The concept of one SAS execution communicating with another SAS execution is certain to be new to many computer users. For many years operating systems designers have made great efforts to ensure that each user's program can be completely unaware of the existence of every other user's program. SAS/SHARE software turns this around and introduces you to communication between programs; the formal name of this is interprocess communication.

- software transparently executed by a user's SAS program that reads and writes data via a communication path with the SAS server instead of directly to disk or tape storage.

When a DATA or PROC step in your SAS program attempts to read an observation from a shared SAS data set, your SAS execution transmits a request to the SAS server for the observation you want to read. Then the SAS server reads that observation from the disk on which your SAS library resides and transmits that observation to your SAS execution. Your DATA or PROC step then processes that observation.

- a SAS procedure for the administration of one or more SAS servers.

The user that controls a SAS server is called the server administrator. The server administrator does not interact with a SAS server execution directly; instead, he executes a SAS procedure that communicates with a server via a communication path. The SAS procedure used by a server administrator is PROC OPERATE, and it communicates with a SAS server much like other SAS procedures and the DATA step do. The difference between PROC OPERATE and other SAS procedures and the DATA step is that PROC OPERATE transmits commands to the SAS server and receives responses to those commands from the server instead of transmitting and receiving data stored in SAS files. The server administrator uses PROC OPERATE commands to request information from the server and to change the status of various resources controlled by the server.
The SAS Programmer's View of SAS/SHARE Software

Your SAS execution does not directly access a SAS library on disk when that library is shared through a server. Instead, your SAS execution and the executions of the other SAS users concurrently reading or updating that library all communicate with the SAS server. The SAS server manages requests from users that conflict with requests from other users so that those who want to read data and those who want to update data can do so safely and easily.

Since the DATA and PROC steps in your SAS program can read and write data by communicating with a SAS server, you do not execute a special SAS/SHARE PROC in your SAS program to "check out" a data set for your use. It also means that you do not make your own copy of a SAS data set you want to share concurrently with other SAS users.

Instead, at the beginning of your SAS program you define the SAS library(ies) you want to share concurrently with other users through a server. Then the DATA and PROC steps in your SAS program use the communication software within the SAS system to exchange data with the SAS server. Your DATA and PROC steps are programmed no differently when processing data in a shared SAS library.

SAS data libraries are a hierarchy. The highest level of the hierarchy is the SAS library, which is a collection of one or more of SAS data sets. The second level of the hierarchy is the SAS data set, which consists of two sections — the header area and the data area. The data area is made up of blocks of observations; these "blocks" are the third level of the hierarchy. The fourth level of the hierarchy is the individual observation, which is a collection of variables.

To allow users to concurrently update SAS libraries, the SAS server must control access at each of these levels of the hierarchy. Each level of the hierarchy is called a lock level, and the kind of access a user has at each lock level is called a control state. SAS/SHARE software chooses a control state for each lock level based on what each step in a user's SAS program attempts to do. In nearly all cases, SAS programmers do not need to be aware of the controls their SAS programs request.

Examples of lock level or control level are, "member" and "record". Examples of control state are "exclusive" and "disallow updates".

However, when the default control states do not meet the requirements of a particular SAS program, a SAS programmer may be able to use the data set option CNTLLEV= to change the controls for the particular application. This option requests the server to set a control level for a SAS program step, which is a combination of lock level and control state. The lock level is specified on the CNTLLEV= option and the control state is derived from the user's SAS program statements (SET, DATA=, etc.). For these cases, the SAS/SHARE User's Guide, Version 5 Edition defines the control levels and control states, shows the default values for SAS procedures and DATA step statements, and tells you how to adjust the type of control your SAS program requests.

At times, attempts to concurrently access SAS data sets through a SAS server will fail. Noninteractive SAS procedures ordinarily detect such failures at the beginning of a SAS program step; the procedure will terminate in a manner similar to the way a SAS program step that requires an input data set terminates when the input data set is not found. Interactive SAS procedures can usually handle concurrent access failures in ways that allow the user more choices of what to do next.

