MWSUG 2016 - Paper 38

A Failure To EXIST: Why Testing for Data Set Existence with the EXIST Function Alone Is Inadequate for Serious Software Development in Asynchronous, Multiuser, and Parallel Processing Environments

Troy Martin Hughes

ABSTRACT

The Base SAS® EXIST function demonstrates the existence (or lack thereof) of a data set. Conditional logic routines commonly rely on EXIST to validate data set existence or absence before subsequent processes can be dynamically executed, circumvented, or terminated based on business logic. In synchronous software design where data sets cannot be accessed by other processes or users, EXIST is both a sufficient and reliable solution. However, because EXIST captures only a split-second snapshot of the file state, it provides no guarantee of file state persistence. Thus, in asynchronous, multiuser, and parallel processing environments, data set existence can be assessed by one process but instantaneously modified (by creating or deleting the data set) thereafter by a concurrent process, leading to a race condition that causes failure. Due to this vulnerability, most classic implementations of the EXIST function within SAS literature are insufficient for testing data set existence in these complex environments. This text demonstrates more reliable and secure methods to test SAS data set existence and perform subsequent, conditional tasks in asynchronous, multiuser, and parallel processing environments.

INTRODUCTION

To be clear, the EXIST function duly performs as advertised—its credibility and virtue remain unblemished and intact. However, the implementation of EXIST can be lacking when SAS practitioners make incorrect assumptions through faulty business logic. For example, business logic in SAS software often assumes that if a data set exists it is also available—that is, that the necessary file lock (either exclusive or shared) will be available. But an existent data set may be unavailable if other users or processes are accessing it concurrently. Thus, methods demonstrated in this text always validate both existence and availability to ensure subsequent data set operations will not fail due to either missing or locked files.

A second assumption often made in EXIST business logic is the belief that EXIST represents a static file state. This may be a valid assumption for SAS practitioners operating in single-user environments in which no other users or processes are attempting to access the data set. However, where data sets reside in shared libraries and where other users or processes have access to those libraries and their respective data sets, production code that demands robustness and reliability must realize the dynamic nature of file states and thus explicitly lock data sets after testing for existence and before attempting to access them. Without these safeguards, parallel processing that attempts concurrent access to the same data set surely will fail.

Another idiosyncrasy of the implementation of EXIST arises when a negative return value is expected—that is, some action is to be performed when a data set is determined *not* to exist. However, because a data set that does not exist cannot be locked (to guarantee availability in addition to existence), this can create a failure pattern in which two or more processes simultaneously assess that a data set does not exist and thereafter simultaneously attempt to create it, causing at least one process to fail. This text demonstrates how this obstacle is overcome by first creating a temporary data set, holding that file lock, and thereafter creating the actual data set while the file lock persists.

While a simple invocation of the EXIST function may be adequate for some uses in most single-user environments, this text dramatically expands the capability of this function, while demonstrating numerous failure patterns that might otherwise vex would-be SAS adventures into the world of parallel processing. As the advantages of concurrent and parallel processing become more widely known and demonstrated through SAS literature and as SAS practitioners shift from monolithic, serialized programming models to more efficient and modular parallel processing programming models, some techniques such as the use of the EXIST function will need to be updated to ensure they fully deliver their intended functionality.

COMMON USAGE OF EXIST FUNCTION

The EXIST function is most widely used to test the existence of a data set (and, implicitly, its respective SAS library) and to execute conditional logic thereafter. For example, if a data set exists (i.e., EXIST returns a 1), then its contents might be printed to the SAS log in the form of an exception report.

```
* positive EXIST return value;
%macro print_dsn();
%if %sysfunc(exist(lib.control)) %then %do;
    proc print data=lib.control;
    run;
    %end;
%mend;

%print_dsn;
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

Conversely, if a data set does not exist (i.e., EXIST returns a 0), the data set might be created, thus ensuring proper functioning of code that follows.

```
* negative EXIST return value;
%macro create_control();
%if %sysfunc(exist(lib.control))=0 %then %do;
    data lib.control;
       length dtg 8 name $10;
    run;
    %end;
%mend;

%macro engine();
%create_control;
* do something with lib.control;
%mend;

%engine;
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

For many purposes and especially in single-user environments, the previous business logic is sufficient and reliable. However, because EXIST can capture only the static file state of a SAS data set, that state could easily be changed by other users or processes. In multiuser environments in which additional users, processes, or SAS sessions may also be interacting with the data set being tested, the EXIST function is insufficient and will lead to heinous and sometimes baffling runtime errors, as demonstrated throughout the following sections.

