Paper BB-18-2014

Will You Smell Smoke When Your Data Are on Fire? The SAS Smoke Detector: Installing a Scalable Quality Control Dashboard for Transactional and Persistent Data

Troy Martin Hughes

ABSTRACT

Smoke detectors operate by comparing actual air quality to expected air quality standards and immediately alerting occupants when smoke or particle levels exceed established thresholds. Just as rapid identification of smoke (i.e, poor air quality) can detect harmful fire and facilitate its early extinguishment, rapid detection of poor quality data can highlight data entry or ingestion errors, faulty logic, insufficient or inaccurate business rules, or process failure. Aspects of data quality—such as availability, completeness, correctness, and timeliness—together should be assessed against stated requirements that account for the scope, objective, and intended use of data products. A single outlier, an accidentally locked data set, or even subtle modifications to a data structure can cause a robust extract transform load (ETL) infrastructure to grind to a halt or produce invalid results. Thus, a mature data infrastructure should incorporate quality assurance methods that facilitate robust processing and quality data products, as well as quality control methods that monitor and validate data products against their stated requirements. The SAS Smoke Detector represents a scalable, generalizable solution that assesses the availability, completeness, and structure of persistent SAS data sets, ideal for finished data products or transactional data sets received with standardized frequency and format. Like a smoke detector, the quality control dashboard is not intended to discover the source of the blaze, but rather to sound an alarm to stakeholders that data have been modified, locked, deleted, or otherwise corrupted. Through rapid detection and response, both the fidelity of data is increased as well as the responsiveness of developers to threats to data quality and validity.

INTRODUCTION

Smoke detectors identify potentially hazardous situations and signal persons to get out and get help. The ear-piercing alarms of simple battery-powered units are intended to wake and warn occupants and provide life-saving seconds, while more complex, hard-wired units common in commercial spaces often automatically activate fire sprinklers and notify the fire department, police, or security company. Regardless of the type of device, the intent of a smoke detector is never to identify the cause of the blaze, but rather immediately to notify both would-be victims and first responders of imminent danger.

Data disasters can be no less perilous than structure fires. Poor quality and erroneous data can thwart deadlines, increase project costs, cause process failure, and lead to flawed analysis. Like smoke, tell-tale signs of trouble immediately should be recognized, triaged, and resolved to prevent further destruction. Within data analytic development, data products and solutions—including data sets, analyses, reports, and data-driven decisions—provide business value to decision makers and keep analysts and developers gainfully employed. Thus, the ability to establish, to maintain, and to demonstrate data quality is a core focus and responsibility of data stewards. Some common attributes of data quality include:

- Data availability – the degree to which data products are where they should be, accessible to stakeholders, and in the prescribed format.
- Data completeness – the degree to which data sets have the required number and type of observations.
- Data correctness – the degree to which data accurately reflect real world constructs (i.e., their actual values.)
- Data timeliness: the degree to which data are current and temporally relevant.

Quality assurance methods can facilitate high quality data through the development of flexible, fault-tolerant code that adapts to environmental, user, and data anomalies or idiosyncrasies. For example, data availability is increased when

code intelligently and automatically adapts to wait for locked data sets, rather than crashing immediately and producing copious downstream errors. Data correctness is increased when exception handling routines model and standardize categorical data to produce fewer missing or erroneous values[i]. And, data timeliness can be facilitated by the implementation of technical best practices that improve the efficiency of data flows which create data products. Inasmuch as these and other technically focused quality assurance methods can improve data quality, non-coding quality assurance activities—such as Agile development best practices[ii]—also can enhance the development environment, thereby ultimately improving the quality of derivative data products. While the Smoke Detector demonstrates only validation of data availability, date completeness, and data structure, the extensible code facilitates future expansion to validate data accuracy and other data quality attributes.

