resource. While a process conducts a search, it yields periodically to allow other processes in the server's session to do work for other users.

A PROC SQL view can consume quite a bit of the CPU resource. The SQL view engine can join tables, it might need to sort intermediate files, and there might be several WHERE clauses in the view that require evaluation. The process in which the SQL view engine executes yields periodically while a view is interpreted.

DATA step views and SAS/ACCESS views also consume the CPU resource. The process in which either of these engines executes does not yield to allow other processes to run, although the server itself allows other processes to run when a group of observations has been prepared for transmission to a user's SAS session. A DATA step view that does a great deal of calculation while preparing each observation can have a visibly harmful impact on a server's response time to other users' requests.

When a compressed SAS data file is read, processes in the server's session decompress each observation; when a compressed SAS data file is created or replaced, a process in the server's session compresses each observation. In many cases the time required to decompress (or compress) is shorter than the time required to read the additional pages of an uncompressed file. In other words, trading increased use of the CPU resource for decreased use of the I/O resource can, on balance, reduce the length of time users wait for a server to respond. While a user processes a compressed data file through a server, other processes in the server's session can execute between groups of observations requested by that user; a SAS data file is not compressed or decompressed in its entirety in a single operation.

The "Programming Techniques" section of this paper offers ideas for reducing the CPU consumption of processes in a server's session under the following topics:

- “Choose the Appropriate Subsetting Strategy”
- “Index Wisely”
- “Know Your Application's DATA Step Views”

## I/O

### Overview

Because most work done by the processes in a server's SAS session involves I/O activity, those processes can spend a significant amount of time waiting for I/O activity to complete. (This time includes moving the head of a disk drive to the correct position, waiting for the disk to spin around to the position of the requested data, and transferring the data from the disk to the computer's working storage.) In the current release of SAS/SHARE software, while a process in a server's session waits for I/O activity to complete, other processes in the server's session do not perform other work that uses a different (CPU, memory, or messages) resource.

That waiting could, it would seem, become a bottleneck for a server, and in a few situations this problem is realized. But in practice most of a server's memory is used for I/O buffers and processes in a server's session usually satisfy most requests for data from I/O buffers that are already in memory.

A server usually allocates memory for one page of a file each time the file is opened, up to the number of pages in the file. For example, if the application being executed by a user opens a file twice, enough of the server's memory to contain two pages of the file is allocated; if ten users run the application, space for 20 pages of the file is allocated in the server's memory. The number of buffers allocated for a file will not exceed the number of pages in the file.

Of course, the pages of the file maintained in memory are not the same set of pages all the time: as users request pages of the file that are not in memory, pages that are in memory are written back to the file on disk if they have been modified, or if an in-memory page has not been modified its buffer is simply used to read the new page.

A larger page size can reduce the number of I/O operations required to process a SAS data file. But it takes longer to read a large page than it takes to read a small one, so unless most of the observations in a large page are likely to be accessed by users, large page sizes can increase the amount of time required to perform I/O activity in the server's SAS session.

There are two patterns in which data is read from or written to SAS files:

- sequential
- random

When an application processes a SAS file in sequential order, no page of the file is read into or written from the server's memory more than once each time the file is read or written. Also, observations are transmitted to and from users' sessions in groups, which conserves the messages resource.

In many applications that are used with concurrently accessed files, data is accessed in random order, that is, a user reads the 250th observation, then the 10,000th observation, then the 5th observation, and so forth. When a file is processed in random order, it is much more difficult to predict how many times each page of the file will be read into or written from the memory of a server's SAS session. In addition, only one observation is transmitted on each message between server and user, which does not conserve the messages resource.

The "Programming Techniques" section of this paper offers ideas for reducing the I/O load of a server under the following topics:

- “Clean Up Your Data Files”
- “Choose the Appropriate Subsetting Strategy”
- “Choose Page Size Wisely”
- “Specify Sequential Access When an SCL Program Doesn't Need Random Access”

### *Overlapping I/O*

Overlapping I/O for sequential read operations is a performance enhancement for applications that use the remote engine. This enhancement was made for SAS 8.1 by improving the internal functions of the remote engine to support the use of multiple data buffers for sequential I/O operations.

When the remote engine is reading sequential data from the SAS/SHARE server, it requests a buffer of data from the server and while it is waiting, it starts the I/O for a second buffer of data. When the server returns the first buffer of data, the remote engine passes it to the requesting application.

While the application is reading observations from the first buffer, the server is returning the observations in the second buffer. When the application has read all of the observations in the first buffer and has started reading observations from the second buffer, the REMOTE engine sends a message to the server to retrieve a third buffer of observations. The REMOTE engine reuses the memory that was allocated for the first buffer to store the third buffer of observations.

This sequence continues until all of the requested data has been read. This reduces the elapsed time for applications that read data sequentially by overlapping reading and processing.

## *Memory*

A computer's working storage is used by a server to load programs, hold I/O buffers, and maintain control information. When a server's working set becomes large compared to the amount of memory installed on a computer, a significant amount of the server's working storage can be stored on disk by the operating environment's virtual memory manager.

Large amounts of a server's memory are consumed by the following:

- a SAS data view that contains an ORDER BY clause
- many indexes on data files accessed through a server
- a large number of files open at the same time
- data files that have large page sizes

Because the ORDER BY clause causes the observations produced by a view to be sorted every time the view is interpreted, it requires memory to be used for a work area for the sorting step. Your application should use this clause only in its views when it has a clear benefit for your users.

When a SAS data file is opened, all indexes on the file are opened. Therefore, when a SAS data file has many indexes, a large amount of memory in the server's SAS session can be used to store pages of the index file and related control information. Of course, when many SAS data files that are accessed through a server each have many indexes, this effect is multiplied.

At SAS, we have observed that the majority of servers' memory has been consumed by I/O buffers. Carefully selecting the number of times each file is opened by your application and the page size of each file can have considerable impact on the amount of memory required by a server.

The "Programming Techniques" section of this paper offers ideas for reducing the memory requirements of a server under the following topics:

- “Choose Page Size Wisely”
- “Index Wisely”
- “Limit the Number of Files Open During Execution of an SCL Program”

## Messages

Messages are the communication events between users' SAS sessions and a server. Whenever a piece of information (for example, an observation) is moved from a server to a user, a message is sent from the user to the server, and a reply is sent back from the server to the user.

Messages and replies are transmitted by communications access methods. The cost of a message varies greatly with access method. Memory-to-memory communication within a single computer, for example by means of the Cross-Memory Services (COMAMID=XMS) access method is very rapid, while messages that flow on cables between computers, for example by means of the TCP/IP (COMAMID=TCP) access method take much longer to travel between SAS sessions.

At SAS, we have observed that the cost of sending data by means of most communications access methods is more directly a function of the number of messages than the amount of data. In other words, to move a million characters of data between a user and a server, it takes less time to send the data in 100 messages than to send the data in 10,000 messages.

SAS/SHARE software conserves the messages resource by doing the following:

- transmitting data between servers and users in groups
- evaluating WHERE clauses in servers' sessions
- interpreting SAS data views in servers' sessions

The "Programming Techniques" section of this paper offers some ideas for conserving the messages resource under the following topics:

- “Choose the Appropriate Subsetting Strategy”
- “Understand and Control Random Access”

The "Tuning Options" section shows options you can use to control the grouping of observations in messages between servers and users:

- TBUFSIZE=
- TOBSNO=

# Minimizing and Optimizing Resource Consumption

## Overview

Now that you understand how SAS and SAS/SHARE software use files and computer resources, it's time to apply that knowledge to the design and implementation of your applications.

The most productive way to optimize the performance of your application is programming it to work as efficiently as possible. You can almost always realize more performance improvement by coding your application to exploit features of SAS than you can gain by adjusting the operation of SAS.

When you decide to adjust SAS to operate differently, remember that tuning is a balancing act and invariably requires compromise. Of course, to effectively tune SAS you must understand what your application's bottlenecks are.

This section will first list some programming techniques that are based on the information presented earlier in this paper. After that, the tuning options of SAS/SHARE software and SAS will be described.

