Paper 3573-2019

Exclusive Data Set Access in a Stored Process Web Application: Macros for mutexes in Windows® OS

Richard A. DeVenezia

ABSTRACT

Data sets used in stored process web applications can experience access contention errors during high frequency usage. What can you do when attempts to use SAS® features such as the LIBNAME statement option FILELOCKWAIT=wait-time and SAS/Share libnames fail?

This paper will demonstrate how these failures can occur in Windows OS servers and how to prevent them using system mutexes. A discussion of creating a custom dynamic link library (DLL) to interact with the mutexes, macros to interact with the DLL and systematic macro usage is made. These techniques are both academically interesting and useful in actual code.

INTRODUCTION

This paper will cover several aspects of creating a stored process web application and the scenario in which mutexes would be needed. You will gain insight on the SAS Metadata server, the Management Console, defining stored processes, Proc STREAM, 3rd party HTML 5 user interface (UI) components, Ajax, MODULEN, C, mutexes.

MUTUAL EXCLUSION

The mutex object is the goal and there will be a lot a groundwork to cover to get to it. Some examples of mutual exclusion:

- Probability: "Two events are mutually exclusive if they cannot both occur at the same time."[1]
- Physics: "Pauli's exclusion principle says that two identical fermions cannot be in the same quantum state."[2]
- Programming: "A concurrency control property requirement that only one thread of execution can be in a critical section at a time."[3]
- Mutex object: "a synchronization object whose state is set to signaled when it is not owned by any thread, and non-signaled when it is owned. … Only one thread at a time can own a mutex object"[4]

Multiple threads coordinate concurrency by opening a mutex to lock it, performing their processing and closing the mutex to release the lock. An attempt to open an already an open mutex places the thread in a waiting state until the mutex lock is granted or timeout occurs. The reality checks for a SAS stored process (which is a SAS program) are as follows:

- Mutex object is a Windows system object, accessed through a WINAPI HANDLE.
- SAS code **cannot** invoke operating system calls that manipulate mutex objects.
- SAS code **can** use MODULEN to invoke DLL routines having SAS compatible signatures

STORED PROCESS WEB APPLICATION

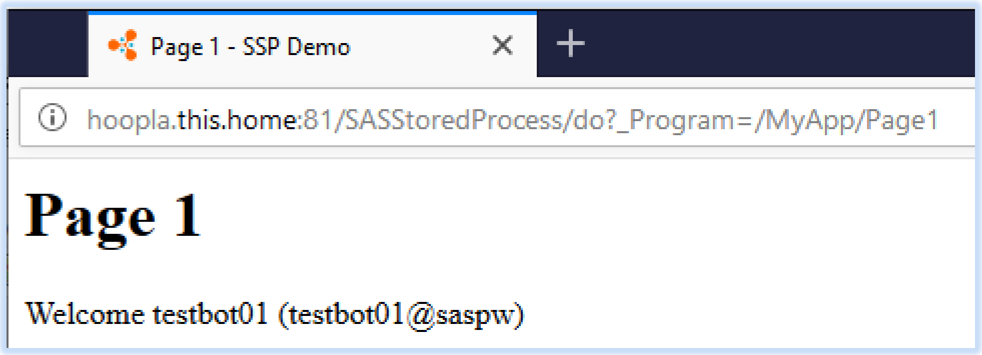
A SAS stored process can be simply described as a SAS program that is registered in a with SAS Metadata Server. The stored process is run from a client connected to a SAS Server. Some examples of clients are; SAS DMS session, Enterprise Guide, SAS Add-In for Microsoft Office, SAS Studio, custom programs written in other coding languages that connect using interfaces accessible through SAS Integrated Technologies Integrated Object Model (IOM), and most importantly, for this paper, the **SAS Stored Process Web Application**.

The SAS Stored Process Web Application is HTTP endpoint that accepts the stored process name to be run and the parameters to be passed. The phrase *"Stored Process Web Application"* stated in paper's title is to mean "a collection of stored processes whose generated output are the pages and data flows of an interactive website", not the installed SAS application used to deliver the content.

SAS SERVER PAGE

For this discussion a SAS server page (SSP) will mean any content generated by a stored process and viewed in a browser. A DATA Step in a stored process can send content destined for a browser by using the `_WEBOUT` fileref and `PUT` statements. A far less cumbersome way is to use the `STREAM` procedure.

```
proc stream outfile=_webout;
BEGIN
<html><head><title>Page 1</title></head>
<body>
<h1>Page 1</h1>
<p>Welcome &_METAPERSON (&_METAUSER);
</body>
</html>
;;;;
```


Page 1 - SSP Demo
hoopla.this.home:81/SASStoredProcess/do?_Program=/MyApp/Page1

Page 1

Welcome testbot01 (testbot01@saspw)

Figure 1. Screenshot of SAS Server Page

METADATA SERVER

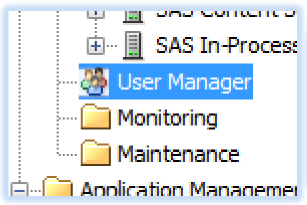
For discussion the *Stored Process Web Application* will be called **MyApp**.