While quality assurance methods can facilitate and instill data quality, quality control methods conversely test and monitor data products to ensure they meet established quality standards. For example, if an organization establishes a threshold that 99 percent of zip codes within a data set must be valid, quality assurance processes might be emplaced to facilitate reaching this goal, perhaps by validating zip codes against a city-state lookup table. However, only through inspection of the final data set can developers and other stakeholders demonstrate whether the 99 percent performance requirement was achieved. And, when quality control tests determine that performance requirements and quality standards are not being met, additional quality assurance methods often can be implemented to improve data quality. Thus, while comprehensive quality assurance methods serve data flow and data product development, quality control measures must be relied upon to monitor and validate that quality.

This text describes the implementation of a scalable, generalizable quality control dashboard that facilitates near-real-time validation of transactional and other persistent SAS data sets. Data sets are pinged at regular intervals (e.g., every five minutes) to assess whether they are available (i.e., both existent and unlocked) and to collect structural metadata for comparison against baseline metrics. Thus, if a persistent data set becomes locked for an unreasonable amount of time, drops a character field, loses half its observations, or is deleted, these exceptions will be detected. Exceptions are stored in library- and member-level control tables that can be accessed by the SAS/BI interface or dynamic SAS Output Delivery System (ODS) reporting. The Smoke Detector thus supplants the ad hoc "putting out fires" approach to data quality control by instilling automated, reliable processes that vigilantly and continuously sniff the air for any signs of trouble.

QUALITY CONTROL

Quality control first necessitates quality requirements (or standards) against which products are measured. Thus, in order to assess the quality of a data product, quality requirements already must exist that describe criteria against which the data set will be judged. For example, when an analyst conducts a manual quality review or quality control check of a data set downloaded from a third-party source, he uses heuristics to judge what he sees (e.g., lots of missing data) against what he expects (e.g., complete data) to determine that the data set lacks quality. Automated quality control methods perform similar assessments, albeit trading heuristics, intuition, and analytic experience for business rules that prescribe acceptable and unacceptable data quality standards. Although the logic espoused by the Smoke Detector is simple and straightforward, complex conditional processing and fuzzy logic can be implemented within quality control coding to identify statistical outliers, multivariate anomalies, and other eccentric or erroneous data that the human eye likely never would catch.

While quality control typically is thought of as occurring at the end of a production line (or data flow, in the data analytics domain), quality control methods can be applied throughout process flows and the data life cycle. For example, quality control methods that validate data sets are useful when applied to third-party data immediately after extraction. Identification of erroneous or incomplete data at this early stage can prevent them from being ingested unnecessarily into a permanent data store, as well as avoid the cost and time associated with having to purge faulty data when they eventually are discovered to be invalid. Transactional databases especially lend themselves to automated quality control measures because they typically are received with standardized frequency and format. Quality control checkpoints also often are staged throughout a data flow, at which points data content and quality are validated and blessed to advance to the next module or process. Thus, by implementing quality control checks within a quality assurance framework, not only can data products be confidently assessed to be legitimate, but also the occasional illegitimate data products can be flagged and resolved immediately.

When quality control checks are applied to finished data products, they can serve as a final validation stamp before demonstration, distribution, or use. Each of the many facets of data quality can be assessed, so long as business rules that dictate quality requirements can be articulated and translated into code logic. When quality control methods are used to validate data values (i.e., to assess data correctness), business rules may become absurdly complex in order to model the range of acceptable and unacceptable data while taking into account mitigating or compounding circumstances. The validation of other aspects of data quality—such as availability or completeness—typically is less complex and can be generalized across projects because data structure consistency can be assessed by comparison of a data product to either its baseline or last observed metrics. The Smoke Detector, thus, maximizes generalizability because it assesses only data availability and completeness and omits inferences to data correctness.

SMOKE DETECTOR QUALITY CONTROL OUPUT

The scope, type, and degree of quality control validation are limited only by the creativity of developers and their ability to translate business rules into measurable attributes. In the Smoke Detector example, for simplicity and proof of concept, only the SAS dictionary table (dictionary.tables) is used to perform quality control validation. The following fields/attributes are collected at baseline and monitored thereafter:

- crdate – The file create date is utilized to identify SAS data sets that have been modified, thus cuing the Smoke Detector that other data set structure attributes also may have changed.