## Programming Techniques

### Clean Up Your Data Files

The most obvious way to reduce the amount of work done by a server is eliminating unused variables and observations from the files that are accessed through the server. To make sure that your files are no larger than they need to be, periodically remove or archive unused data.

As a SAS data file matures, users add new observations, update existing observations, and forget about old observations. In most cases the level of activity is greatest on the newest observations. If the users of your application do not frequently access older information, consider moving older observations from files that are concurrently updated to archive files that are accessed directly (instead of through a server).

Also as a SAS data file matures, new variables are added, some variables turn out to be larger than they need to be, and some variables lose their usefulness. Periodically check your application's SAS data files for variables that are longer than they need to be and for variables that are no longer used.

While compressing a SAS data file reduces the number of pages in it, compression cannot be as efficient at eliminating unused space as you can be by deleting unused observations and variables and by shortening variables that are longer than necessary.

Smaller data files improve the performance of all SAS sessions by reducing the amount of disk space required by each file, by reducing the number of I/O operations required to process the data in each file, and by reducing the number and size of messages required to transmit the data in a file between a server and its users.

### Choose the Appropriate Subsetting Strategy

Creating a subset of the observations in a SAS file can consume large amounts of the I/O and messages resources. There are several subsetting techniques available in SAS:

- any WHERE clause that is optimized by the use of an index
- any WHERE clause that is not optimized by the use of an index
- the subsetting IF statement in the SAS DATA step
- the FIND, SEARCH, and LOCATE commands in SAS/FSP procedures

When an index is not used to locate directly the observations that satisfy a WHERE clause, the process in the server's session must read observations from the data file until it finds one that matches the WHERE clause. This can consume a very large amount of the I/O and CPU resources. Those resource requirements can be greatly reduced when the variables in the WHERE clause are indexed.

The subsetting IF statement in the DATA step and the FIND, SEARCH, and LOCATE commands in SAS/FSP procedures perform the specified comparison in the user's SAS session instead of in a process in a server. This requires that every observation in the SAS data set be transmitted from the server's session to the user's session, which can consume a very large amount of the messages resource, in addition to the I/O and CPU resources required to read the data file. Because the comparisons of a WHERE clause are performed in the server's session, only the selected observations are transmitted to the user's session and the message resource is conserved.

The I/O resource consumption is the same unoptimized WHERE, subsetting IF, and FSP's FIND, SEARCH, and LOCATE. Using WHERE clauses is recommended, however, because the messages resource consumption is higher for the subsetting IF statement and the FIND, SEARCH, and LOCATE commands.

### *Index Wisely*

Indexing is a tool that optimizes WHERE clause selection of observations from SAS data sets. A WHERE clause without an index requires the process in the server to read every observation in a SAS data set to find the observations that match the WHERE selection criteria. An index often enables the server to locate the records that satisfy a WHERE clause without having to read the records that do not match.

Adding indexes might be a good idea if your application seems to be taking too long to execute WHERE clauses. However, indexes require extra memory and might present a problem for a server that is memory constrained.

A complete description of index usage can be found in the paper "Effective Use of Indexes in the SAS System," in the *Proceedings of the SAS User's Group International Sixteenth Annual Conference*.

### *Look at a Clock Before You Create an Index*

When a SAS data file is accessed through a server, creating an index on it prevents the server from responding to other users' requests. While it can be useful to create an index while a data file is accessed through a server, indexes on large files should be created after hours. Indexes on large data files should not be created while a server is expected to respond quickly to users' requests.

### *Choose Page Size Wisely*

Larger page sizes can be used to reduce the number of I/O operations required to process a SAS data file. But it takes longer to read a large page than it takes to read a small one and larger pages can increase the memory load on a server.

Large page sizes can be useful if most of the observations on each page are likely to be accessed each time the page is read into the server's memory, or if a large page size causes all or most of a SAS data file to be kept in the server's memory. Otherwise, large page sizes can increase the amount of time required to perform I/O activity in the

server's SAS session to the detriment of the server's ability to provide timely response to users' requests.

### *Understand and Control Random Access*

It is often worth the effort to study the order in which the users of your application access the data in your application's files. That tells you how widely dispersed your users' patterns of reference are. Sometimes you can reduce the amount of dispersal by sorting one or more files by a variable (like date last updated) that happens to correlate (even to a small degree) with your users' pattern of access.

Here are the components of SAS that are used most frequently to access data in a random order:

- the "n" (position to observation number) command of a SAS procedure
- the POINT= option in the SET and MODIFY statements in the DATA step
- the KEY= option in the SET and MODIFY statements in the DATA step
- the FETCHOBS() function in SAS Component Language
- the SETKEY() function in SAS Component Language
- the use of an indexed variable as a BY variable

### *Specify Sequential Access When an SCL Program Doesn't Need Random Access*

The SCL OPEN() function allows you to specify that a file will be sequentially accessed (the default is random access). There are two types of sequential access that can be requested with SCL OPEN():

- strict sequential ('IS' for input and 'US' for update)
- limited sequential ('IN' for input and 'UN' for update)

The server will by default transmit multiple observations per read for either 'IS' or 'IN' open modes.

If the application's use of data is predominantly sequential, but you occasionally need to reread a previously read observation, then use a mode of 'IN' or 'UN' in your SCL OPEN() function. If the application's use of data is strictly sequential (you will never revisit a previously read observation) then use the open mode 'IS' or 'US'. The 'IS' and 'US' open modes are the most efficient for SCL. An 'IS' or 'US' open mode, however, will restrict an SCL application to those functions that access data sequentially. Here are the SCL functions that access data in a random pattern:

- FETCHOBS()
- DATALISTC()
- DATALISTN()
- POINT()

Specifying an access pattern in an SCL OPEN() function is documented in the *SAS Component Language: Reference*. Here is an example of specifying a sequential access pattern in an SCL OPEN() function:

```
DSID = OPEN( 'MYLIB.A', 'IN' );
```

### Limit the Number of Files Open During Execution of an SCL Program

An open file consumes memory in both users' and servers' SAS sessions. If a server consumes too much memory, check the applications that access data through that server to see if any of them open files before they are needed or leave files open when they are not being used.

There are three strategies for using SAS data sets in an SCL program:

- Open during initialization of the application and leave open until the application terminates.
- Open as needed, and then leave open until the application terminates.
- Open as needed, and then close as soon as possible.

The initialization code of an application is the place to open the SAS data sets that will be used throughout the execution of the application. But if an application's initialization code must open a large number of files, the time it takes to get started can be long. By studying how an application is used, you might discover some SAS data sets that can be opened as functions are requested while the application executes, which can reduce the amount of time the application takes to initialize and reduces the concentration of time required to open files.

Whether they are opened during initialization or later, lookup tables that are small should usually not be closed until an application terminates because the single I/O buffer required by such a lookup table does not require a large amount of memory. In such a case it is frequently economical to use a small amount of the memory resource to conserve the CPU resource that would be required to open and close the lookup table over and over.

Larger SAS data sets, and SAS data sets that are used extremely infrequently (for example, once during initialization) or during a seldom-used function (for example, a lookup table on a rarely updated field), should usually be left closed whenever they are not being used.

### Evaluate Each Report's Timeliness Requirement

Consider how frequently each of your application's reports is generated and how timely the data summarized by the report must be. If a report must be based on current information, it must be based on files that are concurrently updated. A report that does not require up-to-the-second information can be generated from files that are directly (and inexpensively) accessed instead of files that are accessed through a server.

For example, a travel agent making reservations or a stock broker making trades require every query to show up-to-the-second information. On the other hand, daily reports or analysis of long-term trends can use data that are out of date by several hours, several days, or even several weeks or months.

When copying data from a server, it can be subset horizontally with a WHERE clause, and it can be subset vertically with a DROP= or KEEP= data set option. (In relational terminology, the horizontal subsetting is selection and vertical subsetting is projection.) Be sure to take advantage of both methods when copying a file from a server to make the copy of the file as small as possible and, therefore, ensure that reports are generated as efficiently as possible.

