Paper 081-2022

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 evaluates the existence (or lack thereof) of a SAS data set. Conditional logic routines commonly rely on EXIST to validate data set existence or absence before subsequent (dependent) processes can be dynamically executed, circumvented, or terminated based on business rules. In synchronous software design and 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 file state, it provides no guarantee of file state persistence. Thus, in asynchronous, multiuser, and parallel processing environments, data set existence can be evaluated by one process and be instantaneously modified thereafter by a concurrent process that creates or deletes the evaluated data set; this creates a race condition in which EXIST technically returns the correct results, but those results are invalidated before subsequent statements can execute. Because of this vulnerability, most classic implementations of the EXIST function within SAS literature are insufficient for evaluating data set existence in these more complex environments and scenarios. This text demonstrates reliable and secure methods to test SAS data set existence prior to performing 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 invalid when SAS practitioners make incorrect assumptions that are translated into faulty business logic and SAS syntax. For example, a common assumption made in SAS programming is that *if a data set exists, it is also available*—that is, the necessary file lock (either exclusive or shared) will be available. This magical thinking may be correct when you are coding singly in a stand-alone SAS instance and have no other processes accessing the data set. However, a data set may be unavailable when other users or processes are accessing it concurrently (and hold either an exclusive or shared lock on the file). Thus, methods demonstrated in this text always validate both data set existence and availability to ensure subsequent processes will not fail due to missing or locked files.

A second false assumption often made of EXIST is that it evaluates a stable file—one in which the file state (i.e., the file’s existence) is unchanging. Yes, EXIST takes a snapshot of file state, but this state can change in a picosecond, causing the results of EXIST to become outdated and effectively incorrect. Where data sets reside in shared libraries accessible by other users or processes, production code that demands robustness and reliability must account for the dynamic nature of data sets, and must ensure that file state remains frozen until subsequent (dependent) tasks have been performed. For example, if software uses the EXIST function to test data set existence, and subsequently prints a report of the data set *only* when it exists, this logic can fail in a multiuser environment. Immediately after the EXIST function but before the subsequent REPORT procedure, another user or process could have sneaked into that infinitesimal interstice and deleted the data set, causing REPORT to fail. Similarly, had the same sneaky user or process exclusively locked the data set, the REPORT procedure also would have failed—because REPORT requires a read-only lock, which cannot be obtained when another process holds an exclusive lock. Thus, without safeguards to prevent these race conditions, conditional logic that is sufficient for simpler design will fail when confronted with multiuser and parallel processing environments.

Another idiosyncrasy of EXIST implementation arises when a negative return value is expected—that is, some action is to be performed *only* when a data set is determined *not* to exist. For example, the existence of a data set might be tested, after which the data set is dynamically created if it was found to be missing. However, because a data set that does not exist cannot be locked (to guarantee availability in addition to existence), this logic can create a failure pattern in which two or more processes simultaneously evaluate 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 to overcome this obstacle, using a mutex to lock a pointer to a data set even before the data set has been created. With an assist from the LOCKSAFE macro (introduced in a prior text), the reliable use of EXIST to support conditional logic routines is demonstrated.

Although a straightforward invocation of EXIST is sufficient within single-user, serialized, and synchronous program flow, this text expands the capability of EXIST to more complex environments while demonstrating numerous failure patterns that might otherwise thwart would-be adventurers into the world of SAS parallel processing. As the performance advantages of parallel processing become more widely demonstrated throughout SAS literature and as SAS practitioners continue to shift from monolithic, serialized program flow to higher performing, modular parallel processing paradigms, the implementation of some tried-and-true techniques—such as the EXIST function—will need to be overhauled to ensure the intended functionality is delivered.

EXIST FUNCTION USAGE

The EXIST function evaluates the existence of a data set (and, implicitly, its respective SAS library) and often is used as a prerequisite to the conditional execution of subsequent code. The dynamic program flow typically performs one action if the data set exists and another if it does not, operationalized through the SAS macro language. Once data set existence has been demonstrated, subsequent processes typically require either an exclusive or shared lock on the data set. For a background in shared and exclusive locks, consult a separate text by the author. (Hughes, 2014)

For example, if the PERM.Control data set exists (i.e., EXIST returns a 1), then its contents might be subsequently printed, requiring a shared lock on the data set. However, if the data set does not exist, the subsequent PRINT procedure is skipped, which prevents a runtime error that would have resulted from a missing data set. The following code runs in a single-user environment but can fail when concurrent SAS users or sessions are interacting with the data set:

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

%print_dsn;
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

Conversely, if the PERM.Control data set does not exist (i.e., EXIST returns a 0), the data set can be created through conditional logic, thus ensuring correct functioning of any code that follows. However, data set creation always requires an exclusive lock while the data set is being created.