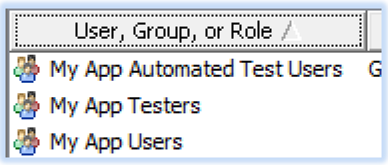
The SAS Metadata Server is the control nexus for managing all aspects of who, what, when, where and how regarding stored processes. The SAS Management Console is the administrative UI for specifying and configuring settings.

USERS AND GROUPS

The "User Manager" Plug-in is used to create new groups named **MyApp Users, My App Testers** and **My App Automated Test Users**. The automated test users are internal accounts created programmatically[5].

METADATA FOLDERS




User, Group, or Role

My App Automated Test Users

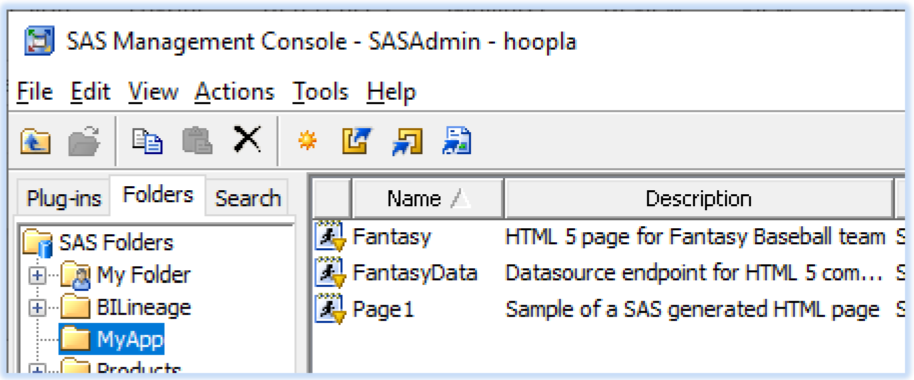
My App Testers

My App Users

The SAS programs that comprise the web application are registered with the Metadata Server in order to be accessed through the Stored Process Server. Figure 2 shows the metadata folder, "MyApp", was created to contain the stored process "Page1". This corresponds to the `_Program` path shown in the address bar in Figure 1.

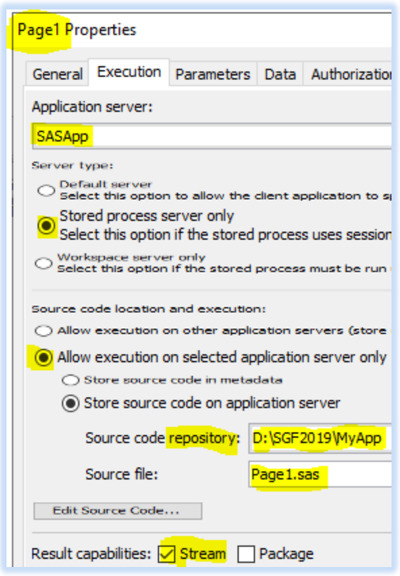
The stored process properties were edited to specify:

- The associated SAS program file and the kind of output produced ()
- The servers and users allow to run it ()


SAS Management Console - SASAdmin - hoopla

Name	Description
Fantasy	HTML 5 page for Fantasy Baseball team
FantasyData	Datasource endpoint for HTML 5 com…
Page1	Sample of a SAS generated HTML page

Figure 2. Folder and stored process in the Metadata Server


Page1 Properties

Application server: SASApp

Server type:

- ○ Default server
  Select this option to allow the client application to s
- ● Stored process server only
  Select this option if the stored process uses session
- ○ Workspace server only
  Select this option if the stored process must be run

Source code location and execution:

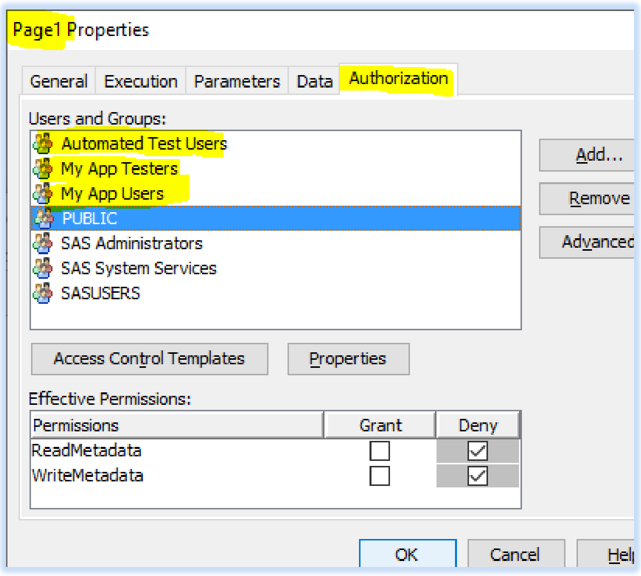
- ○ Allow execution on other application servers (store
- ● Allow execution on selected application server only
  - ○ Store source code in metadata
  - ● Store source code on application server

Source code repository: D:\SGF2019\MyApp

Source file: Page1.sas

Result capabilities: [✓ Stream] Package

Figure 3. Execution settings


Page1 Properties

Users and Groups:

Automated Test Users

My App Testers

My App Users

PUBLIC

SAS Administrators

SAS System Services

SASUSERS

Effective Permissions:

Permissions	Grant	Deny
ReadMetadata	☐	✓
WriteMetadata	☐	✓

Figure 4. Authorization settings

STORED PROCESS DEFINITION

There are some rare cases when a test user will see the message:

STP: User has insufficient authorization to access <repository> <source-file>.