Don't forget that files can be stored in users' WORK libraries. It can be most efficient to copy a file that is concurrently updated from a server to a user's WORK library and then use that file more than one time to generate reports. Such a copy of a file contains very timely data yet is not especially expensive to create or use.

A SAS data file that is accessed directly is almost always less costly to use than a file that is accessed through a server.

### *Be Aware of How Frequently Each File Is Updated*

Many applications contain one or more query functions that use a lookup file to offer a user a set of values that are valid to enter into a field. Such a file is read, but never updated, by the majority of the users of the application. Occasionally, values must be added to and removed from the lookup files as the valid input data for the application changes.

A lookup file that is used frequently and updated less often than once a week is likely to be a good candidate for not being accessed through a server, if it would be easy to find some time during the week when the files can be updated because the application is not being used. On the other hand, a lookup file that is updated many times each day should, in many cases, be accessed through a server because updating the file will be convenient: the lookup file can be updated while users use it to perform queries.

SAS catalog entries illustrate another way that update frequency can change.

An application might use only a few or many catalog entries. Like lookup files, catalog entries that are updated frequently are likely candidates for access through a server. But catalog entries that are never changed, or only changed very infrequently, should not be accessed through a server.

The update frequency might change for some of an application's catalog entries over time. For example, while an application is under development and being tested, it can be extremely convenient for the developers of the application to be able to update any catalog entry while those testing the application continue with their work. During this phase, the convenience of accessing the catalog entries through a server can more than pay for the cost of the overhead of server access. After the testing is completed and the application has stabilized, some or all of the application's catalogs can be moved to a SAS library that is accessed directly by the users of the application; in this phase efficient access by the users is more important than being able to update the catalog entries conveniently.

Remember that not all of an application's catalog entries must be accessed the same way. Catalog entries that must be frequently updated can continue to be accessed through a server, while other catalog entries that change very seldom can be stored in SAS catalogs that are accessed directly by the users of the application.

### *Know Your Application's DATA Step Views*

While it is creating each observation, a process in a server's session that is interpreting a DATA step view does not yield control to allow other processes in the server to execute other users' requests. While DATA step views can be useful in a server, they must be used carefully. A DATA step view that requires a small amount of processing to create each observation will not prevent other processes in a server's SAS session from responding to other users' requests. But a DATA step view that contains many DO loops with many calculations, and reads (or even writes) many records in external files or SAS data sets, can take a very long time to create each observation. Such a DATA step view should not be interpreted in a server's session because it does not yield control until each observation is created.

If it is advantageous to your application for its DATA step views to be interpreted in a server's session, be sure that any external files read by the DATA step view are available to the server's SAS session.

## Tuning Options in SAS/SHARE Software

### Overview

SAS/SHARE software makes some assumptions about the relative values of resources. For example, SAS/SHARE software considers messages to be more expensive than memory so it attempts to use more memory to reduce the number of messages. The default values and behavior might not be optimum for your application, so you have the opportunity to adjust the following:

- when and in what amounts observations are transmitted in groups instead of individually
- which SAS data views are interpreted in users' SAS sessions and which are interpreted in the server's SAS session
- how frequently a long-running process in a server's SAS session yields to allow other users' requests to be processed

SAS/SHARE software automatically attempts to conserve the message resource by transmitting observations in groups whenever possible. Observations can always be transmitted in groups when a data set is being created or replaced, but when a data set is opened for update access it is never appropriate to transmit more than one observation at a time. The grouping of observations when a data set is opened for input depends on the situation; you control whether observations are transmitted in groups according to these factors:

- whether the data set is opened for random or sequential access
- the control level of the data set
- the use of the TOBSNO= data set option to override the default behavior

Here are the factors that control how many observations are transmitted in each group:

- the value specified for the TBUFSIZE= option in the PROC SERVER statement
- the value specified for the TOBSNO= data set option

### TBUFSIZE= Option in PROC SERVER

The TBUFSIZE= system option in the PROC SERVER statement specifies the suggested size of a buffer that the server uses for transmitting information to or receiving information from a client. When this option is not specified in the PROC SERVER statement, the value of the TBUFSIZE SAS system option, if specified, is used. The default value is 128K.

A key use of these transmission buffers is in transmitting observations. The server uses the TBUFSIZE value when computing the number of observations to transmit in each multi-observation transfer between the server and the client sessions. However if the observation size, plus overhead, exceeds the TBUFSIZE value, only single-observation transfers are done.

You cannot calculate the number of observations per transfer by dividing the observation length into the value that you specify for the TBUFSIZE= option. To determine the effect of this option on your data sets, use the PROC SERVER options LOG=MESSAGE and ACCTLVL=DATA and compare the number of messages exchanged between the server and the client sessions as a function of the value of the TBUFSIZE= option and the number of observations in the data set.

Here is an example of using the TBUFSIZE= option:

```
PROC SERVER  TBUFSIZE=256K
             <other PROC SERVER options>;
```

### *TOBSNO= Data Set Option*

Independently of the TBUFSIZE= option's effect on a server's overall behavior, you can control the number of observations per group for individual data sets that are accessed through the server. For example, if you specify TOBSNO=3, three observations will be sent in each message.

The TOBSNO= option can be specified wherever SAS data set options are accepted: as an argument to the OPEN() function of SAS Component Language, in the DATA= option in a SAS procedure, and in the SET, MERGE, UPDATE, and MODIFY statements in the DATA step. It must be specified for each data set for which you want grouping behavior that is different from the default.

When a data set is opened for input with a sequential access pattern, a server calculates the number of observations per group as the smallest of the following:

- 3/4 of the number of observations in the data set
- the number of observations that will fit into an MOTB

When a SAS data set is opened for input with a random access pattern, the default behavior is transmitting observations individually (the group size is one). This ensures that a user always receives up-to-date data when they position to an observation, and it reduces wasted communications bandwidth because no observations are transmitted to a user's session except the specific observations requested.

At other times, the TOBSNO= data set option can be used to increase the number of observations transferred in each group. For example, consider an SCL program in which the SAS data set DSID is passed to a DATALISTC() or DATALISTN() function. The data set is read from beginning-to-end by the function, and then the observation chosen by the user is reread. Because by default the OPEN() function of SCL specifies a random access pattern, observations for that DSID are transmitted individually. But the access pattern of the DATALISTC() and DATALISTN() functions is really skip sequential, so transmitting observations individually is not optimum. TOBSNO=4 could be specified in a case like this to reduce the number of messages by three-quarters. (Note that the user could change the open mode from 'I' to 'IN' as an alternative to specifying the TOBSNO= data set option.)

The number of observations transmitted when a data set is opened for input is summarized below. Here is an example of using the TOBSNO= data set option:

```
PROC FSVIEW DATA=MYLIB.A(TOBSNO=10);
```

### *RMTVIEW= and NORMTVIEW Options*

Consider each SAS data view used by your application and determine whether the view should be interpreted in the server's SAS session or the users' SAS sessions. You decide where to have a view interpreted according to these considerations:

- How many observations does the view produce?
- How much data is read by the view?
- Where is the data that is read by the view?
- How much work must the computer do to interpret the view?

Some PROC SQL views are especially good candidates for interpretation in a server's SAS session because the number of observations produced by the view is much smaller than the number of observations read by the view, the data sets read by the view are

available to the server, and the amount of processing necessary to build each observation is not large.

Other PROC SQL views should be interpreted in users' SAS sessions because the number of observations produced by the view is not appreciably smaller than the number of observations read by the view, some of the data sets read by the view can be directly accessed by the users' SAS sessions, and the amount of processing done by the view is considerable.

By default, SAS data views are interpreted in a server's SAS session, but the RMTVIEW= option in the LIBNAME statement enables you to have the views in a library interpreted in users' SAS sessions instead. The NORMTVIEW option in the PROC SERVER statement enables you to prevent all SAS data views from being interpreted in the server's session.

SAS/ACCESS views do not provide update access to the underlying database when they are interpreted in a server's session, so it is often more practical to interpret SAS/ACCESS views in users' SAS session.