The following code uses EXIST as a sufficient control gate in single-user environments but can fail in multiuser environments:

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

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

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

```
%engine;
```

For many purposes and especially in single-user environments, the previous logic is sufficient and reliable. However, because EXIST captures 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 same data set being evaluated, the EXIST function (as previously implemented) is insufficient and will lead to heinous and sometimes baffling runtime errors, as demonstrated throughout the following sections.

Three environments precipitate failures of the EXIST function:

- **Multiuser** — In a multiuser environment, multiple users share a SAS instance and infrastructure, with persistent libraries and data sets accessible to multiple users across some network. Whenever multiple users have access to the same data set, the risk exists that more than one user could attempt to access or perform some action on the same data set concurrently. For this reason, merely operating in a multiuser environment adds risk not inherently associated with single-user environments, such as file access collisions and failures of EXIST.

- **Parallel Processing** — In some cases, even a single-user environment can act as (and incur risks similar to) a multiuser environment. For example, a SAS practitioner might open two or three SAS sessions on a single laptop at the same time. Software running concurrently on separate SAS sessions acts no differently than multiple users running separate SAS jobs, so again the risk exists that two or more sessions will attempt to access the same data set simultaneously, causing invalid results (or failure) when EXIST is attempted.

- **Asynchronous Processing** — Asynchronous processing represents a specific subset of parallel processing in which a SAS program executes statements that run at the same time rather than in series, as is common in synchronous SAS program flow. For example, by using the SYSTASK statement, one SAS program can spawn multiple batch jobs, each of which executes concurrently. Care must be exercised, however, to ensure that the jobs do not conflict with each other to cause corruption or failure, including unexpected results when EXIST is implemented by either the parent process or its children.

In all three scenarios, failure is caused by underlying race conditions in which two or more processes compete for access to the same data set. Race conditions can be difficult to remedy because of the speed at which they occur and, in many cases, the infrequency. Thus, EXIST might work the vast majority of the time in a multiuser environment, but when it does fail, it can have disastrous consequences that lead to the corruption or deletion of a data set! To illustrate the speed of processing, the following code calls 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(perm.control))=0 %then %do;
           %end;
    %end;
%let stop=%sysfunc(datetime());
%put TIME: %sysevalf(&stop-&start) secs;
%mend;

%speedy;
```

Executing more than 10,000 times in just 3.5 seconds, 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 with EXIST.

To diminish this possibility of corruption, one best practice is to utilize the referenced data set immediately after its existence is tested. For example, in the previous PRINT_DSN macro, the PRINT procedure immediately follows the EXIST function. Similarly, in the CREATE_CONTROL macro, the DATA step immediately follows the EXIST function. However, even this best practice is insufficient to eliminate runtime errors in multiuser environments because the interstice can still be invaded by external users or processes. Thus, a more robust and reliable solution is required.

STRESS TESTING EXIST DEMONSTRATES EXIST VULNERABILITY

In some cases, the failure of EXIST in multiuser environments does not produce runtime errors but rather invalid results. For example, when a data set is created by setting itself (with the SET statement, presumably to modify the data set while maintaining the original data set name), for a brief shudder, EXIST falsely assesses that the data set does not exist. To demonstrate this anomaly, the following DATA step and CREATE macro should be executed:

```
* saved as c:\sas\perm\program1.sas;
libname perm 'c:\sas\perm';

data perm.create;
   length char $10;
   char='sas';
run;

%macro create(dsn=, cnt=);
%do i=1 %to &cnt;
    data &dsn;
          set &dsn;
    run;
    %end;
%put SYSCC: &syscc;
%mend;

%create(dsn=perm.create, cnt=1000);
```

In a second SAS session, the following TEST_EXIST macro should be executed, which repeatedly tests the existence of the PERM.Create data set:

```
* saved as c:\sas\perm\program2.sas;
libname perm 'c:\sas\perm';

%macro test_exist(dsn=, cnt=);
%do i=1 %to &cnt;
   %if %sysfunc(exist(&dsn))
         %then %put YES;
   %else %put NO;
   %end;
%mend;

%test_exist(dsn=PERM.create, cnt=1000);
```

When a sample log from the second session is examined, it demonstrates a pattern of predominantly YESes, interspersed with some NOs:

```
YES
YES
YES
NO
YES
YES
```

```
YES
```

As it turns out, when SAS builds a data set, it first creates a temporary data set with an LCK extension. For example, when PERM.Create is being created, the new data set is temporarily named C:\sas\perm\create.sas7bdat.lck, after which this file is renamed C:\sas\perm\create.sas7bdat. Thus, if the EXIST function evaluates a data set during this rename process, it will generate inaccurate results that depict that an existent file is missing. This vulnerability exists in single-user and serialized processing scenarios, but because the threat of separate sessions utilizing EXIST is absent in these environments, the vulnerability cannot be exploited so the risk is never realized.

To demonstrate the frequency of this risk, the TEST_EXIST macro should be modified to count YESes and NOs (i.e., positive and negative EXIST return codes, respectively) rather than printing them to the log:

```
%macro test_exist(dsn=, cnt=);
%let yes=0;
%let no=0;
%do i=1 %to &cnt;
   %if %sysfunc(exist(&dsn))
         %then %let yes=%eval(&yes+1);
   %else %let no=%eval(&no+1);
   %end;
%put YES: &yes;
%put NO: &no;
%mend;

%test_exist(dsn=PERM.create, cnt=1000);
```

By first executing the CREATE macro in the first session, and by subsequently executing the TEST_EXIST macro in the second session, the results indicate that EXIST failed to detect the PERM.Create data set in 10 out of 1,000 iterations, or 1 percent of the time:

```
%test_exist(dsn=perm.create, cnt=1000);
YES: 990
NO: 10
```

The likelihood of this risk will vary based on processor speed as well as other factors. For example, as a data set increases in size, the ratio of the time spent building the data set (during which the original file name is detectable by EXIST) to the time spent renaming the data set will increase. Thus, the risk of occurrence will be greatest for empty data sets (such as PERM.Create), whereas very large data sets will rarely encounter this race condition. Notwithstanding the frequency of occurrence, the mere fact that EXIST can return invalid results can lead to much more pernicious outcomes, as demonstrated in the next two sections. For example, if rather than merely printing YES or NO or incrementing a counter, other input/output (I/O) functions (such as reading from or writing to a data set) are executed based on the results of erroneous EXIST return codes, several failure patterns emerge, both for positive and negative EXIST return codes.

STRESS TESTING POSITIVE EXIST RETURN CODES

This runtime error occurs when EXIST determines that a data exists but, before the next line of code can access the data set, the data set is subsequently deleted or locked by a separate user or process. For example, the following code first tests the existence of a data set and, after validating existence, reads the data set in a DATA step:

```
* saved as c:\sas\perm\program3.sas;
libname perm 'c:\sas\perm';

%macro test_positive(dsn=, cnt=);
%do i=1 %to &cnt;
    %put ITERATION: &i;
    %if %sysfunc(exist(&dsn)) %then %do;
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

```
          data x;
                set &dsn;
          run;
          %end;
    %end;
%mend;

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

The previous code is sufficient in a single-user environment because no other process should be interacting with the PERM.Test data set. However, if other users or processes are simultaneously interacting with the data set, these interactions could change the file state, either by creating the data set or deleting it. To simulate other processes, the following code repeatedly creates and then deletes the PERM.Test data set. To stress test the EXIST function, the TEST_POSITIVE macro should be run in one SAS session while the following code (including the DATA step and TEST macro) is run in a second SAS session:

```
* saved as c:\sas\perm\program4.sas;
libname perm 'c:\sas\perm';

data perm.test;
    length char1 $10 num1 8;
run;

%macro test(dsn=, cnt=);
%do i= 1 %to &cnt;
    %put ITERATION: &i;
    data &dsn;
          length iteration dtg 8;
          format iteration 8. dtg datetime17.;
          iteration=&i;
          dtg=%sysfunc(datetime());
    run;
    proc delete data=&dsn;
    run;
    %end;
%mend;

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

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

```
ITERATION: 404
ERROR: A lock is not available for PERM.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 evaluating data set availability (i.e., file lock status) as well as existence whenever a data set must be accessed.

The second error type occurs because in the nanosecond between the EXIST function and the subsequent DATA step, the DELETE procedure in the second session deletes the data set; thus, session one can no longer access the data set:

```
ITERATION: 433
ERROR: File PERM.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 type occurs when the EXIST function determines that the data set does exist but does so during the DELETE procedure but before the data set has been deleted:

```
ITERATION: 818
ERROR:  User  does  not  have  appropriate  authorization  level  for  file
PERM.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
```