- nobs – In this example, if the number of observations in a data set decreases, a warning alert is generated. This is an ideal business rule for persistent data sets that never should lose observations. Transactional data sets, however, may have varying numbers of observations, so this rule will yield false positive warnings where observation levels fluctuate. *A more dynamic solution would allow this business rule to be specified uniquely for each data set, rather than singly for the macro, and include statistical and other controls to better define expected and unexpected variations.*

- filesize – Like the number of observations, file size is assumed here to remain either constant or increase, thus a warning is generated when file size decreases. *A more dynamic solution would allow this business rule to be specified for each data set, rather than singly for the macro.*

- compress – The compression status is unlikely to change but, when it does, a warning is generated.

- num_character – This field records the number of numeric fields in each data set and, if this changes, an error is generated. Because only the dictionary.tables data set is interrogated, no additional field-specific metadata is available, such as field names, lengths, or formats. Thus, if one character field vanishes and is replaced by a different character field, the change would not be detected by the Smoke Detector. *To add field-specific granularity, the dictionary.columns data set as well would need to be incorporated and monitored; this is feasible but beyond the scope of this text.*

- num_numeric – Similar to above, an error is generated if the number of numeric fields is modified for a data set.

These additional notes, warnings, and errors are generated by the Smoke Detector and speak to data availability:

- table modified – A note is generated each time the create date of a data set changes, regardless of whether the data or structure are modified.

- new table – A warning is generated when a new data set appears.

- table has zero observations – A warning is generated for all empty data sets.

- deleted table – An error is generated when a data set disappears. *Note that the disappearance will have been confirmed both through the SAS dictionary tables as well as through the operating system via the FILENAME statement and PIPE option.*

While the above quality control attributes are assessed in the outer loop of the Smoke Detector macro, the following lock status attributes are assessed at a higher frequency within the inner loop. Lock notes are generated for each data set that has either a shared or exclusive lock, while lock warnings and errors are generated for those data sets whose lock durations exceed the respective warning and error thresholds, as established in the Smoke Detector macro invocation.

SMOKE DETECTOR INITIALIZATION

One SAS library must be created:

- smoky

In this example (a UNIX environment), the Smoky library is initialized in the following statement. The %LOCKITDOWN macro also is utilized to guarantee that file collisions do not occur when attempting to access persistent data sets in the Smoky library. The %LOCKITDOWN macro is not described in this text, but is detailed in a separate text by the author.[iii]

```
libname smoky '/folders/myfolders/smoky/';
%include '/folders/myfolders/lockitdown.sas';
```

Once the Smoky library is created, all other customization occurs through parameterized macro values. During baseline invocation, the following data sets are automatically initialized in the Smoky library:

- smoke.smoke_detector – This is the primary control table in which each observation represents a unique SAS data set that is being monitored. The control table persists between instances of Smoke Detector execution but regenerates if deleted accidentally.
- smoky.smoke_detector_long – This is the longitudinal data set that demonstrates changes that have occurred in data set structure over time. It persists between instances of Smoke Detector execution but regenerates if deleted accidentally.
- smoky.smoke_detector_lock – This is the longitudinal data set that demonstrates changes in lock status that have occurred over time. It persists between instances of Smoke Detector execution but regenerates if deleted accidentally.
- smoky.smoke_detector_lib_rpt – This data set is used to create the library-level reporting and is created during each inner loop iteration.
- smoky.smoke_detector_historical – This data set is used to create the member-level reporting and is created during each inner loop iteration.