If it is useful for your application to have a SAS/ACCESS view interpreted in a server's session, ensure that all of the necessary database interface components are available to the server's session.

If a user's SAS session is capable of using a SAS/ACCESS interface engine to access the underlying database, it is more efficient to execute the SAS/ACCESS interface engine in the user's SAS session. Note that in this case it might be convenient to store the view file in a SAS library that is accessed through a server if the view will be updated frequently and used by more than one user.

Like SAS/ACCESS views, DATA step views are very often most useful when interpreted in users' SAS sessions. For more information about interpreting DATA step views in a server's session, see “Know Your Application's DATA Step Views” on page 229.

For a complete description of the RMTVIEW= option in the LIBNAME statement, see Chapter 10, “Remote Library Services,” on page 123.

Here are some examples of specifying the RMTVIEW= and NORMTVIEW options:

```
LIBNAME MYLIB 'my SAS library'
              RMTVIEW=YES
              <other LIBNAME options>;
PROC SERVER  NORMTVIEW
             <other PROC SERVER options>;
```

### LRPYIELD= Option in PROC SERVER

Some components of SAS yield control periodically and can be directed to do so more or less frequently than their default rate. These components are called long-running processes and include evaluating WHERE clauses and interpreting PROC SQL views.

Changing the rate at which control is yielded is delicate because the act of yielding control consumes some CPU resource: increasing the frequency at which control is yielded increases a server's CPU consumption all by itself. You can change the rate at which the processes in a server yield control by varying the value of the PROC SERVER option LRPYIELD=. The default value of this option is 10,000; the option has no units.

To make long-running processes yield relatively more frequently, specify a value greater than 10,000. While a higher value might have the effect of providing more even response time to a server's users, this comes at the expense of increased consumption of the server's CPU resource. Also, the processes that run for a long time run even longer when they are asked to yield more frequently.

To make a long-running process yield less frequently, specify a value smaller than 10,000. A lower LRPYIELD= value might make some individual user requests (like an SQL join with a sort) complete sooner, but the server's other users are forced to wait as the long-running process does more work before it yields control. Response time can become more uneven when long-running processes are allowed to yield less frequently.

This option is documented in "Overview of the SERVER Procedure" on page 105.

Here is an example of specifying the LRPYIELD= option:

```
PROC SERVER  LRPYIELD=5000
             <other PROC SERVER options>;
```

### Multiple Servers

This is not an option you specify in a SAS program statement; instead it is a method of managing the workload of concurrent access to SAS data sets.

If you determine that a server is consuming too much of a resource and you cannot reduce the server's consumption of that resource any further, creating an additional server allows you to divide your applications' workload among several servers.

SAS/SHARE software includes a family of SAS macros that help you manage SAS file access through multiple servers. Those macros are documented in "SAS/SHARE Macros for Server Access" on page 75.

## SAS System Options

### Overview

SAS has several SAS I/O tuning options. These options are most relevant to applications that access data through a server:

- the BUFSIZE= data set and system option
- the COMPRESS= data set and system option

### BUFSIZE= Option

When a file is created, use the BUFSIZE= data set option to specify the size of the pages of the file. The SAS default page size is optimum for files that are processed sequentially, but it might not be optimum when the observations of a file are accessed in random order. PROC CONTENTS shows the page size of a SAS data file.

You might find it useful to balance the pattern in which a file is randomly accessed against the number of observations stored on each page of the file. If most random access sequences access observations in very different locations in the file, then a small page size will improve performance because most of the observations on each page are not used. On the other hand, if most random access sequences are likely to be to observations that are physically near each other in the file, you might be able to take advantage of a large page size to have many of the observations read from the file into the server's memory at once.

If you want to keep all or most of a SAS data file in memory, you can choose a very large page size. Of course, this can consume a lot of the server's memory so you should use such a page size only when you really want to. If you expect that not much data from a large file will need to be in memory at one time, choose a small page size to make reading and writing each page as fast as possible.

If you find that your server is spending a significant amount of time waiting for I/O operations to complete, consider recreating the files that are not used for sequential access with a smaller page size.

Here is an example of using the BUFSIZE= data set option:

```
DATA MYLIB.A(BUFSIZE=6K);
   SET MYLIB.A;
RUN;
```

### *COMPRESS= Option*

This option is used to cause a SAS data file to be stored in compressed format. Compressing files usually makes them smaller, so a server is able to read more observations per I/O request when a file is compressed. The reduction in file size for a compressed file (which conserves the I/O resource) is paid for, though, by an increase in the consumption of the CPU resource (which is used to decompress the observations as the file is read). If your server is CPU-bound, compression could do more harm than good, but if your server is I/O-bound, compression could reduce its consumption of the I/O resource.

## Using Operating Environment Tools

### *Introduction*

Up to this point, we have been looking at SAS application and server performance from an internal point of view. Now we turn to an external point of view. By performance externals, we mean several things. First, at what rate is a server consuming resources such as CPU, memory, and DASD I/O? Second, with what other workloads is a server competing for these resources? And third, what policy is being used to manage a server's access to resources with respect to other work in the system?

There are several monitors available for MVS and VM to help you analyze a server's resource utilization and contention with other workloads. On MVS, most sites license the IBM RMF product. RMF Monitor II and Monitor III support interactive analysis of SAS/SHARE performance. Also available on MVS are Candle Corporation's Omegamon and Landmark System's TMON for MVS. Prominent products on VM include Omegamon from Candle Corporation and XAMAP and XAMON from Velocity Software.

These monitors can help you answer the following questions:

- Are my servers getting appropriate access to resources?
- Is another workload causing a severe contention problem for one of my servers? For example, is my server fighting with another application over access to the same disk drive?
- What resource bottlenecks are most critical to my applications? Where should I direct my tuning efforts?

Often, solutions to resource utilization problems result in making trade-offs among resources. For example, you might be able to reduce I/O by allocating additional buffers. But the additional buffer allocation will take more memory. Use of one of these monitors can help you evaluate the effectiveness of the trade-off.

It is beyond the scope of this paper to tell you exactly how to use specific operating environment performance monitors. We are making the non-trivial assumption that you

or someone else on your staff have that knowledge. Basically, every system has three principal resources: CPU, I/O, and memory. We will look at examples of managing each of these for servers:

### Managing CPU

The most critical factor here is assuring that your servers are getting a reasonable share of the available CPU time. Servers in general ought to run at a higher (for example, transaction monitors and database servers). You can tune your SAS/SHARE application meticulously only to be foiled if a background process (for example) is preventing your servers from getting CPU time.

If CPU time is a scarce resource on your system, that is, your system is usually running at very high CPU utilizations, then you need to consider SAS/SHARE tuning actions that can reduce CPU time. Two specific examples are type of server connection and whether to use data compression.

### Managing I/O

The first thing to consider here is the amount of contention with other work on the system. Are your SAS libraries competing with other work on channels, disk controllers, or disk drives which are too busy? Too busy on I/O channels and control units is highly specific to each operating environment and hardware vendor. But in general it is safe to say that if a disk drive is consistently above twenty percent busy, then off-loading work from that drive ought to be considered.

If there is no significant contention with other work, then you need to consider spreading application libraries using SAS/SHARE across multiple disks.

If waiting for I/O is still a problem for your servers, then you need to consider SAS/SHARE tuning options which can reduce I/O time. These include using smaller page sizes for randomly accessed data, adding indexes for randomly accessed data, and possibly using data compression. Data compression is a specific example of the resource trade-off problem mentioned earlier. Data compression can reduce I/O and disk storage but will increase CPU time.

### Managing Memory

Memory is an interesting resource in that it directly affects both CPU and I/O resource consumption. Too little memory increases both. Additional memory can reduce both. The most critical factor here is to ensure that your servers have sufficient memory to prevent excessive wait for paging. Most operating environments have controls to differentiate the amount of memory given to various workloads on the system.

If real memory is a scarce resource on your system, then you need to consider SAS/SHARE tuning actions that reduce memory consumption. Chief among these are reducing data set page sizes to reduce I/O buffer memory requirements and using shared SAS system images where possible.