Runtime errors also occur in session two (running the TEST macro). For example, the DELETE procedure will fail if it cannot gain an exclusive lock on the data set:

```
ERROR: A lock is not available for PERM.TEST.DATA.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE DELETE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

ITERATION: 89
```

A separate failure of DELETE can also be observed in session two:

```
ERROR: File deletion failed for PERM.TEST.DATA.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE DELETE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

In other cases, the DATA step in session two will fail before an observation can even be created, because an exclusive lock is not available:

```
ERROR: A lock is not available for PERM.TEST.DATA.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set PERM.TEST was only partially opened and will not be
saved.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Finally, the DATA step in session two can also fail when an observation is created, but then cannot be saved—again because an exclusive lock cannot be obtained:

```
NOTE: The data set PERM.TEST has 1 observations and 2 variables.
ERROR: A lock is not available for PERM.TEST.DATA.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

These numerous failure patterns across both SAS sessions demonstrate the unreliability of EXIST in multiuser and concurrent processing environments.

STRESS TESTING NEGATIVE EXIST RETURN CODES

The previous examples performed some action because a data set existed, as indicated through the EXIST return code of 1. In other cases, an action (e.g., creating a data set) is conditionally performed because a data set does *not* exist. The following code creates the PERM.Test data set if it does not exist, and is sufficient and reliable when executed in a single-user, synchronous environment:

```
* saved as c:\sas\perm\program5.sas;
%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=perm.test, cnt=50000);
```

UNRELIABLE USE OF EXIST IN MULTIUSER ENVIRONMENT

Thus, the first time the code is executed, it creates the PERM.Test data set (if it didn't already exist); the loop performs no action the second and subsequent times it is iterated. When TEST_NEGATIVE is executed while the previous TEST macro (i.e., Program4.sas) concurrently executes, several failure patterns again emerge. Note that TEST should be started before TEST_NEGATIVE because TEST_NEGATIVE will otherwise iterate its full 10,000 times before TEST is manually started.

In this sample run, the first observed runtime error is also the most pernicious. Whenever "ERROR: Rename of temporary member" appears in a SAS log, this is an indication that the data set was corrupted during creation and was likely deleted. The log from the SAS session running TEST_NEGATIVE demonstrates this failure pattern:

```
ITERATION: 329

NOTE: Variable char1 is uninitialized.
NOTE: The data set PERM.TEST has 1 observations and 1 variables.
ERROR: Rename of temporary member for PERM.TEST.DATA failed.
 File can be found in c:\perm.
NOTE: DATA statement used (Total process time):
      real time           1.03 seconds
      cpu time            0.12 seconds

ITERATION: 330
```

A second runtime error occurs when EXIST evaluates that the data set does not exist, but the second SAS session running TEST begins to delete the data set:

```
ITERATION: 420
```

```
ERROR:  User  does  not  have  appropriate  authorization  level  for  file
PERM.TEST.DATA.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

ITERATION: 421
```

A third runtime error occurs when TEST_NEGATIVE evaluates that the data set does not exist, but either the DATA step or DELETE procedure in the TEST macro executes before the DATA step in TEST_NEGATIVE can lock the data set:

```
ITERATION: 421

ERROR: A lock is not available for PERM.TEST.DATA.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set PERM.TEST was only partially opened and will not be
saved.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