SMOKE DETECTOR MACRO INVOCATION

The Smoke Detector acts as an independent agent that collects metadata metrics from persistent SAS data sets that reside in a parameterized list of libraries. General metrics—including data set existence, availability, structure, and completeness—are collected at macro initialization (i.e., baseline) as well as at regular intervals, typically every 20 to 30 minutes. Lock status metrics—including the existence and duration of shared and exclusive file locks—also are collected at baseline as well as at more frequent intervals, such as every five to ten minutes. In this example and text, the Smoke Detector invocation is depicted to run for a limited duration, specified by the &MINTOT macro parameter. Under operational conditions, the macro should be scheduled to run automatically to monitor the health of persistent data sets.

```
%macro smoky (mintot = /* number of minutes code executes before terminating */,
    libloopmax = /* frequency (secs) of structure (dictionary) queries */,
    lockloopmax = /* frequency (secs) of locking queries */,
    lockwarthresh = /* seconds after which lock warning alert will occur */,
```

```
lockerrthresh = /* seconds after which lock error alert will occur */,
hrsretain = /* hours after which historic records are deleted */,
liblist = /* space-delimited list of libraries to be included */,
tabexclude = /* space-delimited list of data sets to exclude */,
outputloc = /* directory where output reports are created */);
```

Examples of these parameters include:

- &MINTOT– For example, a value of 1200 would cause the code to execute 20 hours throughout the workday, after which the code could be scheduled to start again the next morning.
- &LIBLOOPMAX – A value of 1800 would cause this loop to iterate approximately every 30 minutes, at which time data set availability and structure metrics would be assessed.
- &LOCKLOOPMAX – A value of 300 would cause this loop to iterate approximately every 5 minutes, at which time lock status would be tested for each data set. Lock status is tested more frequently because even a data set that is locked for 10 minutes may signal that a process has stalled or failed.
- &LOCKWARTHRESH – A value of 600 would cause a warning alert to register once a data set had been locked for ten minutes or more. In reality, if this value were being utilized in conjunction with the above parameter values, this would indicate that when this warning was first identified, the lock could have been in effect for from between 10 and 15 minutes. *In the current version of Smoke Detector, both lock warning and error thresholds are static for all data sets. In a more optimized, dynamic environment, these lock thresholds could be distinct to each data set, possibly by calculating the data set size and number of observations to assess expected lock durations under normal operations.*
- &LOCKERRTHRESH – A value of 1200 would cause an error alert to register once a data set had been locked for twenty minutes.
- &HRSRETAIN – A value of 48 indicates that longitudinal reporting will demonstrate warnings and errors—by hour—for the most recent 48 hours. Typical values can range between 24 and 72 (one to three days), but scrolling through a 72-column report is no joy.
- &LIBLIST – This is the space-delimited list of SAS libraries that should be interrogated. Libraries can be empty, but at least one library is required to have at least one data set to be monitored.
- &TABEXCLUDE – This is the list of space-delimited SAS data sets (LIBRARY.DATASET format required) that should not be monitored, typically because they are not critical or because they fluctuate too dramatically to monitor through business rules. Only data sets found within libraries specified in the &LIBLIST parameter need to be included here.
- &OUTPUTLOC – This is the location where HTML reporting will be saved, for example: /folders/myfolders/smoky/.

These additional persistent macro variables are generated throughout operation:

- &AUTOLIBLIST – A space-delimited list of all SAS libraries that exist (i.e., in dictionary.tables), excluding those expunged by the &WHERE parameter value.
- &WHERE – This represents a compilation of the &LIBLIST and &TABEXCLUDE parameter values, and is used throughout the code to subset the data
- &STARTTIME – This represents the date and time of the Smoke Detector execution.
- &LOCKCLR – This macro variable is generated by the %LOCKITDOWN macro and used throughout the code to clear the exclusive (i.e., read-write) locks created to access data sets in the Smoky library.

BEHIND THE SMOKE AND MIRRORS – OUTER LOOP

In Washington, D.C., the “outer loop” represents the counterclockwise motion—or, lack thereof—of traffic through Maryland and Virginia around the Interstate 495 capital beltway. Within the Smoke Detector, however, the “outer loop” references the continual loop that gathers data availability, completeness, and structure metrics. The outer loop recycles after a parameterized number of seconds, declared in the &LIBLOOPMAX macro parameter, and concludes when the macro parameter &MINTOT is exceeded. The Smoke Detector utilizes the SAS dictionary table (dictionary.tables) to validate the &WHERE clause, including exception handling that ensures at least one SAS library contains persistent files to be monitored.

```
    proc sql;
          select unique libname
          into :autoliblist separated by ' '
          from dictionary.tables
          where &where;
          quit;
    run;
%if &sqlobs=0 %then %do;
    %let errsmoke=No valid libraries were selected, or selected libraries contained
          no valid files;
    %goto err;
    %end;
```

