Writing SAS Applications That Access Data in a Multiple-Update Environment
Mark Stranieri, SAS Institute Inc., Denver, CO

ABSTRACT
A multiple update environment is any situation in which a single SAS data set may be accessed by more than one process at a time and one or more of the processes may be updating the data. This paper serves as a tutorial for SAS/SHARE software which facilitates a multiple update environment. However, some of the concepts covered may also be applicable to other multiple update environments such as accessing a DBMS database through a SAS/ACCESS view.

Situations arise in multiple update environments that cannot occur in single update environments. For example:

- an observation read once may have different variable values if/when it is read a second time,
- a record that was read may not be updateable because another process has it locked,
- the nth record that a process reads in one sequential pass of the data, may not be the nth record retrieved in a second sequential pass of the data.

An application which accesses a SAS data set in a multiple update environment must be written to anticipate these and other situations.

INTRODUCTION
SAS/SHARE software allows concurrent update access to SAS files from multiple SAS user sessions. Concurrent update access is controlled by a SAS server. A SAS server is a SAS session that has an execution of Proc SERVER running. Once a SAS server is established users can reference server libraries as well as their own libraries in batch mode SAS programs, or interactively in Display Manager sessions and applications.

Concurrent update access includes the following operations.

- different users can simultaneously operate on files in the same SAS library
- different users can simultaneously operate on members of the same SAS catalog
- different users can simultaneously browse, edit and/or copy the same SAS data set.

The following tutorial begins with a section that describes the components of SAS/SHARE software. This is followed by a section that describes how to reference SAS libraries through the SAS/SHARE server. The third section discusses considerations a SAS programmer needs to make when addressing shared SAS libraries. This includes several common situations that SAS programmers may encounter in a multiple-update environment. Further, several methods are presented to handle these situations. Finally, this tutorial concludes with a review of the topics covered.

SECTION I
COMPONENTS OF SAS/SHARE SOFTWARE
SAS/SHARE software consists of

- a REMOTE engine
- PROC SERVER
- PROC OPERATE

The REMOTE engine

- is the tool that processes read/write requests between a user's SAS session and the SAS SERVER

PROC SERVER is used to

- create a SAS server (an execution of PROC SERVER is a server)
- manage multi-user access to SAS data libraries and SAS files

PROC OPERATE is used to

- control the execution of PROC SERVER
- stop a server
- start and stop server libraries
- display information about a server or a server library

**Figure 1 Server Model**

NOTE: users also have access to SAS libraries that are not server libraries (ex. WORK).
SECTION II
ACCESSING SHARED SAS LIBRARIES

Referencing SAS Servers and SAS Libraries

Programs that use SAS/SHARE software must include a LIBNAME statement to identify the SAS server that will be used to access a particular SAS library.

The server form of the LIBNAME statement is

\[ \text{LIBNAME libref 'SAS-data-library' serverid options}; \]

Where libref is a logical reference to the physical location of the SAS-data-library and serverid is the ID of the server established by an execution of Proc SERVER.

The above LIBNAME statement requires you to know the serverid associated with the SAS-data-library you are interested in referencing.

Since server administrators can add servers, change serverids, and switch libraries and users from one server to another, programs using the server form of the LIBNAME statement could easily become inaccurate.

SAS/SHARE software provides several SAS macros that allow programmers to use aliases that will transparently access the correct SAS server and SAS library. Aliases can be established by the server administrator.

After library aliases have been established, the following two SAS/SHARE macros can be used to generate an accurate server-form LIBNAME statement:

\[ \text{%SHRMACS(user,applsys=appname...);} \]
\[ \text{%LIBDEF(libref, SAS-data-library)}; \]

The %SHRMACS macro with user as the first parameter defines and compiles other SAS/SHARE macros, including %LIBDEF. The optional applsys parameter specifies the name of a member in the APPLSYS macro library. The members in the APPLSYS macro library are maintained by the server administrator, and are used to specify the SAS data libraries that are defined to specific servers. %SHRMACS generates look-up tables based on information from the APPLSYS macro library members.

The %LIBDEF macro can then generate a LIBNAME statement that specifies the correct server for the SAS-data-library based on the look-up tables generated by %SHRMACS.

For example, assume your server administrator has set up a member in the APPLSYS macro library called PERSNL. The PERSNL member contains information that associates the following SAS data libraries with the servers they are defined in.

<table>
<thead>
<tr>
<th>SAS data library</th>
<th>serverid</th>
</tr>
</thead>
<tbody>
<tr>
<td>user1.employee.sasdata</td>
<td>server1</td>
</tr>
<tr>
<td>user1.policy.sasdata</td>
<td>server1</td>
</tr>
<tr>
<td>user1.salary.sasdata</td>
<td>server2</td>
</tr>
</tbody>
</table>

Table 1 Example Shared Libraries

To write an application that is going to be referencing data sets in the USER1.EMPLOYEE.SASDATA SAS library use the following macro invocations.

\[ \text{%SHRMACS(user,applsys=persnl)} \]
\[ \text{%LIBDEF(emp, user1.employee.sasdata)}; \]

The %SHRMACS macro defines and compiles the %LIBDEF macro. The %LIBDEF macro generates the following LIBNAME statement based on information in the above table.

\[ \text{LIBNAME emp 'user1.employee.sasdata' server=server1; } \]

If you need to reference the user1.policy.sasdata SAS library later in the same program, you do not need to re-invoke %SHRMACS.