In all cases, the failure occurs because EXIST demonstrates only a split-second snapshot of file status, an inherently moving target. And, even when the code that requires a data set to exist immediately follows the EXIST conditional logic, it's possible for other processes to sneak in and change that file state. To give an idea of the speed at which this occurs, the following code executes only the EXIST function inside a loop so that its average duration can be calculated.

```
%macro speedy;
%let start=%sysfunc(datetime());
%do i=1 %to 10000;
    %put &i;
    %if %sysfunc(exist(lib.control))=0 %then %do;
       %end;
    %end;
%let stop=%sysfunc(datetime());
%put TIME: %sysevalf(&stop-&start) secs;
```

```
%mend;

%speedy;
```

Executing 10,000 times in 3.5 seconds on average, each iteration completes in only 0.00035 seconds, or 0.35 milliseconds! Notwithstanding this speed, because other SAS processes are moving just as quickly, other processes can sneak in and alter or gain control of the data set being tested. To diminish this possibility, one best practice is to utilize the referenced data set immediately after its existence is tested. For example, in the PRINT_DSN macro above, the PRINT procedure immediately follows the EXIST function. And, similarly, in the CREATE_CONTROL macro, the DATA step immediately follows the EXIST function. As it turns out, however, even this best practice is insufficient to eliminate costly runtime errors in multiuser environments so a more robust approach must be implemented.

FAILURE 1: EXIST SAID MY FILE WAS THERE BUT THEN IT VANISHED!

This runtime error occurs when EXIST determines that a data exists but, by the time subsequent code requires access to the data set, the code has been deleted. For example, the following code first tests the existence of a data set and, after a positive return indicating existence, performs a subset of the data set.

```
%macro test_positive(dsn=, cnt=);
%do i=1 %to &cnt;
    %put ITERATION: &i;
    %if %sysfunc(exist(&dsn)) %then %do;
       data x;
          set &dsn;
       run;
       %end;
    %end;
%mend;

%test_positive(dsn=lib.test, cnt=1000);
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

The code is sufficient for a single-user environment because no other process should be interacting with the data set LIB.Test. However, if other users or processes are simultaneously interacting with the same data set, these interactions could change the file state, either by locking the data set or deleting it. To simulate other processes, the following code repeatedly creates and then deletes the data set LIB.Test. It should be run in a separate SAS session, after which the TEST_POSITIVE macro should be executed immediately so the two programs will run concurrently for a few seconds.

```
data lib.test;
    length char1 $10 num1 8;
run;

%macro test(lib=, dsn=, cnt=);
%do i= 1 %to &cnt;
    %put ITERATION: &i;
    data &lib..&dsn;
       length iteration dtg 8;
       format iteration 8. dtg datetime17.;
    run;
    proc datasets library=&lib nolist;
       delete &dsn;
       quit;
    run;
    %end;
%mend;
```

```
%test(lib=lib, dsn=test, cnt=1000);
```

Several different runtime errors should have been produced, and readers may need to vary the number of iterations and timing to adequately demonstrate all error types. The first error occurs when session one (executing the TEST_POSITIVE macro) attempts to access the data set LIB.Test when session two is either creating or deleting the data set, both of which require an exclusive lock on the file. The following output demonstrates this error.

```
ITERATION: 404
ERROR: A lock is not available for LIB.TEST.DATA.

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.X may be incomplete.  When this step was stopped there
were 0 observations and 0 variables.
WARNING: Data set WORK.X was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

This illustrates the critical importance of checking for availability (i.e., file lock status) as well as existence whenever a data set is required for use. Because use of the data set in the SET statement requires only a shared lock (and not an exclusive lock), data set availability can be assessed with the OPEN I/O function called from the %SYSFUNC macro function. However, two more error types still exist when the two SAS sessions are executed concurrently. The second error type occurs because in the split-second between the EXIST function and the subsequent DATA step, the first process DATASETS procedure deletes the data set, thus session one no longer can access it.

```
ITERATION: 433
ERROR: File LIB.TEST.DATA does not exist.

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.X may be incomplete.  When this step was stopped there
were 0 observations and 0 variables.
WARNING: Data set WORK.X was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The third error occurs when the EXIST function determines that the data set does exist but when the DATASETS procedure is executing in the second session when the first session subsequently attempts to utilize the data set in the SET statement.