The dictionary table (dictionary.tables) later is utilized to select all data sets to be processed. Note that in this example, only data sets of member type “DATA” are included, thus SAS views are excluded from monitoring.

```
proc sql;
    create table dic as
    select unique libname, memname, memtype, crdate, nobs, compress, filesize,
maxvar, num_character, num_numeric
    from dictionary.tables
    where &where and memtype= "DATA";
    quit;
run;
```

One of the first idiosyncrasies to be overcome is the inability of SAS natively to recognize data sets that are being modified. Whether the SAS dictionary tables or the CONTENTS procedure is utilized to produce a library inventory of data sets, all data sets that are being modified—by external sessions, processes, or users—will not be discovered. Thus, to get a more accurate inventory of data sets, the PIPE option is utilized within the FILENAME statement to ingest file metadata from the Windows operating system, which does recognize all SAS7BDAT files, even those which are being created, modified, or saved. The following code fragment demonstrates the use of the FILENAME statement to ingest a comprehensive list of SAS data sets.

```
%let path=%sysfunc(pathname(&lib));
filename indata pipe "dir %quote(&path) /a-d";
data modir (keep=libname memname memtype crdate);
    length libname $8 memname $32 memtype $8 memnamelong $100 crdate 8 inp $200;
    format crdate datetime17.;
    infile indata truncover;
    input inp $200.;
```

BEHIND THE SMOKE AND MIRRORS – INNER LOOP

The inner loop builds the three primary control tables (smoky.smoke_detector, smoky_smoke_detector_long, and smoky.smoke_detector_lock) in a single, heroic data step. One of the first objectives iteratively tests each monitored

data set for lock status. Within SAS, two types of locks are distinguished. Exclusive locks (i.e., read-write) are utilized when a data set is created or modified, such as the primary data set when a DATA step is implemented. While an exclusive lock is held by one user or process, no other user or process can access the data, neither directly to view it, nor even through the SAS dictionary tables, as discussed in the preceding section. A shared lock (i.e., read-only) is utilized when a data set only needs to be viewed but not modified. Multiple users and processes can access shared locks simultaneously, for example, allowing several users to sort the same data set, so long as they are sorting the results into distinct output data sets. Within the Smoke Detector, both shared and exclusive locks are monitored, which can demonstrate to SAS developers and administrators significant process bottlenecks that exist and where efficiency retrofitting efforts should be focused. At best, when properly handled through exception handling paradigms such as the %LOCKITDOWN macro, file locking will cause process delays, while unhandled file access collisions will stall and fail data processes.

In the following code, the FCLOSE I/O function quickly attempts to lock and unlock each data set and, when successful, demonstrates that no other lock exists. If unsuccessful, the OPEN I/O function next attempts to gain a shared lock and, when successful, demonstrates that only a shared lock already exists from another process. If no lock can be gained, an exclusive lock already exists on the data set. It should be noted that no existent locks are harmed in this iterative pinging process.

```
z=filename(tabref,pathname(libname) || '\' || strip(memname) || '.sas7bdat');
           lockexcl=fopen(tabref,"u");
           if lockexcl=1 then do; * exclusive lock gained, no locks elsewhere;
                  z=fclose(lockexcl);
                  lock_first=.;
                  lock_last=.;
                  lock_type=0;
                  end;
           else do;
                  lockshared=open(libname || '.' || memname);
                  if lockshared=1 then do;
                         z=close(lockshared);
                         lock_type=1; * shared lock gained, thus s-lock elsewhere;
                         end;
                  else do; * no locks gained, thus exclusive lock elsewhere;
                         lock_type=2;
                         end;
                  end;
```