## Conclusion

The concurrent access capabilities that SAS/SHARE software adds to SAS give developers opportunities to create applications that allow their users to have up-to-date data and to be more productive.

Such applications use SAS in new ways. This paper has discussed areas to be aware of and ways to trade usage of one resource for another. This information enables developers of applications that take advantage of concurrently accessed data to write those applications to use the available computer resources in the most efficient ways possible.

*Appendix 4*

# SAS Component Language (SCL) Application



## Introduction to the SAS Component Language (SCL) Application

The SAS Component Language (SCL) application that is presented in this appendix, when used with the FSEDIT procedure or the FSEDIT command, implements an order-processing and inventory-maintenance system. Because the purpose of this example is to illustrate SCL programming techniques, some error-handling and user-friendly enhancements were omitted to keep the example as clear as possible.

This example exploits the features of SAS/SHARE that allow several users to update a SAS data set at the same time. That capability automates maintenance of the inventory data and allows data about the orders to be maintained centrally, which can facilitate analysis of the orders that have been received.

## Audience

This example is for SAS application programmers who are familiar with SCL and with the FSEDIT procedure and FSEDIT command.

# Inventory and Order System

## Overview

There are three general types of data that are involved in an order-processing system:

- customer information, which includes name, address, and a customer number that uniquely identifies the customer
- inventory information, which tracks the amount of each item that is in stock at any specific time
- orders information, which identifies items in inventory that a customer has placed an order for

## Customer Information

In order to simplify this example, the application in this section does not include a customer data set. This SCL program, which is expanded for use in an actual order-processing system, would open the customer data set during the FSEINIT step and close it during the FSETERM step.

The customer data set is ideal to access through a SAS/SHARE server to keep information up-to-date, such as customers' address changes, and contact information.

For the purpose of entering orders, the customer data set would usually be opened for read-only access through the server, because customer information is not usually updated at the time an order is taken.

Updating the customer data would ordinarily be done by using a second FSEDIT application that would also access the data set through the server. That application would be available to customer service representatives and administrative personnel.

## Inventory Information

In this example, the inventory data is accessed through a SAS/SHARE server. The data is completely hidden from the user of the order-entry application. The purpose of this example is to show how to maintain information automatically in a SAS data set.

The inventory file is also ideal for accessing through a SAS/SHARE server to keep the available quantity of each item current.

In the order-processing example, the inventory data set is opened for update access through the server. Read-only access to the inventory data set might be given for the reporting programs that are run either nightly or on-demand, which would also access the data set through the server. The inventory reporting and analysis programs are not included in this example.

In a mature inventory-management system, there would be additional access to the inventory data by employees who receive merchandise. This example shows only the order-processing side (the “inventory depletion” process), but it is important to remember that replenishing inventory must also take place. Accessing the inventory data by using a SAS/SHARE server allows both the ordering and the receiving operations to maintain current information in the inventory file.

### Orders Information

The orders data set is a series of transactions. When a product is ordered, that event is recorded in the orders data set. This makes the orders data set a time-based record of events, which is quite different from the type of information that is maintained in the customer and the inventory data sets.

Accessing the orders data set through a SAS/SHARE server allows all order information to be captured in a single file. This enables simplified reporting and analysis of the orders data, which can be performed on current information at any time.

In this example, the data set ORDERS is updated by the users of the application. Reporting and analysis of the orders data, which is not shown here, can be accomplished with read-only access to the data set ORDERS through the server.

## The Inventory/Order System SCL Application

The following sample application is referred to in "SCL Programming Considerations" on page 50. See that section for more information about updating concurrently shared data in SCL applications.

```
/*-------------------------------------------------------------------

 *  Inventory/Order System SCL application for use with PROC FSEDIT
 *  when editing an orders data set.
 *
 * CAUTION:
 * + The deletion of a non-null order (quantity>0) results in an
 *   error message being written to the SAS log because the inventory
 *   data set will not have been updated to reflect the returned
 *   inventory.
 * + Do not issue a DELETE command to cancel a new order (not yet added to the
 *   data set ORDERS). In this case, the
 *   program will not detect a cancel or delete condition and will
 *   debit the inventory for the quantity in the cancelled order.
 *
 *  The SCL program included here is designed to run with the following
 *  set up and data set prototype:
 *
 *  The data set ORDERS has these variables:
 *
 *  o  PRODUCT   type=character  /* Product Code    */
 *  o  QUANTITY  type=numeric    /* Amount of Order */
 *
 *  The INVENTOR data set has these variables:
 *
 *  o  CODE      type=character  /* Product Code        */
 *  o  DESC      type=character  /* Product Description */
 *  o  INVENT    type=numeric    /* Stock on hand       */
 *
 *
 *
 *  For information about selecting a communications access method
```

```
 *  and server name, see Chapter 3.
 *  To start a SAS/SHARE server to access the data that is used by this
 *  example, execute these SAS statements in a SAS session:
 *
 * OPTIONS COMAMID=communications access method;
 *  LIBNAME DLIB 'physical name';
 *  DATA DLIB.INVENTOR;
 *   CODE='ABC'; DESC='PRODUCT ABC'; INVENT=100;
 *  RUN;
 *  DATA DLIB.ORDERS;
 *   PRODUCT='ABC'; QUANTITY=20;
 *  RUN;
 *  PROC SERVER  ID=server name;  RUN;
 *
 *  To create a client SAS session that you can use to execute this
 *  example, execute these SAS statements in a second SAS session:
 *
 *  OPTIONS COMAMID=communications access method;
 *  LIBNAME DLIB SERVER=optional computer name.server name;
 *  /* EDIT AND COMPILE SCL PROGRAM ON SCREEN AND RUN IT */
 *  PROC FSEDIT  DATA=DLIB.ORDERS
 *               SCREEN=DLIB.DISPLAY.ORDERS.SCREEN;  RUN;
*-----------------------------------------------------------------*/


length rc 8 ;      /* System return code storage */
length invent 8 ;  /* Current n of items inventoried*/

FSEINIT:
    /*---------------------------------------------------------------

    / Open the product control data set and save the needed variable
    / numbers. "Control term" ensures non-null deletions can be
    / detected in TERM.
    /---------------------------------------------------------------*/

    codeid=open('dlib.inventor','U');
    vdesc=varnum(codeid,'desc');
    vinvent=varnum(codeid,'invent');
    control term;
    return;

INIT:
    /*---------------------------------------------------------------

    / Save initial order values for later.  For a pre-existing order,
    / get the inventory info (item description) for the display, and
    / do not forget to unlock the record.  Also prohibit *changing*
    / the product code on a pre-existing order by using the FIELD
    / function.
    /---------------------------------------------------------------*/

    _msg_=' ';
    sav_prod=product; sav_quan=quantity;
    if (obsinfo('new')) then do;
       oldorder=0;
```

```
      rc=field('unprotect','product');
      if (product=' ') then link needcode;
      return;
    end;
  oldorder=1;
  link getrec;
  rc=unlock(codeid);
  rc=field('protect','product');
  return;

MAIN:
  /*-----------------------------------------------------------------

  / For a change in quantity or for a new order, fetch (and lock) the
  / inventory record, validate the request, and update the
  / inventory data set.  In either case, if all operations succeed,
  / issue a SAVE command in the primary data set so that the data set
  / cannot be made out-of-sync with the inventory due to a
  / subsequent CANCEL command from the user.
  /-----------------------------------------------------------------*/


  if (_STATUS_='C') then return;

  else if (product=' ') then link needcode;

  else if (sav_quan^=quantity or ^oldorder) then do;

     /* Try to lock inventory record to update. */
      loop_cnt=0;
  lokloop: loop_cnt=loop_cnt+1;
      link getrec;
      if (not gotrec) then return;
      if (rc=%sysrc(_swnoupd)) then do;
         if (loop_cnt<500) then goto lokloop;
         _msg_='Error: Product was locked.';
         erroron product;
         return;
       end;

     /* Check and debit the inventory. */
      link chkquan;
      if (not quanok) then goto unlok;
      invent=invent-quantity;
      call putvarn(codeid,vinvent,invent);
      rc=update(codeid);
      if (sysrc()>0) then do;
         _msg_=sysmsg();
         erroron product;
         goto unlok;
       end;

      /* Force FSEDIT to save the observation so that */
      /* the primary data set will be up-to-date now. */
      call execcmd('save;');
```

```
        /* In case user did not leave observation,      */
        /* clarify that this order is saved.            */
        sav_prod=product; sav_quan=quantity;
        oldorder=1;
    unlok: rc=unlock(codeid);
    end;
    return;

getrec:
    /*-----------------------------------------------------------------
    / Usually, this section fetches the record of the inventory
    / data set that you want.  If it is successful, 'gotrec'
    / will have value 1; else, 0.  This section leaves the fetched
    / record locked.
    /-----------------------------------------------------------------*/

    gotrec=0;
    rc=WHERE(codeid,"code='"||product||"    '");
    if (rc>0) then do;
       _msg_='WHERE: '||sysmsg();
       erroron product; return;
     end;
    rc=FETCH(codeid);
    if (rc>0) then do; /* Error! */
       _msg_='FETCH: '||sysmsg();
       erroron product; return;
     end;
    else if (rc=-1) then do;

       /* Product not found but no error. */
       _msg_='ERROR: The product code is invalid. Please re-enter';
       erroron product; return;
     end;
    else gotrec=1;
    desc=getvarc(codeid,vdesc);
    invent=getvarn(codeid,vinvent);
    return;

chkquan:  /* Check the amount available */
    /*-----------------------------------------------------------------

    / This section checks that the available inventory is sufficient
    / for the quantity that is being requested.  If so, 'quanok' will
    / have value 1; else, 0.  This section may modify 'invent' if a
    / quantity change is being verified.
    /-----------------------------------------------------------------*/

    quanok=1;
    /* If just a quantity change, add back old quantity. */
    if (oldorder) then invent=invent+sav_quan;
    if (quantity=0) then do;
       _msg_='This order is null due to a zero quantity';
       cursor quantity;
     end;
    else if (quantity>invent) then do;
       _msg_='ERROR: Available stock is ' || put(invent,best.);
```

```
      erroron quantity;
      quanok=0;
    end;
  return;

needcode:
  /*-----------------------------------------------------------------

  / Ask user to enter product code. Set ERRORON to prevent exiting
  / the observation.
  /-----------------------------------------------------------------*/

  _msg_='Please enter a product code';
  desc=' ';
  erroron product;
  return;

TERM:
  /*-----------------------------------------------------------------
  / For safety, check if the user accidentally deleted a non-null
  / observation, which we are leaving, and log an error message if so.
  /-----------------------------------------------------------------*/

  if (oldorder & sav_quan & obsinfo('deleted')) then
     put 'ERROR: Order consisting of ' sav_quan
         'units of product number ' sav_prod 'has been deleted.';
  return;

FSETERM:
  /*-----------------------------------------------------------------

  / Termination: Close the lookup data set if it was
  / successfully opened.
  /-----------------------------------------------------------------*/

  if (codeid>0) then rc=close(codeid);
  return;
```