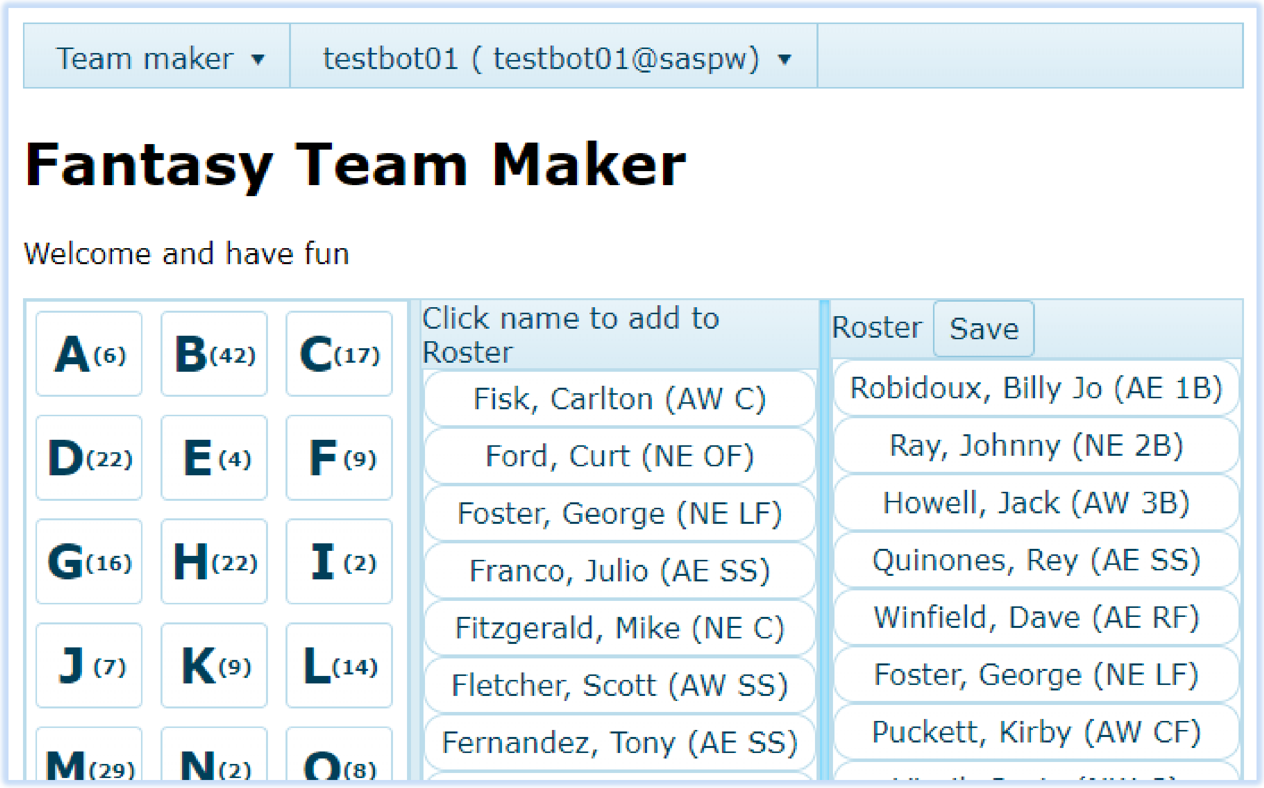
This can occur when the source file was copied into the repository through a network share, or by a developer user account. **Tip:** All source files in the repository must be readable by the **sassrv** account at the operating system level. The same policy applies to any data sets the stored process will use.

What are some of the effects of the authorization settings shown in Figure 4? PUBLIC users are denied ReadMetadata so they would see error message:

Stored process not found: /MyApp/Fantasy

MYAPP - FANTASY TEAM

For discussion suppose part of MyApp is making a roster for a Fantasy team


Team maker ▾ testbot01 (testbot01@saspw) ▾

Fantasy Team Maker

Welcome and have fun

A(6) B(42) C(17)
D(22) E(4) F(9)
G(16) H(22) I (2)
J (7) K(9) L(14)
M(29) N(2) O(8)

Click name to add to Roster

Fisk, Carlton (AW C)
Ford, Curt (NE OF)
Foster, George (NE LF)
Franco, Julio (AE SS)
Fitzgerald, Mike (NE C)
Fletcher, Scott (AW SS)
Fernandez, Tony (AE SS)

Roster Save

Robidoux, Billy Jo (AE 1B)
Ray, Johnny (NE 2B)
Howell, Jack (AW 3B)
Quinones, Rey (AE SS)
Winfield, Dave (AE RF)
Foster, George (NE LF)
Puckett, Kirby (AW CF)

Figure 5. Screenshot of Fantasy Team page

BACKEND DATA STORE

The back-end data store for the page are SAS data sets named Rosters, RosterMembers and Players. The primary keys, foreign keys and indices are shown in Figure 6 diagram.