An "deliberate" kind of access failure can occur when the SAS server is stopped by a server administrator while users are concurrently accessing data through it. When this happens, the users' DATA steps and SAS procedures, except FSEDIT, terminate abnormally. FSEDIT handles this problem by displaying an informative message on the user's edit screen.

This discussion is limited to the sharing of SAS data sets. Other kinds of SAS files, such as FSLETTER catalogs, FSCALC spreadsheets, and PROC BUILD catalogs, cannot be updated through a SAS server in this release of SAS/SHARE software.

An Example of Using SAS/SHARE Software Figure 1 shows a simple SAS program that accesses a shared SAS library, creates a data set in that library, and then gives up access to the library.

```
1 libname gollum server=share;
 NOTE, LIBREF SUCCESSFULLY ALLOCATED AS FOLLOWS:
 NOTE, LIBREF=GOLLUM SERVER=SHARE
2 data gollum.mordor;
3 do i = 1 to 5;
4   do j = 1 to 3;
5     x = sqrt(i) - sqrt(j);
6     output;
7   end;
8 end;
9 run;
NOTE, DATA SET GOLLUM.MORDOR HAS 15 OBSERVATIONS
 AND 3 VARIABLES.
```

Figure 1 Creating a SAS Data Set Through a Server

The first LIBNAME statement in the SAS program in Figure 1 does several things. First, it attempts to establish a communication path between the user's SAS execution and the SAS
server named SHARE. Part of the process of establishing that connection is verifying that this particular user is allowed to connect to that particular server. If a communication path is successfully established, the LIBNAME statement next asks server SHARE for access to library GOLLUM. If this user is allowed to access library GOLLUM, GOLLUM becomes a libref that is defined to the user's program as a SAS library that is shared through server SHARE. The user's SAS program can now use libref GOLLUM as it would any other libref.

The DATA step in Figure 1 creates data set MORDOR in server library GOLLUM. Because this data set is being created, this user is given a control level of “data set”, which will prevent the server from allowing other users to access the data set while this DATA step is executing.

As the DATA step creates each block of observations, that block is not written directly to disk; instead, it is transmitted from the user's SAS execution (where the DATA step is executing) to SAS server SHARE. The SAS server will write the block of observations to the SAS library on disk on behalf of the user. (When the control level is “data set”, observations are blocked to improve efficiency because doing so does not reduce concurrency.)

Now that the data set has been created in a library controlled by a SAS server, many users can concurrently update the data set. Figure 2 shows an example of the SAS program step each user might execute to update the data set through the server.

10 proc fsedit data=gollum.mordor; run;
NOTE: THE PROCEDURE FSEDIT USED 0.54 SECONDS AND 452K.

Figure 2 Using FSEDIT to update a shared SAS data set

Notice that this is a garden variety PROC FSEDIT step, exactly the same as a PROC FSEDIT step you would use to update a locally accessed SAS data set. This step and the DATA step in Figure 1 point out a very important aspect of SAS/SHARE software: You do not have to learn a new SAS programming language to use it.

Since this is a continuation of the SAS program that began in Figure 1, libref GOLLUM is still defined to the user's SAS execution as a SAS library accessed through SAS server SHARE. In fact, libref GOLLUM will retain this definition until the user's SAS session ends or until a LIBNAME statement is executed to clear the libref definition. An example of the latter is shown in Figure 3.

Figure 3 shows two SAS program steps that allow the user to make a copy of the shared data set and then do local analysis.