NOTE: this example was written for the MVS environment. If you are using SAS/SHARE in a different environment, see the SAS technical report for SAS/SHARE software specific to that environment.

Open Modes and Control Levels

Every step in a SAS program is considered to be a separate task. A task may attempt to obtain a member lock on a SAS data set, or a record lock on an observation of a SAS data set. The SAS server detects conflicts, and performs locks.

The type of lock attempted for a particular task is determined by a task's open mode and control level.

<table>
<thead>
<tr>
<th>open mode</th>
<th>one of three ways a task accesses and operates on a SAS data set - input, update, or output</th>
</tr>
</thead>
<tbody>
<tr>
<td>control level</td>
<td>one of two ways a task can access a SAS data set concurrently - member or record</td>
</tr>
</tbody>
</table>
Table 2: Open Modes and Control Levels

As a SAS programmer, you may need to know a task's open mode in order to safeguard your program from data integrity problems. Implementing safeguards is discussed in the next section.

To use SAS/SHARE Software in a SAS/SHARE Environment, you must be familiar with the concepts discussed in this section. This section discusses the use of SAS/SHARE libraries and the use of SAS/SHARE Software with the SAS/SHARE libraries.

Once Proc SERVER is executing, you have established a connection to the server. If you are interested in the open mode of a task, you can find out the open mode and control levels of the task by examining the server SAS log after you stop the server. To create your own server and submit statements to it, do the following.

1. Logon to your system and start SAS Display Manager.
2. Submit the following to create a server:
   ```
   proc server id=shr1;
   run;
   ```
   where SHR1 is the server ID that can be any legal SAS name.

   Once Proc SERVER is executing, you have established a server session.

3. Logon to your system using a different logon ID, and start SAS Display Manager.
4. Create a server library using the following LIBNAME statement:
   ```
   libname shr 'SAS-data-library' server=shr1;
   ```
   where SAS-data-library is an existing SAS library with SAS data sets available to operate on and SHR is a libref that can be any legal SAS name.

   After this LIBNAME statement executes successfully, you have a user SAS session capable of referencing the shared library SHR.

5. Submit and execute the tasks you are interested in finding out the open codes for from the user SAS session.

6. Stop the server using Proc OPERATE as follows:
   ```
   proc operate;
   set server shr1;
   proc operate;
   stop server shr1;
   run;
   ```

7. Examine the log of your server SAS session.

The following example SAS log demonstrates this technique.
Writing SAS Programs in a SAS/SHARE Environment

There are two questions that SAS programmers should consider when working in the SAS/SHARE environment.

1. Does the program attempt to update any data sets that other users may already have locked?
2. Can the program results be affected by other users updating data sets while the program is referencing those data sets?

If the answer to either of these questions is yes, you must develop code to safeguard the program.

NOTE: an observation in a SAS data set is locked when it is being read by a SAS procedure, a DATA step, or an SCL function.

Example 1:

Does the following code need any safeguards in a multiple update environment?

```sas
libdef emplib;
proc sort data=emplib.phone;
by city;
proc print data=emplib.phone;
run;
```

Yes, this program does need safeguards, because an error will result if Proc SORT is unable to read an observation in the data set. In addition, since other tasks can perform updates while Proc PRINT is executing, the output report may display observations that are not ordered by CITY.

There are two strategies for avoiding this type of error.

1. create a data set that is not in a shared library

```sas
libdef emplib;
proc sort data=emplib.phone out=work.phone;
by city;
proc print data=work.phone;
run;
```

This program creates a sorted copy of EMPLIBPHONE called WORKPHONE. The WORK library is a temporary library in the user SAS session.

NOTE: if a temporary data set is created and then copied back to the originating shared library, it may overwrite or undo other users' updates that occurred in the interim.

2. attempt to acquire a member lock on the data set, and perform the update only if the lock is successful.

```sas
libdef emplib;
lock emplib.phone;
data _null_; 
  %sysput('syslckrc');
  if (x ne '0') then do;
    put 'ERROR: could not obtain lock on emplib.phone; ending SAS';
    abort; 
    end;
  run;
proc sort data=emplib.phone;
by city;
proc print data=emplib.phone;
run;
lock emplib.phone clear;
```