After both the data structure and lock status have been interrogated for each data each data set, codified business rules create all alerts, which include notes, warnings, and errors. As mentioned, these alerts can be as creative as the developers engineering them. For each library, alerts cumulatively accrue every hour to ensure that all threats to data quality are recorded. After each hour, the alert notifications are cleared, but the historical data persists in the smoky.smoke_detector_long data set. The sample business rules included in the Smoke Detector include both single variable alerts, such as the warning generated when a data set has no observations, as well as multivariate alerts, such as the comparison that occurs between the current and previous number of numeric fields in a data set. These two alerts are illustrated in the following code, which produces text descriptions of the specific alerts to be included in later HTML reporting. Notes, warnings, and errors are distinguished and accrue in separate streams to facilitate color-coded stoplight reporting.

```
if num_numeric^=num_numeric2 then tab_err=strip(tab_err) ||
    strip(put(timepart(&lockloopstart),hhmm5.)) || ' - ERROR - # of num vars
    changed from ' || strip(num_numeric) || ' to ' || strip(num_numeric2) ||
    byte(13);
if nobs2=0 then tab_war=strip(tab_war) ||
    strip(put(timepart(&lockloopstart),hhmm5.)) || ' - WARNING - table has zero
    observations' || byte(13);
```

QUALITY CONTROL REPORTS

Two sample types of quality control reports are demonstrated in the Smoke Detector code, at the library level and data set level. Both reports exhibit stoplight reporting in which notes and normal functioning are colored green, warnings are colored yellow, and errors are colored red. Hovering over any cell initiates a popup window that displays succinct text summaries of each alert, including the time it was encountered. The library-level report contains cumulative, quantitative metrics summarizing all inclusive data sets, as well as hourly stoplight reporting.

Table 1 demonstrates a sample library-level report that has been run for three hours, from 11 AM until at least 2 PM. The two warnings in column 11 are typical of initializing the Smoke Detector, because all new data sets receive a warning. Column 13 for the Chicago library is pink, indicating that an error has been encountered in at least one of its 4 data sets. Other notes and warnings may also have occurred in some or all of the Chicago data sets, but if at least one error is present, the cell color will turn pink. By hovering over any of the stoplight cells, the list of notes, warnings, and errors will be viewable. Thus, alert inheritance is propagated from the data set to the library level.

				10-06-2014		
				11	12	13
Library	**Tables**	**Observations**	**File Size (MB)**			
TEST	3	32,546	5.34			
CHICAGO	4	21,109	9.21			

Table 1. Library-level Smoke Detector HTML report

If more information is desired about specific data sets within a library, selecting the library link will navigate to the respective member-level report, as indicated in Table 2. Member-level reports exhibit identical stoplight reporting as library-level reports, albeit specific to each data set. In this example, after further investigation, the member-level report demonstrates that data sets One and Two performed as expected from 12 PM until 2PM (as indicated by the green coloring), which also might indicate they were not modified during this period. However, a warning was generated for data set Three between 1 PM and 2PM, and an error was generated for data set Four between 1 PM and 2 PM. Hovering over cell Three-13 might further indicate that the number of observations dropped to zero, hence the warning, while cell Four-13 might indicate that the data set had been deleted.

		10-06-2014		
		11	12	13
Library	**Table**			
Chicago	**ONE**			
	TWO			
	THREE			
	FOUR			

Table 2. Member-level Smoke Detector HTML report

NEXT STEPS

Because all library- and member-level reports are generated dynamically, the Smoke detector represents a rapidly deployable solution that instantly can begin to monitor a diverse array of data sets spanning limitless libraries. While the Smoke Detector does not provide protection or ensure data quality, it inherently provides out-of-the-box warning and monitoring of risks to data quality, including threats to data availability and completeness. Notwithstanding its utility, the Smoke Detector represents a generic structure from which more complex and intelligent business rules should be depended. Moreover, the dual levels of validation—at both the library and member levels—demonstrate its versatility and ability to “drill down” into quality metrics.