```
ITERATION: 818
ERROR: User does not have appropriate authorization level for file LIB.TEST.DATA.

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.X may be incomplete.  When this step was stopped there
were 0 observations and 0 variables.
WARNING: Data set WORK.X was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

To remedy all three errors, the following updated macro TEST_POSITIVE should be implemented. The code first assesses existence and immediately opens a read-only data stream via the OPEN function. If the data stream fails, this indicates that the data set is in use by some other user or process. But, if the data stream is successful, the

shared lock it creates guarantees that no other user or process can modify or delete the data set while the stream remains open, thus ensuring the success of the following DATA step.

```
%macro test_positive(dsn=, cnt=);
%local dsid;
%do i=1 %to &cnt;
    %put ITERATION: &i;
    %let dsid=0;
    %if %sysfunc(exist(&dsn)) %then %let dsid=%sysfunc(open(&dsn));
    %if &dsid>0 %then %do;
        data x;
            set &dsn;
        run;
        %let close=%sysfunc(close(&dsid));
        %end;
    %end;
%mend;

%test_positive(dsn=lib.test, cnt=1000);
```

RELIABLE USE OF EXIST TO GAIN SHARED FILE ACCESS

Also note that the CLOSE function is required to close the data stream. Without this statement, because the OPEN function was invoked through the %SYSFUNC macro function, the stream will remain open indefinitely until session one is close, thus preventing all other users or processes from gaining an exclusive lock on the data set. Thus, this differs from use of OPEN within a DATA step in which the stream automatically is closed when the DATA step terminates.

Throughout this demonstration, session one has only required a shared file lock for the data set LIB.Test so that it can be used in the SET statement. However, if the process instead requires an exclusive lock (i.e., read-write access to create or modify a data set), then assessing availability must instead achieve an exclusive lock. This is demonstrated below with use of the FOPEN I/O function, which allows the data set LIB.Test to be modified in session one rather than just subset.

```
%macro test_positive(dsn=, cnt=);
%do i=1 %to &cnt;
    %put ITERATION: &i;
    %if %sysfunc(exist(&dsn)) %then %do;
        lock &dsn;
        %put SYSLCKRC: &syslckrc;
        %if &syslckrc=0 %then %do;
            data &dsn;
                set &dsn;
            run;
            lock &dsn clear;
            %end;
        %end;
    %end;
%mend;

%test_positive(dsn=lib.test, cnt=1000);
```

RELIABLE USE OF EXIST TO GAIN EXCLUSIVE FILE ACCESS

Note that because of limitations in the Base SAS language, the above code will produce runtime errors; however, these runtime errors are handled and thus ensure that the DATA step only will execute when an exclusive lock is held on LIB.Test in session one. The errors occur because Base SAS offers no way to test the presence of a file lock via the LOCK statement without actually attempting to lock the data set with the LOCK statement. However, the automatic macro variable &SYSLCKRC (system lock return code) will be 0 when a lock is successful, thus unsuccessful lock attempts can be identified through conditional logic.

The first (handled) error occurs when the LOCK statement attempts to lock the data set while the session two DATA step is executing. Note the non-zero value of &SYSLCKRC, which the exception handling uses to prevent the DATA step from executing.

```
ITERATION: 64
ERROR: A lock is not available for LIB.TEST.DATA.
SYSLCKRC: 70031
```

The second (handled) error occurs when the EXIST function assesses that the data set does exist, but when session two subsequently deletes the data set before session one can lock it. Note the non-zero value of &SYSLCKRC, which the exception handling uses to prevent the DATA step from executing.

```
ITERATION: 372
ERROR: You cannot lock LIB.TEST.DATA with a LOCK statement because LIB.TEST.DATA
does not exist.
SYSLCKRC: 630289
```

The third (handled) error occurs when the EXIST function assess that the data set does exist, but when session two subsequently runs the DATASETS procedure before session one can lock the data set. Note the non-zero value of &SYSLCKRC, which the exception handling uses to prevent the DATA step from executing.

```
ITERATION: 844
ERROR: User does not have appropriate authorization level for file LIB.TEST.DATA.
SYSLCKRC: 70030
```