ITERATION: 422
```

Thus, as with the previous section, evaluating file existence with EXIST is insufficient when the possibility exists that other users or processes could be attempting to access or alter the same data set.

THE EXISTENCE-AVAILABILITY MATRIX

The existence-availability matrix describes the intersection of the EXIST function return code (either positive or negative) and file lock status (either exclusive or shared). Exclusive (i.e., read-write) locks are required whenever a data set is created, modified, or deleted. As the name suggests, only one user or process can gain an exclusive lock on a specific data set at one time. Shared locks, conversely, are read-only and enable multiple users or processes to gain concurrent access to a data set. For example, PRINT, FREQ, REPORT, and MEANS procedures can each be run on the same data set at the same time from separate SAS sessions because none of these procedures needs to modify the data. In fact, a SAS session printing a data set will have no awareness that a separate session is simultaneously running the MEANS procedure on the same data set.

Given that EXIST is primarily used to drive conditional logic in which the existence of some data set is evaluated before that data set is later used, the relationship between EXIST return codes and the required file lock status can be summarized in three distinct outcomes, represented in Figure 1.

EXIST Return Code	Exclusive Lock	Shared Lock
Positive (EXIST=1)	• modify an existing data set • delete a data set • sort a data set (to same file)	• PROC PRINT or REPORT • PROC MEANS or FREQ • sort a data set (to new file)
Negative (EXIST=0)	• create a new data set	N/A

Figure 1. Existence-Availability Matrix with Example Conditional Logic Objectives

For example, if a data set needs to be modified (by setting itself with the SET statement) or deleted, the data set must both exist and have an exclusive lock available—this lock demonstrates that no other user or process is accessing the data set concurrently. This use case is captured in the Positive-Exclusive quadrant of the matrix.

When creating a new data set from scratch, conversely, a prerequisite might be that the data set *not* already exist, which is captured in the Negative-Exclusive quadrant. Finally, the Positive-Shared quadrant captures use cases in which the data set must exist but for which exclusive access is not required. Figure 2 demonstrates examples of EXIST functionality in each of these quadrants, with all code demonstrating valid uses of EXIST in single-user environments.

EXIST Return Code	Exclusive Lock	Shared Lock

Positive (EXIST=1)	`%if %sysfunc(exist(&dsn))`
	`   %then %do;`
	`      data &dsn;`
	`         set &dsn;`
	`      run;`
	`      %end;`
		`%if %sysfunc(exist(&dsn))`
		`   %then %do;`
		`      proc print`
		`data=&dsn;`
		`         var char1;`
		`      run;`
		`      %end;`

Negative (EXIST=0)	`%if %sysfunc(exist(&dsn))=0`
	`   %then %do;`
	`      data &dsn;`
	`         length char $10;`
	`         char='new data';`
	`      run;`
	`      %end;`
		N/A

Figure 2. Existence-Availability Matrix with Example EXIST Implementations

Whereas Figures 1 and 2 require the implied exclusive or shared locks, respectively, no mechanism within the Figure 2 examples validates or enforces data set availability. Thus, although testing for data set existence alone is sufficient in single-user, synchronous, and serialized design, data set availability must additionally be validated within multiuser, asynchronous, and parallel processing design. In the following section, the LOCKSAFE macro is implemented to ensure both data set existence and availability—sufficiently to provide reliable data set locking, effectively freezing the file state while other actions can be performed on the data set.

LOCKSAFE SOLUTION

In a multiuser or parallel processing environment, the only way to guarantee that a data set exists (or does not exist) is to lock down its file state, thus preventing external data set access for some duration following the EXIST function. This can be accomplished with the use of *mutex semaphores*—including the business logic and concurrency controls that ensure that no other process accesses the data set exclusively at the same time. In a separate text, the author demonstrates reliable file locking with the use of mutex semaphores—techniques that facilitate not only reliable file locking but also reliable EXIST functionality in multiuser environments. (Hughes, 2017)

To implement LOCKSAFE to support data set existence checking, download the referenced paper, and save the LOCKSAFE macro to the following (or similar) location:

```
C:\perm\locksafe.sas
```

The LOCKSAFE macro relies on semaphore and mutex text files that are created, and which indicate whether a data set is locked. The mutexes and semaphores are empty text files that are stored in the MUTEX library, which has a default location of C:\perm\mutex. To change this location, modify the MUTEX library definition within the Locksafe program:

```
libname &mutlib 'c:\perm\mutex'; * CHANGE TO ACTUAL LOCATION
```

The LOCKSAFE invocation should precede the test for existence and, because LOCKSAFE produces a return code LOCKSAFERC, this return code must additionally be tested before use of the EXIST function. For example, the following code demonstrates a modification of the Positive-Exclusive quadrant code from Figure 2, now rendered safe and reliable in multiuser environments:

```
%macro reliable_exist(dsn=);
%locksafe(dsn=&dsn, sec=5, max=600, type=W);
    %if %length(&locksafeRC)=0 %then %do;
          %if %sysfunc(exist(&dsn)) %then %do;
                data &dsn;
                      set &dsn;
                run;
                %end;
          %else %put Data Set Does Not Exist!;
          %locksafe(dsn=&dsn, terminate=YES);
          %end;
    %else %put Data Set Locked!;
%mend;