This program attempts to lock EMP LIBPHONE. If the lock is successful, the automatic macro variable &SYSLCKRC is set to zero. The DATA step checks the value of &SYSLCKRC. If &SYSLCKRC is not equal to zero, SAS ends with an error message to the user's display. If you want to end the DATA step instead of the entire SAS session, use the use the STOP statement in place of the ABORT ABEND statement. If &SYSLCKRC is equal to zero, the program executes Proc SORT and Proc PRINT. A partial SAS log of a similar program where the lock was successful is shown below.

```
150 lock emplib.phone;
NOTE: EMP LIBPHONEDATA is now locked by you through server EMPLIB.
155 data _null_; 
156 %sysput('syslckrc');
160 if (x ne '0') then do;
162 put 'ERROR: could not obtain lock on emplib.phone; ending SAS';
165 abort; 
167 end;
170 run;
NOTE: The DATA statement used 0.03 CPU seconds and 1018.
180 proc sort data=emplib.phone;
185 by city;
NOTE: The data set EMP LIBPHONE has 26 observations and 5 variables.
NOTE: The PROCEDURE SORT used 0.03 CPU seconds and 1018.
195 proc print data=emplib.phone;
210 run;
236 lock emplib.phone clear;
NOTE: EMP LIBPHONEDATA is no longer locked by you through server EMPLIB.
```

Example 3: Example SAS Log
An alternate, and somewhat more versatile method using the SAS Macro facility follows.

```sas
%macro sortit();
libdef(emplib);
lock emplib.phone;
%if %syslcrc eq 3 %then %do;
proc sort data=emplib.phone;
by city;
proc print data=emplib.phone;
run;
lock emplib.phone clear;
%end;
%else put EMPLIBPHONE locked by another user;
%end;
%sortit
```

When you invoke the %SORTIT macro above, SAS will attempt to obtain a lock on EMP LIBPHONE. If the lock is successful, the automatic macro variable &SYSLCKRC is set to zero. The Proc SORT and Proc PRINT execute, and the macro ends. If the lock is unsuccessful, the program puts an error message into the SAS log, as shown in the following example SAS log.
Using this macro has the advantage of allowing you to conditionally execute SAS procedures or a DATA step within the macro %DO-%END group.

You can allow the program to make multiple attempts at getting a lock by putting the above code within a macro %DO-loop.

```sas
display 4 example sas log
```

When you invoke the above %SORTIT macro, SAS will attempt to obtain a lock on EMPLIBPHONE. If the lock is successful, the automatic macro variable &SYSLCKRC is set to zero, the Proc SORT and Proc PRINT execute, and the macro ends. If the lock is unsuccessful, the program will make up to 500 attempts to obtain a lock, and will put an error message in the log each time the lock is unsuccessful, as shown in the following partial example.

```sas
display 5 example sas log, page 1
```

Yes, safeguards are necessary, because any DATA step without the MODIFY statement opens the data set within a macro %DO-loop.

```sas
display 6 example sas log, page 2
```

Example 2:

Does the following code need any safeguards in a multiple update environment?

```sas
%libdef(emplib);

data emplib.phone;
  set emplib.phone;
  if upcase(city)='DENVER' then city='DENVER';
run;
```

Yes, safeguards are necessary, because any DATA step without the MODIFY statement opens the data set in the DATA statement for output. An open mode of output means no other user can access the data set at all (refer back to Table 2 for open mode definitions). The following partial SAS log demonstrates this.

```sas
display 7 example sas log
```

However, even after you rewrite this program using a MODIFY statement, an error will occur if another task has a lock on one of the observations, causing the DATA step to end. You could use one of the two strategies used in Example 1 above, or you can use a third strategy.

If your program is performing an update using the DATA step, you can take advantage of the automatic _JORD variable that stores return codes.

```sas
%libdef(emplib);

data emplib.phone;
```

1381
The DATA step checks EMPUB.PHONE sequentially. If there is not a return code indicating the observation is not available (SYSREC(SENOLCK)) and puts an appropriate message in the log. The observation number of any observation not updated will be output to the log with a message, as demonstrated in the following partial SAS log.

```
130 data emplib.phone;
131 modify emplib.phone;
if _JRC_ < 0 then do;
   replace emplib.phone;
   run;
else if _JRC_ = sysrec(SENOLCK) then
   put 'obs ' _ .. _ not replaced; record lock not available';
   else
   put 'obs ' _ .. _ not replaced';
run;
```

To specify you want views to execute in the user SAS session you must use the rmtview= no option in the server form of the LIBNAME statement as follows.

```
libname emplib '_examples.sgp19.sasdata' server=serverid;
proc sql;
update emplib.db2alemp
set _name=ucpase(_name);
```

To access SAS/SHARE servers, the user's SAS session will interpret views. The default is rmtview=no; however, after reading this tutorial, SAS programmers should have a good understanding of what SAS/SHARE software provides, how to access SAS/SHARE servers, and how to write programs that protect against situations that may arise in a multiple-update environment.

**CONCLUSION**

After reading this tutorial, SAS programmers should have a good understanding of what SAS/SHARE software provides, how to access SAS/SHARE servers, and how to write programs that protect against situations that may arise in a multiple-update environment.
REFERENCES


Thanks to Bill Brideson for help with retrieving control levels, and to Jack Wallace for conceiving this tutorial and writing the original abstract.

SAS, SAS/ACCESS, SAS/FSP, and SAS/SHARE are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. DB2 is a registered trademark of International Business Machines Cooperation. © indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.