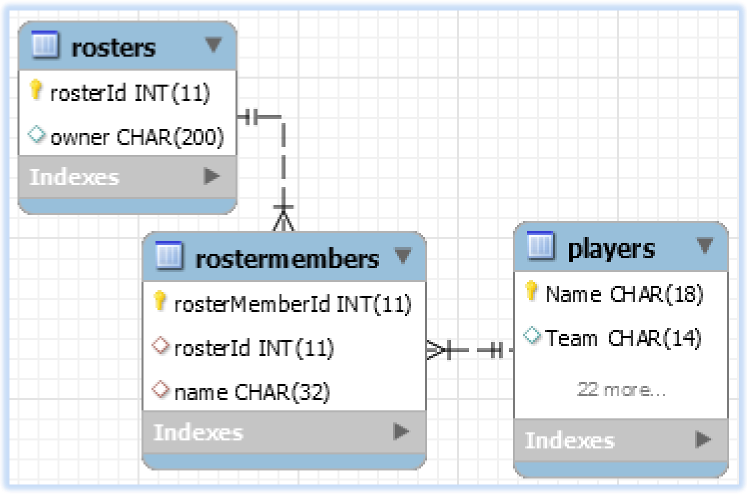


Figure 6. Entity relationship diagram (ERD)

FANTASY PAGE – A SAS SERVER PAGE

The page content is delivered by a stored process using Proc STREAM. The HTML5 page source is hand written and utilizes common web programming technologies such as jQuery and 3rd party JavaScript UI components. The page in Figure 5 was built using Menu, Splitter, ListView and Button components. The ListViews are dynamically populated and interact with a remote sever using Ajax to invoke Create, Read, Update and Delete (CRUD) service routines that pass information in JavaScript Object Notation (JSON) format.

The CRUD routines are stored processes that use the JSON Procedure. The **Fantasy** SSP will have a corresponding stored process named **FantasyData**.

PROC STREAM OUTPUT

The page HTML is sent to _webout using Proc STREAM. Standard HTML tags with ID attributes, such as `id="menu"`, are hook points where the components are installed via jQuery plug-ins invoked during the document ready closure.

```
proc stream outfile=_webout prescol;
BEGIN
<html><head>
  <title>Fantasy Team</title>
  <link rel="stylesheet" href="http://.../styles/kendo.common.min.css" />
  <link rel="stylesheet" href="http://.../styles/kendo.blueopal.min.css" />
  <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
  <script src="https://.../js/kendo.ui.core.min.js"></script>
</head>
<body>
<ul id="menu">
  <li>Team maker
    <ul>
      <li>SASGF 2019 - Dallas!</li>
    </ul>
  </li>
  <li>&_METAPERSON (&_METAUSER)       macro substitution occurs in output
…
<script>
  $(document).ready(function() {
    $("#menu").kendoMenu();             component built by plug-in
```
Code 1. Stored process code for a SAS Server Page. STREAMs HTML and JavaScript

COMPONENT DATA SOURCE

The Fantasy ListView components are:

- Letters – First letters of a player names
- Names – All the names of a selected letter
- Roster – Names selected to make a roster

The extract below shows how the HTML tag for the Splitter and ListView hooks are laid out.

```
<div id="across">
  <div class="split-pane" id="left-pane">
    <div id="letters"></div>
  </div>
  <div class="split-pane" id="center-pane">
    <div class="k-header">Click name to add to Roster</div>
    <div id="names"></div>
  </div>
  <div class="split-pane" id="right-pane">
    <div class="k-header">Roster <div id="save" class="k-button">Save</div>
<span id="roster-msg">Click name to remove</span></div>
    <div id="roster"></div>
  </div>
</div>
```
Code 2. HTML in Proc STREAM

The JavaScript that configures the data bound component specifies a URL that invokes the CRUD endpoint stored process through the SAS Stored Process Web Application.

```
    $("#letters").kendoListView({
      template: kendo.template('<div class="letter k-widget k-button" data-
letter="#:letter#"><h3>#:letter#</h3><br /><h4>(#:N#)</h4></div>'),
      dataSource: {
        schema: { type: "json", data: "letters", model: { fields: [
"letter", "N" ] }},
        transport: {
          read: {
            dataType: "json",
            url:
"&_URL?_program=&_PROGRAM.Data%str(&)operation=Read%str(&)for=letters"
          }
        }
      }
    });
```
Code 3. JavaScript source in Proc STREAM

Remember, The HTML and JavaScript code are all within a stored process source code within a Proc STREAM step. Normal macro processing rules apply and automatic macro variables[6] can be leveraged to create a URL corresponding to the **FantasyData** stored process:

- &_URL contains the absolute path such as `/SASStoredProcess/do`

- &_PROGRAM contains the stored process name such as `/MyApp/Fantasy.` Notice how Data appears after &_PROGRAM — together they are **FantasyData**.
- %str (&) masks macro ampersands, allowing them to be an ampersand (&) separating the GET parameters in the URL query string
- The components configuration property schema: describes how the returned JSON should be parsed for binding data into the component
- The `#:<field>#` token is a resolver indicator, much like SAS macros &, that causes a bound data value to appear in the at template rendering time.

The JavaScript that configures the ListView will:

- Cause Ajax to query endpoint `http: //hoopla.this.home:81/SASStoredProcess/do?_program=/MyApp/FantasyData&operation=Read&for=letters`
- Expect JSON returned to represent an object that describing a frequency table `{"letters":[{"letter":"A","N":6},{"letter":"B","N":42}…]}`

Two techniques for examining SSP activities are browser Developer Tools feature (F12) and tacking on query parameter `&_DEBUG=LOG` to the URL.