*Appendix 5*

# SAS/SHARE Cross-Version Issues, SAS 9.3



## Limitations of Cross-Version Client/Server Access

- SAS/SHARE 9 clients and servers can communicate with SAS/SHARE 8 clients and servers.
- SAS/SHARE 8 clients and servers can communicate with SAS/SHARE 6 clients and servers.
- SAS/SHARE 9 clients and servers cannot communicate with SAS/SHARE 6 clients and servers.

The following topics explain how to accommodate these limitations.

# Consequences of a Client/Server Upgrade to SAS 9.3

## Upgrade: Definition

An upgrade is the process of installing a later version of SAS software over an existing version of SAS software at the customer site (for example, from SAS 8.2 to SAS 9.3). A system manager at your site is responsible for performing the upgrade.

The result of an upgrade can affect the ability of SAS/SHARE clients and servers to connect and share data.

## Data Migration

Accessing your data is a primary concern after upgrading to a new version of SAS. If the server (and clients) have been upgraded to SAS 9.3, and you want to make all new I/O engine features available, you can migrate the data to the release that the server runs. For complete details about migration, see **http://support.sas.com/rnd/migration**.

## Access Restrictions Following an Upgrade to SAS 9.3

If you do not migrate your SAS data and applications to the new version of SAS, you will be accessing SAS files and using SAS applications in a cross-version environment. Therefore, it is important to be aware of any restrictions when operating in a cross-version environment. The following tables identify the combinations of SAS/SHARE clients and servers, the SAS version that they run, and their ability for access.

***Table A5.1*** *Results of a SAS/SHARE Server Upgrade*

| Version of SAS/SHARE | SAS/SHARE Client to Server Access Restrictions after an Upgrade from One SAS Release to Another SAS Release | | |
|---|---|---|---|
| | SAS 6 to SAS 8 | SAS 6 to SAS 9 | SAS 8 to SAS 9 |
| SAS 6 | Yes* | No** | No** |
| SAS 8 | Yes* | Yes* | Yes* |
| SAS 9 | Required*** | Required*** | Yes* |

* Yes. After SAS has been upgraded, the client and server can connect and exchange data.

** No. After SAS has been upgraded, the client and server cannot connect.

*** Required. If SAS is not upgraded, the SAS application cannot run.

Some example results of a SAS/SHARE server upgrade are:

- If a SAS/SHARE server is upgraded from SAS 6 to SAS 9.3, a SAS/SHARE client that runs SAS 6 cannot access the SAS/SHARE server.
- If a SAS/SHARE client that runs SAS 9.3 and uses applications that are written in SAS 9.3 wants to access a SAS/SHARE server that runs SAS 6, the SAS 6

SAS/SHARE server must be upgraded to SAS 9.3. If the SAS/SHARE server is not upgraded, the SAS/SHARE client's SAS 9.3 applications cannot run.

Some example results of a SAS/SHARE client upgrade are:

- If a SAS/SHARE client is upgraded from SAS 8 to SAS 9.3, a SAS/SHARE server that runs SAS 9.3 can be accessed by the SAS/SHARE client.
- If a SAS/SHARE client that runs SAS 6 wants to access a SAS/SHARE server that runs SAS 9.3 and uses applications that are written in SAS 9.3, the SAS 6 SAS/SHARE client must be upgraded to SAS 9.3. If the SAS/SHARE client is not upgraded, the SAS/SHARE server's SAS 9.3 applications cannot run on the client.

# Observations and Variables: SAS 9.3 and SAS 8 Differences

## *Resolving the Number of Observations and Variables Supported across Versions*

More observations and variables are supported in SAS 9.3 than in SAS 8.

For example:

- If a SAS 9.3 client accesses SAS files on a SAS 8 server, the client can access as many observations and variables as the server.
- If a SAS 8 client accesses SAS files on a SAS 9.3 server, the client cannot access as many observations and variables as the server can store in the file or supply to the client.

To resolve differences between the number of observations and variables that are supported in a cross-version environment, SAS/SHARE will support the older version. For example, a SAS 9.3 server will not present more variables or observations to a SAS 8 client than the client is capable of accessing. Conversely, SAS 8 clients might not be able to access all the variables or observations in SAS files that are on a SAS/SHARE 9.3 server.

## *Number of Variables Supported*

The maximum number of variables that can be defined in a SAS data set and accessed by a SAS/SHARE client varies according to the SAS version that is used. The number of variables that can be defined and accessed in a data set is greater in SAS 9.3 than in SAS 8.

Consequently, a SAS/SHARE 9.3 client can access all the variables in a SAS 9.3 data set that is on a SAS 9.3 server. Also, a SAS/SHARE 8 client can access all the variables in a SAS 8 data set that is on a SAS 8 server. However, a SAS/SHARE 8 client can access only a maximum of 32,767 variables in a SAS 9.3 data set that is on a SAS/SHARE 9.3 server. Attempts to exceed this limit will result in failure.