FAILURE 2: EXIST SAID MY FILE WAS MISSING BUT THEN IT APPEARED

The previous examples have all performed some action because a data set did exist, based on a positive return code (i.e., 1) from the EXIST function. In other cases, some action is conditionally performed instead because a data set does not exist, such as creating the data set. The following code creates the data set LIB.Test if it does not exist and is sufficient and reliable when executed in a single-user environment:

```
%macro test_negative(dsn=, cnt=);
%do i=1 %to &cnt;
    %put ITERATION: &i;
    %if %sysfunc(exist(&dsn))=0 %then %do;
       data &dsn;
          length char1 $10;
       run;
       %end;
    %end;
%mend;

%test_negative(dsn=lib.test, cnt=2);
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

Thus, the first time the code is executed, it creates the data set LIB.Test (if it didn't already exist), while the code performs no action the second time the loop is iterated, as demonstrated in the following output.

```
%test_negative(dsn=lib.test, cnt=2);
ITERATION: 1

NOTE: Variable char1 is uninitialized.
NOTE: The data set LIB.TEST has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

```
ITERATION: 2
```

However, in a multiuser environment, it's possible that other users or processors could create the same data set

FAILURE 3: SIMULTANEOUS EXIST STATEMENTS

This failure pattern is very uncommon because it involves two sessions of SAS simultaneously testing for data set existence, receiving results that the data set does not exist, and each simultaneously attempting to create the data set thereafter. It is similar, however, to the second failure type because it involves the difficulty encountered when testing the negative return value of the EXIST statement.

In this scenario, an engine or driver program spawns multiple SAS batch jobs asynchronously—that is, the batch jobs each run concurrently rather than in series. The collective software—driver program and all batch children—communicate via a control table (LIB.Control), so the control table is first deleted in the event that a previous version already exists. The following driver program should be saved as C:\perm\driver.sas.

```
libname lib 'c:\perm';

%macro del_control(dsn= /* requires LIB.DSN format or WORK.DSN format */);
%if %sysfunc(exist(&dsn)) %then %do;
    proc datasets library=%scan(&dsn,1,.) nolist;
       delete %scan(&dsn,2,.);
       quit;
    run;
    %end;
%mend;

%macro asynch(ctrl=, cnt=);
%del_control(dsn=&ctrl);
%do i=1 %to &cnt;
    systask command """%sysget(SASROOT)\sas.exe"" -noterminal -nosplash -
nostatuswin -noicon -sysparm ""&ctrl"" -sysin ""c:\perm\asynch_test.sas"" -log
""c:\perm\asynch_test_&i..log""" status=asynch_rc_&i taskname=asynch_task_&i;
    %end;
waitfor _all_;
%put All Done!;
%mend;

%asynch(ctrl=lib.control, cnt=2);
```

The driver program iterates through the loop which spawns one or multiple sessions of SAS, each of which simultaneously executes the following SAS batch job c:\perm\asynch_test.sas. Because the driver program deleted the control table, each program subsequently attempts to build it, but only one will succeed because the others will not be able to gain an exclusive lock on the data set.

```
** THIS CODE MUST BE RUN AS A BATCH JOB **;
libname lib 'c:\perm';

%macro dosomething(dsn=);
%if %sysfunc(exist(&dsn))=0 %then %do;
    data &dsn;
       length char1 $10 num1 8;
       do i=1 to 10000;
          char1='hello';
          output;
          end;
```

```
    run;
    %end;
%mend;

%dosomething(dsn=&sysparm);
```

When the driver program is executed, it spawns the two SAS batch jobs at virtually the same time, thus each batch job simultaneously assesses the EXIST result simultaneously and each batch job assesses that the data set does not exist, thus each batch job attempts to create it, creating inherent redundancy.

CONCLUSION

The EXIST function has dutifully served the SAS community for decades and its service is duly noted. While EXIST continues to function reliably within single-user environments, its use in asynchronous, multiuser, and parallel processing environments, implementation of EXIST can lead to puzzling if not disastrous results, as competing processes are able to slip in and access, modify, or delete data sets already assessed with EXIST. This text has demonstrated methods that can overcome these vulnerabilities, enabling SAS practitioners to test for data set existence reliably and securely within more complex operational environments, and with this advancement, parallel processing can be made more reliable and robust.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com