CRUD ENDPOINT - FANTASYDATA

A close examination of the configuration url: reveals the combination of parameters that the example stored process will have to handle:

	Operation=			
For=	Create	Read	Update	Delete
Letters		X		
Names		X		
Roster	X			

Table 1. Grid of CRUD Parameters

The complete CRUD implementation is not needed because the ListViews are only displaying values in the case of Letters and Names and writing for the case of Roster. The **FantasyData** endpoint is coded as a set of macros whose names correspond to the expected combinations of operation and for. The invocation of the correct macro is accomplished using a simple dispatch macro. Serialization of a data set into JSON is done with `Proc JSON`

FantasyData, stored process source code

```
%macro dispatch();                                      define the dispatch macro
  %local routine;                                       invoked at the end of STP
  %let routine = &for._&operation;
  %&routine
%mend;

%*--------------------------;                         define macro for combination
%macro Letters_Read();                                for=letters & operation=read
%*--------------------------;
  proc sql;
    create table letters as                        work.letters is frequency table
    select                                         of first letter in player name
```

```
      upcase(name) as letter length=1,
      count(*) as N
    from appdata.players
    group by letter;

  proc json out=_webout;                                      yeah! Proc JSON
    write values 'letters';                                  open {"letters":
      write open array;                                      open [
        export letters / nosastags;                        rows as name:value
      write close;                                           close ]
  run;
%mend;

%dispatch
```

Code 4. FantasyData - Dispatch according to parameters

CRUD ENDPOINT FANTASYDATA ROSTER_CREATE

The processing of the create operation is more complicated. There will be additional data corresponding to the user actions that filled the Roster ListView. The data must be inserted into the proper tables and the data must be returned to the component so it can update itself as needed.

Roster ListView configuration

```
         transport: {
           create: {
             type: "POST",
             url:
"&_URL?_program=&_PROGRAM.Data%str(&)operation=Create%str(&)for=roster",
             dataType: "json"
           },
           parameterMap: function (options, operation) {
             return { roster: kendo.stringify({names:
                $.map(options.models,function(m){return {Name:m.Name}})}) };
           },
         },
         batch: true,
```

Code 5. Roster ListView configuration

The twist with the Roster ListView is that the operation is Create and the data sent is JSON representing an array of JavaScript objects with a Name property. The data is POSTed to the endpoint and not as part of the query string. Example of posted data:

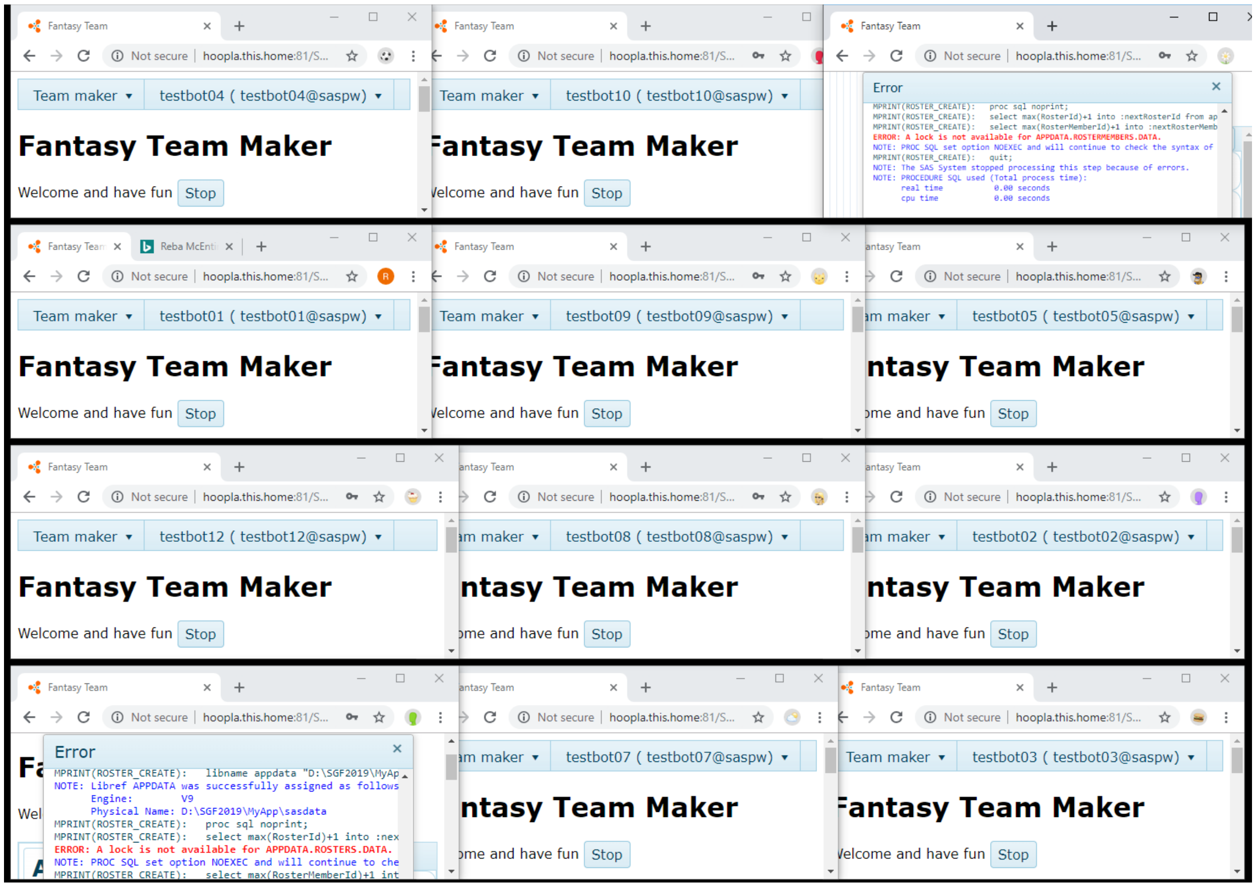
```
roster=%7B%22names%22%3A%5B%7B%22Name%22%3A%22Robidoux%2C+Billy+Jo%22%7D%2C%7
B%22Name%22%3A%22Ray%2C+Johnny%22%7D%2C%7B%22Name%22%3A%22Howell%2C+Jack%22%7
D%2C%7B%22Name%22%3A%22Quinones%2C+Rey%22%7D%2C%7B%22Name%22%3A%22Winfield%2C
+Dave%22%7D%5D%7D
```