## *Number of Observations Supported*

The number of observations in a file is a consideration only if a SAS 8 client must access observations by number (for example, if the POINT= option in the SET statement is

used). A SAS 8 SAS client that reads through a SAS file sequentially is not limited by the number of observations in the file.

A SAS 9.3 client that accesses SAS files on a SAS 8 server can randomly access any observation.

# SAS Files Access in a Mixed Client/Server Environment

## *SAS File Format: Definition*

A SAS file format is the collection of attributes of a SAS file that are specific to the SAS release. The following are different file formats:

- SAS 9.3
- SAS 8
- SAS 6

The following are characteristics of newer file formats (SAS 9.3 and SAS 8):

- long variable names
- long variable labels
- long data set labels
- integrity constraints
- data set generations

For details about newer file formats, see the topic about comparing 9.3 to earlier releases in *SAS Language Reference: Concepts*.

## *Client/Server Access to Version-Specific SAS Files*

Regardless of the version of SAS that the SAS/SHARE client or the SAS/SHARE server runs, the server automatically selects the correct engine for the format of the library. An automatic engine selection means that newer clients can access older (SAS 6) SAS files on newer servers (and older clients can access newer files when the files don't use new features). To override the server's automatic engine selection, a SAS client can specify the RENGINE= option in the LIBNAME statement to tell the server which engine to use. For example, a SAS 8 client can use the V6 remote engine to access a SAS 6 file on a SAS 8 server.

For example:

```
libname grades 'SAS-data-library' server=shr1 rengine=V6;
```

However, a SAS/SHARE client cannot access a SAS 9.3 file on a SAS 8 or a SAS 6 server. Nor can a SAS/SHARE client access a V8 file on a V6 server.

A SAS/SHARE 9.3 server can use any SAS library or view engine to search for and retrieve the SAS files of the specified version. The capabilities of each engine when it is used by a SAS/SHARE server are the same as the engine's capabilities when it is used in a single-user SAS session. For details about the capabilities of each engine in a SAS 9.3 session, see the topic about compatibility in *SAS Language Reference: Concepts*.

### SAS 8 Clients Accessing SAS Files Created on SAS 9.3 Servers

Consider the following:

- The SAS/SHARE server has been upgraded to SAS 9.3.
- SAS 9.3 SAS files are being created on the SAS 9.3 server.
- A SAS 8 client has not been upgraded to SAS 9.3.
- Until the SAS 8 client is upgraded to SAS 9.3, the client needs to access the SAS files that have been created on the SAS 9.3 server.
- The SAS 8 client cannot access the SAS 9.3 SAS files on the SAS 9.3 server if the files contain SAS 9.3 features, such as long format names.

How can the SAS 8 client, which has not yet upgraded to SAS 9.3, but must continue client/server processing, be accommodated?

As a temporary measure, SAS files on the SAS 9.3 server can be created in SAS 8 format to ensure that SAS 8 clients can access them.

In the SAS application that creates a SAS file, specify the VALIDFMTNAME= option to ensure that format names are restricted to the length that is supported in SAS 8. For more information, see “VALIDFMTNAME= System Option” in *SAS System Options: Reference* in *SAS System Options: Reference*.

# Glossary

**access descriptor**
a SAS/ACCESS file that describes data that is managed by SAS, by a database management system, or by a PC-based software application such as Microsoft Excel, Lotus 1-2-3, or dBASE. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors.

**access method**
See communications access method.

**architecture**
the manner in which numeric data and character data are represented internally in a particular operating environment. Architecture encompasses standards or conventions for storing floating-point numbers (IEEE or IBM 390); for character encoding (ASCII or EBCDIC); for the ordering of bytes in memory (big Endian or little Endian); for word alignment (4-byte boundaries or 8-byte boundaries); and for data-type length (16-bit, 32-bit, or 64-bit).

**authentication**
See client authentication.

**batch mode**
a noninteractive method of running SAS programs by which a file (containing SAS statements along with any necessary operating system commands) is submitted to the batch queue of the operating environment for execution.

**catalog**
See SAS catalog.

**catalog entry**
See SAS catalog entry.

**client authentication**
the process of verifying the identity of a person or process for security purposes.

**communications access method**
an interface between SAS and the network protocol or interface that is used to connect two operating environments. Depending on the operating environments, SAS/SHARE and SAS/CONNECT use either the TCP/IP or XMS communications access method.

**concurrent**
pertaining to the simultaneous use of resources by multiple users or applications.

**control level**
one of the determinants in the kind of lock that a task obtains on a SAS data set or on an observation in the data set. The control level specifies how other SAS tasks can access the SAS data set concurrently.

**Cross-Memory Services**
See XMS.

**data set**
See SAS data set.

**DATA step view**
a type of SAS data set that consists of a stored DATA step program. A DATA step view contains a definition of data that is stored elsewhere; the view does not contain the physical data. The view's input data can come from one or more sources, including external files and other SAS data sets. Because a DATA step view only reads (opens for input) other files, you cannot update the view's underlying data.

**data value**
a unit of character, numeric, or alphanumeric information. This unit is stored as one item in a data record, such as a person’s height being stored as one variable (namely, a column or vertical component) in an observation (row).

**data view**
See SAS data view.

**database management system**
a software application that enables you to create and manipulate data that is stored in the form of databases. Short form: DBMS.

**DBMS**
See database management system.

**encryption**
the act or process of converting data to a form that is unintelligible except to the intended recipients.

**engine**
a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

**entry type**
a characteristic of a SAS catalog entry that identifies the catalog entry's structure and attributes to SAS. When you create a SAS catalog entry, SAS automatically assigns the entry type as part of the name.

**external file**
a file that is created and maintained by a host operating system or by another vendor's software application. An external file can read both data and stored SAS statements.

**file reference**
See fileref.

**fileref**
a name that is temporarily assigned to an external file or to an aggregate storage location such as a directory or a folder. The fileref identifies the file or the storage location to SAS.

**global option**
an option that affects the processing of an entire SAS program or interactive SAS session from the time the option is specified until it is changed. Examples of items that are controlled by SAS system options include the appearance of SAS output, the handling of some files that are used by SAS, the use of system variables, the processing of observations in SAS data sets, features of SAS initialization, and the way SAS interacts with your host operating environment.

**index**
a component of a SAS data set that enables SAS to access observations in the SAS data set quickly and efficiently. The purpose of SAS indexes is to optimize WHERE-clause processing and to facilitate BY-group processing.

**interactive line mode**
a method of running SAS programs in which you enter one line of a SAS program at a time at the SAS session prompt. SAS processes each line immediately after you press the ENTER or RETURN key. Procedure output and informative messages are returned directly to your display device.

**interface view engine**
a type of SAS engine that SAS/ACCESS software uses to retrieve data from files that have been formatted by another vendor's software. Each SAS/ACCESS interface has its own interface view engine, which reads the interface product data and returns the data in a form that SAS can understand (that is, in a SAS data set).

**Internet Protocol Version 4**
See IPv4.

**Internet Protocol Version 6**
See IPv6.

**IP address**
a unique network address that is assigned to each computer that is connected to the Internet. The IP address can be specified in either of two formats: Internet Protocol Version 4 (IPv4) or Internet Protocol Version 6 (IPv6). The IPv4 format consists of four parts in dot-decimal notation, as in 123.456.789.0. The IPv6 format can consist of up to eight groups of four hexadecimal characters, delimited by colons, as in FE80:0000:0000:0000:0202:B3FF:FE1E:8329.

**IPv4**
a protocol that specifies the format for network addresses for all computers that are connected to the Internet. This protocol, which is the predecessor of Internet Protocol Version 6, uses dot-decimal notation to represent 32-bit address spaces. An example of an Internet Protocol Version 4 address is 10.23.2.3. Short form: IPv4.