11 proc copy in=gollum out=work;
12 select mordor;
13 run;
NOTE: DATA SET WORK.MORDOR HAS 15 OBSERVATIONS AND 3 VARIABLES.
14 proc sort data=mordor;
15 by j;
16 run;
NOTE: DATA SET WORK.MORDOR HAS 15 OBSERVATIONS AND 3 VARIABLES.
17 data drack;
18 select mordor;
19 by j;
20 if ( first.j ) then y = i + j - x;
21 else y = i - x;
22 run;
NOTE: DATA SET WORK.DRACK HAS 15 OBSERVATIONS AND 4 VARIABLES.
23 libname g011um clear;
NOTE: LIBREF GOLLUM SUCCESSFULLY CLEARED

Figure 3 Analyzing a Data Set With a BY Statement

Look first at the DATA step in Figure 3 and notice that it contains a BY statement. Since the other users accessing the data set could change the value of variable J in any observation in the data set, it would not be safe to execute this DATA step using the shared data set directly as input. Instead, the SAS programmer should make a private, “snapshot” copy of the shared data set, sort the private copy of the data set, and then execute the DATA step (with its BY statement).

The PROC COPY step in Figure 3 makes the private “snapshot” copy. PROC COPY uses a control level of “block”, which allows it to read observations while users are updating them. However, since the values of the variables in the observations have no effect on PROC COPY processing, it does not matter whether users are changing values of particular variables in particular observations. Once the local copy of the data set is made, the DATA step can be executed safely.

After the DATA step has finished executing, the LIBNAME statement will execute. The CLEAR option in this LIBNAME statement releases the user's access to library GOLLUM and disconnects the user's communication path to server SHARE. It also nullifies the definition of libref GOLLUM to the user's SAS program. If the user refers to libref GOLLUM in a later SAS program step without another LIBNAME statement to redefine libref GOLLUM, the SAS system will attempt to use GOLLUM to directly access a (presumably different) SAS library on disk.
Applications of SAS/SHARE Software

Now you know the rules of concurrent data access and have seen how some of the DATA step statements and SAS procedures access shared data. Since many of you are SAS programmers, I would like to rearrange the information presented above into a form more directly applicable to the questions a SAS programmer would ask in the process of developing an application to take advantage of shared data access.

The highest level of concurrency is achieved by control at the observation (record) level. This control level is used by the FSEDIT and APPEND procedures, and allows several users to simultaneously update individual observations in a single SAS data set. Several SAS data sets could simultaneously be updated, each by a different group of users, through a single SAS server.

Block level control is used by PROC COPY and can be requested with the CNTLLEV=BLK data set option. The CNTLLEV=BLK option can be used to increase the concurrency of the SET statement or it can be used to decrease the concurrency of PROC FSEDIT. Block level control provides less concurrency than record level control but still provides more concurrency than data set (member) level control. Block level control results in fewer transmissions of data between users' SAS executions and the server because more data is transferred in each transmission. Block level control is useful in situations where users' SAS programs exchange a great deal of data with a SAS server.

Data set (member) level control is used by the DATA, SET, MERGE, and UPDATE statements and by the DATA= and OUT= options of most SAS procedures. The DATA statement and the OUT= option require this level of control because only one user can be allowed to access a SAS data set while it is being created or replaced. The DATA= option and the SET, MERGE, and UPDATE statements use this control level by default because many times users' SAS program steps include a BY statement. This level of control prevents the order of BY variables from changing unexpectedly.

SAS/SHARE software uses the same control mechanisms under CMS and MVS, and you should note that all concurrency control is exercised within a SAS data library. Under MVS, a SAS server can access several SAS libraries at the same time, and various levels of concurrency can occur in each library independently. Under CMS, a SAS server can also access several SAS libraries at the same time, and it doesn't matter whether the SAS libraries are all on the same minidisk or scattered across several minidisks.

The Server Administrator's View of SAS/SHARE Software

SAS/SHARE software is designed for minimum management -- your installation should not need to devote full-time resources to the administration of a SAS server. However, PROC OPERATE does allow administrative functions to be performed on a SAS server so I want to introduce you briefly to SAS server administration.