when decoded is human readable

```
roster={"names":[{"Name":"Robidoux, Billy Jo"},{"Name":"Ray,
Johnny"},{"Name":"Howell, Jack"},{"Name":"Quinones, Rey"},{"Name":"Winfield,
Dave"}]}
```

Proc JSON only create output, a different technique is needed to parse the incoming JSON into a useable SAS data set. The SAS JSON library engine will parse a fileref containing JSON and allow an array of objects property to be accessed as if the property was a SAS data set. The Roster_Create macro in the CRUD endpoint stored process will write the POSTed data to a temporary file and read it as data using the JSON libref. Saving, or creating, a new roster requires the computation of a new identity value, and the names of the roster need to be appended to an application data set. These requirements are the reason a sharing or locking mechanism must be utilized.

What happens when there is no locking, or attempted locking techniques fail?


Fantasy Team
Not secure | hoopla.this.home:81/S...
Team maker
testbot04 (testbot04@saspw)
testbot10 (testbot10@saspw)
testbot01 (testbot01@saspw)
testbot09 (testbot09@saspw)
testbot05 (testbot05@saspw)
testbot12 (testbot12@saspw)
testbot08 (testbot08@saspw)
testbot02 (testbot02@saspw)
testbot07 (testbot07@saspw)
testbot03 (testbot03@saspw)
Fantasy Team Maker
Welcome and have fun
Stop
Error

```
MPRINT(ROSTER_CREATE):   proc sql noprint;
MPRINT(ROSTER_CREATE):   select max(RosterId)+1 into :nextRosterId from ap
MPRINT(ROSTER_CREATE):   select max(RosterMemberId)+1 into :nextRosterMemb
ERROR: A lock is not available for APPDATA.ROSTERMEMBERS.DATA.
NOTE: PROC SQL set option NOEXEC and will continue to check the syntax of
MPRINT(ROSTER_CREATE):   quit;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

```
MPRINT(ROSTER_CREATE):   libname appdata "D:\SGF2019\MyAp
NOTE: Libref APPDATA was successfully assigned as follows
      Engine:        V9
      Physical Name: D:\SGF2019\MyApp\sasdata
MPRINT(ROSTER_CREATE):   proc sql noprint;
MPRINT(ROSTER_CREATE):   select max(RosterId)+1 into :nex
ERROR: A lock is not available for APPDATA.ROSTERS.DATA.
NOTE: PROC SQL set option NOEXEC and will continue to che
MPRINT(ROSTER_CREATE):   select max(RosterMemberId)+1 int
```

A mutex will ensure atomic operations in those critical sections. First, look at the Create macro:

```
%macro waitLock; %mend;                          to be implmented later
%macro clearLock; %mend;

%macro Roster_Create();
  filename roster temp;                       copy JSON value into file
  data _null_;
    file roster; length roster $32767; roster = symget("roster");
    put roster; stop;
  run;

  libname roster json fileref=roster;      access POST data as roster.names

  %local nextRosterId nextRosterMemberId;

  %let nextRosterId = .;
  %let nextRosterMemberId = .;
```

```
   %waitLock (appdata.Rosters)                               start crticial section
   %waitLock (appdata.RosterMembers)

     proc sql noprint;                                 compute next identity values
       select max(RosterId)+1 into :nextRosterId from appdata.Rosters;
       select max(RosterMemberId)+1 into :nextRosterMemberId
       from appdata.RosterMembers;
     quit;

     data work.newRoster;                         create roster data to be appended
       if 0 then set appdata.Rosters;
       rosterId = coalesce(&nextRosterId,1);
       owner = symget("_METAUSER"); output; stop;
     run;

     data newRosterMembers;                 create rosterMember data to be appended
       if 0 then set appdata.RosterMembers;
       rosterId = coalesce(&nextRosterId,1);
       do rosterMemberId = coalesce(&nextRosterMemberId,1) by 1
       while (not lastItem);
         set roster.names(keep=name) end=lastItem; output;
       end; stop;
     run;

     proc append base=appdata.Rosters data=newRoster;
     proc append base=appdata.RosterMembers data=newRosterMembers;
     run;