**IPv6**
a protocol that specifies the format for network addresses for all computers that are connected to the Internet. This protocol, which is the successor of Internet Protocol Version 4, uses hexadecimal notation to represent 128-bit address spaces. The format can consist of up to eight groups of four hexadecimal characters, delimited by colons, as in FE80:0000:0000:0000:0202:B3FF:FE1E:8329. As an alternative, a

group of consecutive zeros could be replaced with two colons, as in FE80::0202:B3FF:FE1E:8329. Short form: IPv6

**library engine**
an engine that accesses groups of files and puts them in the correct form for processing by SAS utility windows and procedures. A library engine also determines the fundamental processing characteristics of the library and presents lists of files for the library directory.

**library reference**
See libref.

**libref**
a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

**line mode**
See interactive line mode.

**log**
See SAS log.

**member name**
a name that is assigned to a SAS file in a SAS library.

**member type**
a SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, AUDIT, DMBD, DATA, CATALOG, FDB, INDEX, ITEMSTOR, MDDB, PROGRAM, UTILITY, and VIEW.

**noninteractive mode**
a method of running SAS programs in which you prepare a file of SAS statements and submit the program to the operating system. The program runs immediately and comprises your current session.

**noninteractive processing**
See noninteractive mode.

**observation**
a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains either one data value or a missing-value indicator for each variable.

**open mode**
the way in which a SAS task accesses and operates on a member in a SAS library. There are three open modes for SAS files: input, update, and output.

**operating environment**
a computer, or a logical partition of a computer, and the resources (such as an operating system and other software and hardware) that are available to the computer or partition.

**port**
in a network that uses the TCP/IP protocol, an endpoint of a logical connection between a client and a server. Each port is represented by a unique number.

**PROC SQL view**
a SAS data set that is created by the SQL procedure. A PROC SQL view contains no data. Instead, it stores information that enables it to read data values from other files, which can include SAS data files, SAS/ACCESS views, DATA step views, or other PROC SQL views. The output of a PROC SQL view can be either a subset or a superset of one or more files.

**RDBMS**
a database management system that organizes and accesses data according to relationships between data items. The main characteristic of a relational database management system is the two-dimensional table. Examples of relational database management systems are DB2, Oracle, Sybase, and Microsoft SQL Server.

**relational database management system**
See RDBMS.

**REMOTE engine**
a SAS library engine that enables a client to access data on a server.

**Remote Library Services**
a feature of SAS/SHARE and SAS/CONNECT software that enables you to read, write, and update remote data as if it were stored on the client. RLS can be used to access SAS data sets on computers that have different architectures. RLS also provides read-only access to some types of SAS catalog entries on computers that have different architectures. Short form: RLS.

**Resource Measurement Facility**
a feature of the z/OS and OS/390 operating systems that records information about each job that is processed. Short form: RMF.

**RLS**
See Remote Library Services.

**RMF**
See Resource Measurement Facility.

**SAS catalog**
a SAS file that stores many different kinds of information in smaller units called catalog entries. A single SAS catalog can contain different types of catalog entries.

**SAS catalog entry**
a separate storage unit within a SAS catalog. Each entry has an entry type that identifies its purpose to SAS.

**SAS console log**
a file that contains information, warning, and error messages if the SAS log is not active. The SAS console log is normally used only for fatal system initialization errors or for late-termination messages.

**SAS data file**
a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data.

**SAS data set**
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

**SAS data view**
a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. Short form: data view.

**SAS library**
one or more files that are defined, recognized, and accessible by SAS and that are referenced and stored as a unit. Each file is a member of the library.

**SAS log**
a file that contains a record of the SAS statements that you enter, as well as messages about the execution of your program.

**SAS Management Console**
a Java application that provides a single user interface for performing SAS administrative tasks.

**SAS Metadata Repository**
a container for metadata that is managed by the SAS Metadata Server.

**SAS Metadata Server**
a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories.

**SAS task**
a logical process that is executed by a SAS session. A task can be a procedure, a DATA step, a window, or a supervisor process.

**SAS/ACCESS view**
a type of file that retrieves data values from files that are stored in other software vendors' file formats. You use the ACCESS procedure of SAS/ACCESS software to create SAS/ACCESS views.

**SAS/CONNECT client**
a SAS session that receives services, data, or other resources from a specified server. The server can run on the same computer as the client or on a different computer (across a network).

**SAS/CONNECT server**
a SAS session that delivers services, data, or other resources to a requesting client. The server can run on the same computer as the client, or on a networked computer.

**SAS/SECURE**
an add-on product that uses the RC2, RC4, DES, and TripleDES encryption algorithms. SAS/SECURE requires a license, and it must be installed on each

computer that runs a client and a server that will use the encryption algorithms. SAS/SECURE provides a high level of security.

**SAS/SHARE client**
a SAS/SHARE session that acts as a client. The user who runs a SAS/SHARE client accesses data on a SAS/SHARE server through Remote Library Services (RLS).

**SAS/SHARE server**
the result of an execution of the SERVER procedure, which is part of SAS/SHARE software. A server runs in a separate SAS session that services users' SAS sessions by controlling and executing input and output requests to one or more SAS libraries.

**SAS/SHARE server library**
a SAS library that has been defined to a SAS/SHARE server. The SAS/SHARE server controls access to the library.

**SASProprietary algorithm**
a fixed encoding algorithm that is included with Base SAS software. The SASProprietary algorithm requires no additional SAS product licenses. It provides a medium level of security.

**server library**
See SAS/SHARE server library.

**server session**
a SAS session that runs in a special mode on a server. No log messages or output are displayed on the server. Instead, the results of a server session are transmitted back to the log file and output files on the client.

**services file**
a file that contains a list of service names and the TCP/IP ports that are mapped to those services. The services file is stored on both the SAS client and the SAS server. The UNIX services file is located in /etc/services. A service can be specified for any of the following: a SAS/CONNECT spawner, a SAS/SHARE server, an MP CONNECT pipe, and a firewall server.

**SMP**
See symmetric multiprocessing.

**SQL**
See Structured Query Language.

**SSL (Secure Sockets Layer)**
a protocol that provides network security and privacy. SSL uses encryption algorithms RC2, RC4, DES, TripleDES, and AES. SSL provides a high level of security. It was developed by Netscape Communications.

**Structured Query Language**
a standardized, high-level query language that is used in relational database management systems to create and manipulate objects in a database management system. SAS implements SQL through the SQL procedure. Short form: SQL.

**symmetric multiprocessing**
a hardware and software architecture that can improve the speed of I/O and processing. An SMP machine has multiple CPUs and a thread-enabled operating

system. An SMP machine is usually configured with multiple controllers and with multiple disk drives per controller. Short form: SMP.

**TCP/IP**
an abbreviation for a pair of networking protocols. Transmission Control Protocol (TCP) is a standard protocol for transferring information on local area networks such as Ethernets. TCP ensures that process-to-process information is delivered in the appropriate order. Internet Protocol (IP) is a protocol for managing connections between operating environments. IP routes information through the network to a particular operating environment and fragments and reassembles information in transfers.

**thin client**
a computer that is deployed across a network, and is dependent on a server for much of its functionality. In contrast to the more independent rich client machine, thin clients share resources with other computers, thereby reducing the costs for software and support.

**threaded processing**
processing that is performed in multiple threads in order to improve the speed of CPU-bound applications.

**TLS**
the successor to Secure Sockets Layer (SSL) V3.0. The Internet Engineering Task Force (IETF) adopted SSL V3.0 as the de facto standard, made some modifications, and renamed it TLS. TLS is virtually SSLV3.1. Short form: TLS.

**Transport Layer Security**
See TLS.

**view**
a definition of a virtual data set that is named and stored for later use. A view contains no data; it merely describes or defines data that is stored elsewhere.

**view descriptor**
a SAS/ACCESS file that defines part or all of the DBMS data that is described by an access descriptor.

**work task**
a SAS/SHARE server resource that consists of a pair of lightweight threads that service requests from SAS/SHARE clients. More work tasks enable the SAS/SHARE server to service more asynchronous requests.

**XMS**
a cross-task communication interface that is part of z/OS. XMS is used by programs that run within a single z/OS operating environment. XMS is also the name of the SAS communications access method that uses XMS for client/server communication. Short form: XMS.

# Index

## D



## E

## M



## N



## O

## S

## T