Remember that a SAS server is often a "service process," and that no human being is ordinarily logged on to a terminal "running" a SAS server "program". To examine a SAS server, a SAS user called the server administrator uses a SAS/SHARE procedure, PROC OPERATE, to ask one or more SAS servers questions and to give those servers commands. A server administrator can do things like allow or disallow access to certain servers by certain users, stop servers gently or quickly, and perform other administrative functions.

PROC OPERATE establishes a communication path with the server very much like other SAS users' programs do. The PROC OPERATE user sends data to and receives data from the server just like users of other SAS DATA and PROC steps. The only difference is that instead of observations, PROC OPERATE sends commands to the server and receives data to display to the server administrator.

To manage a SAS server, the server administrator needs to know what is done through that server. SAS/SHARE software provides two sources of this information: the server's SAS log and the PROC OPERATE DISPLAY commands. The server's SAS log records many events, including when each library is accessed by a user, when data sets are opened and closed, and when conflicts occur. Whenever a server administrator executes a PROC OPERATE command, the administrator's user id and the first part of the PROC OPERATE command are recorded in the server's SAS log.

The DISPLAY commands of PROC OPERATE enable a server administrator to find out "on demand" what is happening on a SAS server. The other PROC OPERATE commands tell the server administrator the new status of a resource controlled by the SAS server (the result of the command's execution) as well as the status of the resource before the command executed.

What SAS/SHARE Software is NOT

SAS/SHARE software is not an efficient way for users to share read-only access to a SAS library. Both CMS and MVS provide more efficient ways to share a SAS library when users will only read data from the library.

SAS/SHARE software is not a full-fledged data base management system. It is not a transaction processor, and it does not allow updates to be rolled forward or rolled back. SAS/SHARE software does not require the comprehensive administrative resources demanded by a complete data base management system.

SAS/SHARE software does not control access at the variable, or field, level within each observation.

644
THE HISTORY OF SAS/SHARE SOFTWARE SINCE FEBRUARY, 1986

At SUGI 11, SAS Institute introduced a research project that was well received in the demonstration area, at a presentation, and at a birds-of-a-feather session. That research project acquired the identity "SAS/SHARE software" and was shipped later that month to several MVS installations for Alpha testing. That shipment included PROC SERVER, PROC OPERATE, code to implement communication via the cross-memory services of MVS, and code to implement the SAS LIBNAME statement. PROC OPERATE only executed an early form of the STOP SERVER command.

In April, 1986, an update tape was shipped to those test sites. That tape introduced no new function; instead, it fixed a number of errors that the testing had uncovered.

In July, 1986, the third tape was shipped to the MVS alpha test sites along with release 5.16 of the SAS system under MVS. SAS/SHARE software included a significant performance enhancement achieved by implementing multiple buffering for disk input/output in the server. A new module was shipped that implemented communication between a user and a server via VTAM (Virtual Telecommunications Access Method). New code in the SAS system implemented security checking via RACF (Resource Access Control Facility), so that data set security could be limited to the minimum allowed to the server and to the user. The documentation included with SAS/SHARE software was expanded in this release.

In October, 1986, SAS/SHARE software was released in beta test status. This was the first test release of the product under CMS and the second test release of the VTAM communication capability. PROC OPERATE was given its complete complement of commands in this release; that resulted in considerable expansion of SAS/SHARE software documentation.

In January, 1987, SAS/SHARE software was released in production status on MVS and in rollover beta test status on CMS. The VTAM communication code was shipped in "third test" status. No new functions were introduced during the beta test period; instead, all effort was directed toward improving reliability and fixing errors. As of this date, SAS/SHARE software under CMS will continue in rollover beta test status until SAS Institute is satisfied that an adequate beta test has been performed.

FUTURE DIRECTIONS OF SAS/SHARE SOFTWARE

Before I tell you what we have in mind, I want to remind you that SUGI meetings are your best opportunity to tell us what your needs are and what you would like us to do. Please talk with us and come to the birds-of-a-feather session tonight if you can. Share your ideas with us!