   %clearLock (appdata.Rosters)                                end critical section
   %clearLock (appdata.RosterMembers)

   proc json out=_webout;                                send roster data with new ids
     write values 'roster';                              back to the component      .
       write open array;
         export newRosterMembers / nosastags;
       write close;
   run;
 %mend;
```

Code 6. Roster_create macro as CRUD service routine

The use of macro %WaitLock and %ClearLock must bound critical sections. The SAS coding within must be kept as small and fast as possible. The bounding of data set accesses within critical sections must be consistent across all the stored processes that comprise the web application.

MUTEX ADAPTER ROUTINES

SAS code can not directly call the system routines needed to manage mutexes. SAS code can use the `CALL MODULE` Routine to run methods in a dynamic link library (DLL). DLL methods written to interface with MODULE are adapters that provide new utility to the SAS session. A compiler is needed to create a DLL, and the Tiny C Compiler is up to the task and has a very small footprint. A full listing will be available by email contact.

DLL SIGNATURE

```
DLL_EXPORT void STP_WaitOnMutex
(
      const char* in_name   // data set name, will be a system mutex name
    , const char* in_timeout
    , char* update_handle   // create mutex handle
    , char* update_status1  // get last error
    , char* update_status2  // wait rc
    , char* update_status3  // wait time
    , char* update_status4  // tick when wait achieved (for elapsed comp.)
    , char* update_status5  // current thread id
) {
```

```
DLL_EXPORT BOOL STP_ReleaseMutex
(
    const char* in_hMutex
  , const char* in_timeAcquired
  , char* update_status1
  , char* update_status2
  , char* update_status3
  , char* update_status4
) {
```

All the parameters are character pointers (char*). Character values, even for numeric information, are the easiest to manage and debug within the macro environment. Special attention to this detail is important.

SASCBTBL SPECIFICATIONS FOR MODULE ROUTINE

The C signatures are communicated to MODULE through information specified in the SASCBTBL fileref. The name of the DLL is indicated in the module parameter and must be findable by the thread, either in current directory or one listed in %PATH%.

```
routine STP_WaitOnMutex
                       minarg=8 maxarg=8 stackpop=called module=stp_helper;
arg 1 char input  byaddr format=$cstr200.;* // mutex name;
arg 2 char input  byaddr format=$cstr20.; * // timeout (seconds) atoi
arg 3 char update byaddr format=$cstr20.; * // mutex handle;
arg 4 char update byaddr format=$cstr20.; * // LastError from CreateMutex;
arg 5 char update byaddr format=$cstr20.; * // RC WaitForSingleObject;
arg 6 char update byaddr format=$cstr20.; * // time waited (ms);
arg 7 char update byaddr format=$cstr20.; * // timestamp at acquired (ms);
arg 8 char update byaddr format=$cstr20.; * // current thread id;

routine STP_ReleaseMutex
                       minarg=6 maxarg=6 stackpop=called module=stp_helper;
arg 1 char input  byaddr format=$cstr20.; * // hMutex as a string;
arg 2 char input  byaddr format=$cstr20.; * // timestamp at acquire
arg 3 char update byaddr format=$cstr20.; * // LastError from ReleaseMutex;
arg 4 char update byaddr format=$cstr20.; * // LastError from CloseHandle;
arg 5 char update byaddr format=$cstr20.; * // duration in acquired (ms);
arg 6 char update byaddr format=$cstr20.; * // current thread id;

routine GetCurrentThreadId
            minarg=0 maxarg=0 stackpop=called module=kernel32 returns=long;
```

MACROS - %WAITLOCK AND %CLEARLOCK

The lock macros use global macro variables to track acquired mutex handles, names and timing values. %SYSCALL is used to invoke MODULE to run the DLL method. Values passed into the routines suffixed with a letter or string. For example, "360" is passed in as "360sec". Experience shows the "360" value is implicitly eval'd by the macro processor into a numeric value and type and used as such for the %SYSCALLed MODULE invocation. The implicit eval means the MODULE call is not honoring the DLL routine signature nor SASCBTL declaration. A "360sec" value is passed as pointable memory and the C routine uses `atoi()` to obtain the actual needed integer value. The handle and status macro variables have the same caveat and an additional requirement of needing a filler value, such as ``` `````````````` ```, assigned prior to the call. The filler ensures there is memory allocated to be written to through the char* declaration.

```
%macro waitLock(data);
  %local control routine mutex timeout symbolgen;
  %local handle status1 status2 status3 status4 status5;
…
  %global MUTEX_COUNT;
  %if %length(MUTEX_COUNT)=0 %then %let MUTEXT_COUNT=0;

  %let MUTEX_COUNT = %eval (&MUTEX_COUNT+1);
  %global MUTEX_HANDLE_&MUTEX_COUNT;
  %global MUTEX_NAME_&MUTEX_COUNT;
  %global MUTEX_TIME_&MUTEX_COUNT;

  %let control = *E;
  %let routine = STP_WaitOnMutex;
  %let mutex   = Global\MyApp.&data;
  %let timeout = 360sec;                          %*  4 minutes;
  %let timeout = 45sec;                           %* testing, quick timeout;
  %let handle  = ```````````````````;  %* fill with backtick to alloc mem;
…
  %syscall MODULE (control, routine, mutex,
             timeout, handle, status1, status2, status3, status4, status5);
…
```

The clearLock macro iterates through the global tracking macro variables to find the handle(s) corresponding to the data