%reliable_exist(dsn=perm.control);
```

The EXIST function operates reliably between the two LOCKSAFE invocations because LOCKSAFE effectively freezes file status by preventing other sessions (that are also using LOCKSAFE) from accessing the data set. In other words, between the two LOCKSAFE invocations, EXIST operates as it would in a single-user environment. The second LOCKSAFE invocation is required to terminate the file lock, and releases the lock on the PERM.Control data set so it can again be accessed by external SAS sessions (if necessary). Without the TERMINATE=YES invocation, the current SAS session would unnecessarily maintain the exclusive lock on the data set, and other SAS sessions (that were also using LOCKSAFE) would be indefinitely unable to access the data set.

A benefit of LOCKSAFE is that it can be used to lock data sets that don't yet exist—functionality not available in the now-deprecated, embarrassingly unfulfilling, grossly underwhelming SAS LOCK statement, which would return a runtime error when an attempt to lock a nonexistent data set was made. For example, if a process needed to create the PERM.Control data set but the data set did not exist, there exists the possibility that two processes running in separate SAS sessions could each simultaneously evaluate that the data set did not exist, and each simultaneously attempt to create the data set, leading to failure of one or both processes. Because LOCKSAFE creates a unique pointer to each data set, that pointer can be created before the actual data set exists, which enables the lock to be achieved, and the data set to be created with confidence that no other process is trying to create, modify, or access the same data set.

Implementation and caveats of LOCKSAFE are described in detail in the referenced text. For example, LOCKSAFE must be implemented on all processes that reference a data set or else rogue processes will be able to sneak in and bypass its semaphores and mutexes. Another hiccup can occur when a process locks a data set with the first LOCKSAFE invocation but terminates abruptly before the subsequent TERMINATE=YES invocation. When this occurs, the semaphores and mutexes may need to be manually deleted from the MUTEX library, as described in detail in the separate LOCKSAFE text.

One final caveat of the LOCKSAFE solution supporting EXIST functionality is the lack of shared locking in LOCKSAFE. In its beta release, LOCKSAFE generates only exclusive—not shared—file locks, so the Positive-Shared quadrant of the existence-availability matrix is missing. Although an exclusive lock can be generated in lieu of a shared lock, this unnecessarily prevents other users or processes from simultaneously accessing the data set. For example, the following code freezes a file state so that PERM.Control can be printed reliably, even in a multiuser environment:

```
%macro reliable_exist(dsn=);
%locksafe(dsn=&dsn, sec=5, max=600, type=W);
    %if %length(&locksafeRC)=0 %then %do;
```

```
            %if %sysfunc(exist(&dsn)) %then %do;
                  proc print data=&dsn;
                  run;
                  %end;
            %else %put Data Set Does Not Exist!;
            %locksafe(dsn=&dsn, terminate=YES);
            %end;
      %else %put Data Set Locked!;
   %mend;

   %reliable_exist(dsn=perm.control);
```

However, because the PRINT procedure requires only a shared lock (but LOCKSAFE generates an exclusive lock), this implementation unnecessarily prevents separate SAS sessions from concurrently running MEANS, FREQ, or other read-only procedures or processes on PERM.Control that also require only a shared file lock. Future versions of the LOCKSAFE macro could benefit EXIST functionality in the Positive-Shared quadrant by allowing users to specify whether an exclusive or shared lock were required, thus expanding the versatility of LOCKSAFE.

CONCLUSION

The EXIST function has dutifully served the SAS community, and although EXIST functions reliably within single-user environments and synchronous process flows, its implementation within asynchronous, multiuser, and parallel processing environments can lead to puzzling—if not disastrous—results. Race conditions that can occur between competing processes in separate SAS sessions can cause EXIST to falsely depict that a data set does not exist when it in fact does. Other race conditions can occur when EXIST accurately depicts that a data set does or does not exist but when that data set is immediately created or deleted by a subsequent process is a separate SAS session. This text demonstrates use of the LOCKSAFE macro to overcome these vulnerabilities and risk, and to enable EXIST to perform reliably and securely even in more complex, concurrent environments.

REFERENCES

Hughes, T. M. (2014). From a One-Horse to a One-Stoplight Town: A Base SAS Solution to Preventing Data Access Collisions through the Detection and Deployment of Shared and Exclusive File Locks. *Western Users of SAS Software (WUSS).* San Jose, California.

Hughes, T. M. (2017). Stress Testing and Supplanting the SAS® LOCK Statement: Implementing Mutex Semaphores To Provide Reliable File Locking in Multiuser Environments To Enable and Synchronize Parallel Processing. *SAS Global Forum (SGF).* Orlando, Florida.

CONTACT INFORMATION

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

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