Integration

At the top of our list is the integration of sharing capabilities into other SAS software products. Since SAS/SHARE software was introduced late in the life of Version 5 of the SAS system, not all SAS procedures take complete advantage of concurrent data access. In future releases we hope to clean up any annoying restrictions that are the inevitable result of integration growing pains.

SAS users have already asked for more versatile control of locking. This includes the capability of holding a specified lock on a data set across several steps of a SAS program, or holding a lock on a data set during some but not all of the program statements of an interactive SAS procedure.

We also hope to allow concurrent updating of SAS files that are catalogs in addition to SAS data sets.

Connectivity

We hope to expand the connectivity of SAS/SHARE software on an ongoing basis. You can help us to do this by telling us what network configurations your site presently uses and what it plans for the future. Our first networking enhancement is our VTAM support for MVS systems in an SNA network, and we are committed to developing VM VTAM (VM SNA) communication, too. (It is relevant to note that one of the items on the SASware Ballot® requests the ability to write on CMS minidisks from MVS; using VTAM to communicate with a SAS server under CMS would provide exactly that capability.)

Our VTAM code is useful for communication between computers in an SNA network, but CPU clusters and guest operating systems in virtual machines are quite a different matter. We would like to know if you have any needs for communication in these environments.

SAS Server Enhancements

Though the capabilities provided for the server administrator have expanded greatly since the first alpha test release of SAS/SHARE software, we realize that more can be done. We have received requests from users to allow PROC OPERATE commands to be executed outside of a PROC OPERATE step in a SAS program. We are committed to implementing this capability for both MVS and CMS.

We are open to ideas for improving the output from existing PROC OPERATE commands and for adding new PROC OPERATE commands.
The addition of RACF checking in our third alpha test release was in response to requests from our test sites for improved security control over data access. Other presently implemented security controls include the PROC OPERATE STOP commands and the SERVERPW= option. The SAS server's SAS log provides the present level of auditability. As more SAS servers are installed, we will be open to your suggestions for improving the security and auditability of SAS/SHARE software.

We hope to improve the configurability of SAS/SHARE software by allowing all of the server configuration options to be specified on the PROC SERVER statement as well as in PROC OPERATE commands. We are also aware of a number of opportunities to improve users' ability to configure a SAS server under CMS. At SUGI 11, we sought your advice on server minidisks and you talked us out of some ideas we had discussed that could have caused some of your problems. We continue to seek your advice on server configurability under CMS, especially since those discussions have been most interesting.

The performance of SAS servers will increase in importance as more and more applications access SAS data sets through servers. We are looking at design-level improvements, especially taking better advantage of asynchronous I/O to disk storage and on communication paths with users. Multithreading user requests within the server would be the natural next step in a more asynchronous implementation. We have also considered moving high-level functions to the server from the user executions.

Future Releases of SAS/SHARE Software

We plan to make this software available under other operating systems. In the IBM mainframe environment, interest has been expressed in running SAS/SHARE software under DOS/VSE, and we hope to be able to execute SAS/SHARE software under VAX/VMS, AOS/VS, and PRIMOS. Those of you who are interested in concurrent update of SAS data sets in these other environments should come to our birds-of-a-feather session this evening.

CONCLUSION

This paper has attempted to cover three areas. It began with an overview of the implementation of SAS/SHARE software and how SAS programmers and server administrators use it. The concepts of concurrent update and locking were discussed, and an example application was presented.

The history of SAS/SHARE software in the past year and the possible future directions of the software in the coming years were presented. Specific areas of interest were integration with the rest of the SAS system, connectivity, the server implementation, and what future releases of the SAS system might bring.

As I have done several times in this paper, I invite those of you with questions, opinions, and advice to our birds-of-a-feather session this evening.

Thank you.

SAS and SASware Ballot are registered trademarks of SAS Institute Inc., Cary, North Carolina, United States of America. SAS/SHARE is a trademark of SAS Institute Inc.