```
%macro clearLock(data);
  %local control routine mutex handle symbolgen;
  %local status1 status2 status3 status4;
…
  %global MUTEX_COUNT;
  %put NOTE: MUTEX_COUNT=&MUTEX_COUNT;

  %let control = *E;
  %let routine = STP_ReleaseMutex;
  %let mutex = Global\MyApp.&data;
  %let status1 = ```````````````````; %* release mutex get last error;
…

  %do index = &MUTEX_COUNT %to 1 %by -1;
    %global MUTEX_HANDLE_&index MUTEX_NAME_&index MUTEX_TIME_&index;
    %if "&&MUTEX_NAME_&index" eq "&mutex" %then %do;

      %syscall MODULE (control, routine, MUTEX_HANDLE_&index,
                    MUTEX_TIME_&index, status1, status2, status3, status4);
…
```

Code 7. Wait and Clear Macros use %SYSCALL MODULE

DLL INNARDS

The DLL routines have two very simple tasks; acquire a mutex handle and then do something with it. The routines also perform sides task such as zeroing memory of the status arguments, gathering timing values and handle rare cases of timeout or error.

```
STP_WaitOnMutex
…
    hMutex = OpenMutex(SYNCHRONIZE, 0, in_name);             Opens a Handle
…
    DWORD rc = WaitForSingleObject (hMutex, timeout);    Wait for exclusive
…
```

The in_hMutex value passed in (below) is from information tracked in global macro variables managed by waitLock and clearLock macros.

```
STP_ReleaseMutex
…
    HANDLE hMutex = (HANDLE) atoi (in_hMutex);          Handle to exclusive
    BOOL rc1 = ReleaseMutex (hMutex);           Release it for next Waitee
…
    BOOL rc2 = CloseHandle (hMutex);                         Closes a Handle
…
```

Code 8. Open, Wait / Release, Close mutex mantra for critical sections

The success of Open and Closing a mutex handle mark the start and end of a critical section in the SAS code.

The OpenMutex system call will create a Mutex if one does not already exist. However, the system-wide mutex is only persistent for the lifetime of the thread that creates it. When process A creates a mutex M and process B acquires M before A terminates (the SAS stored process finishes) there will eventually be an error for abandonment or timeout when B releases a 'vanished' mutex.

An external process, P, not connected to a SAS thread can remove this problem. P creates all the mutexes needed for MyApp apriori and hibernates with sleep(INFINITE). The existent but unacquired mutex belong to P. The stored processes, as may be running concurrently, use the waitLock and ClearLock macros to without worry of abandonment issues.

MUTEX MASTER

A mutex master program is written to read a list of data sets that will be used in MyApp, and creates a mutex for each one. The master program is started before the MyApp pages are server, at either boot time or prior to Application Server startup.

A more dynamic mutex master would operate as a service and the DLL would make a request to the master for a mutex, of the desired name, to be created. The master process being a separate infinite thread would own but not use the mutexes. The master would also have an algorithm for acquiescing the mutexes that are not being requested.

```
Mutex Master - monolith
…
  err = fopen_s (&fp, MUTEXT_LIST_FILE, "r");       text file list of names
…
  while (++linenum) {
    if (fgets(line, sizeof(line), fp) == NULL) break;  get a name from list
    strcpy (buffer, "Global\\MyApp.");                         preface the name
    strncat_s (buffer, sizeof(buffer), line, strlen(line));
    CreateMutex (NULL, 0, buffer);
    printf ("%2d. createMutex %s\n", linenum, buffer);
  _fcloseall();
  Sleep (INFINITE);
```

Code 9. Mutex master program

Naming the mutexes with a preface MyApp. makes troubleshooting easier when examining system resource use with a tool like SysInternals Process Explorer.

CONCLUSION

SAS Stored Processes can surface a complex web application that uses the single page approach for dynamic and responsive views. HTML components can use Ajax to reach stored processes as CRUD endpoints. Coordination between application developer and SAS administrator is needed for a successful deployment that will benefit the organization. Concurrency issues require thinking outside the box, ensuring the user experience is error free and productive. The capabilities of SAS never cease to amaze me.

RECOMMENDED READING

- *PROC STREAM and SAS® Server Pages: Generating Custom HTML Reports*,
  Don Henderson
- *SAS® Server Pages <?sas and <?sas=*,
  Richard A. DeVenezia
- *SAS® 9.4 Stored Processes: Developer's Guide, Third Edition*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Richard DeVenezia
(315) 351-6774
rdevenezia@gmail.com
https://www.devenezia.com

[1] https://en.wikipedia.org/wiki/Mutual_exclusivity
[2] https://www.quora.com/What-is-the-Pauli-Exclusion-Principle
[3] https://en.wikipedia.org/wiki/Mutual_exclusion
[4] https://docs.microsoft.com/en-us/windows/desktop/Sync/mutex-objects
[5] https://stackoverflow.com/questions/52810999/programmatically-create-a-new-person-having-an-internal-login
[6] SAS® 9.4 Stored Processes: Developer's Guide, Third Edition, Using Reserved Macro Variables https://documentation.sas.com/?docsetId=stpug&docsetTarget=p184mqqbi9w6qjn1q0619x19eg02.htm&docsetVersion=9.4&locale=en