The Smoke Detector's sole reliance on the SAS dictionary table (dictionary.tables) tremendously limits the scope of data structure validation that can be provided, because significant member- and field-level metadata are unavailable in this table. Thus, a future step to increase its value would be to incorporate a third level of data structure validation at the field level that could monitor field attribute changes, such as modifications to field type, length, format, label, or sort criteria. This information would be invaluable to further assess the availability and structure of data sets to gauge their quality and stability. With field-level validation in place, the fourth and final level of validation would entail value-level quality control methods that could address the correctness or accuracy of the data. For example, data constraints and other complex business rules could demonstrate (and possibly eliminate) data that fell outside established norms. And, in keeping with the alert inheritance propagated within the Smoke Detector, a single violation of a data value constraint could alert stakeholders that a quality assurance process has failed. With each level of quality control, processing time inherently will increase, but the business intelligence and peace of mind gained through quality control feedback might just outweigh those precious minutes.

CONCLUSION

Data quality lies at the heart of data analytics because analysis is only as good as the quality of its data. In data analytics development, quality assurance methods are a common way to facilitate reliable, robust ETL processes and data flows that are flexible and fault-tolerant. However, it is not enough only to plan for and hope that stated quality requirements will be met—data products must be validated against stated requirements, including quality standards. Thus, quality control processes should be emplaced that can monitor aspects of quality, such as data availability, completeness, timeliness, and correctness. The Smoke Detector represents a scalable, extensible quality control dashboard that instantly can identify and visualize threats to data quality to key stakeholders. Moreover, through the identification and tracking of shared and exclusive file locks on SAS data sets, it facilitates discovery of processes that may unnecessarily, accidentally, or inefficiently request file locks and which cause process timeout or failure.

APPENDIX A. SMOKE DETECTOR (AKA SMOKY) CODE

```
libname smoky '/folders/myfolders/smoky/';
%include '/folders/myfolders/lockitdown.sas';

%macro smoky (mintot = /* number of minutes code executes before terminating */,
    libloopmax = /* frequency (secs) of structure (dictionary) queries */,
    lockloopmax = /* frequency (secs) of locking queries */,
    lockwarthresh = /* seconds after which lock warning alert will occur */,
    lockerrthresh = /* seconds after which lock error alert will occur */,
    hrsretain = /* hours after which historic records are deleted */,
    liblist = /* space-delimited list of libraries to be included */,
    tabexclude = /* space-delimited list of data sets to exclude */,
    outputloc = /* directory where output reports are created */);

*proc printto log='/folders/myfolders/smoky/log.txt';
%local i where;
%global errsmoke;
%let errsmoke=;
%let lockloopstart=;
%let liblist=%upcase(&liblist);
%let tabexclude=%upcase(&tabexclude);
%let where=;
%let i=1;
* create library and table inclusion and exclusion criteria;
%do %while(%length(%scan(&liblist,&i,,S))>1);
    %if &i=1 %then %let where=&where LIBNAME IN(;
    %else %let where=&where ,;
    %let where=&where "%scan(&liblist,&i,,S)";
    %let i=%eval(&i+1);
    %end;
%let where=&where );
%let i=1;
%do %while(%length(%scan(&tabexclude,&i,,S))>1);
    %let excludes=%scan(&tabexclude,&i,,S);
    %let where=&where and ^(libname="%scan(&excludes,1,.)" and
memname="%scan(&excludes,2,.)");
    %let i=%eval(&i+1);
    %end;
%let starttime=%sysfunc(datetime());
%if %sysfunc(exist(smoky.smoke_detector))=0 %then %do;
    %lockitdown(lockfile=smoky.smoke_detector, sec=1, max=600, type=w,
canbemissing=y);
    data smoky.smoke_detector;
          length libname $8 memname $32 memtype $8 crdate 8 filesize 8 nobs 8
compress $8 maxvar 8 num_character 8
                num__numeric 8 lock_first lock_last lock_type lastlocktest
deleted_first deleted_last last_verified 8;
          format libname $8. memname $32. memtype $32. crdate datetime17. filesize
nobs comma15. compress $8. maxvar
                num_character num_numeric 8. lock_excl lock_shared lock_type 8.
lock_first lock_last lastlocktest
                deleted_first deleted_last last_verified datetime17.;
    run;
    &lockclr;
```

```
    %end;
%lockitdown(lockfile=smoky.smoke_detector_long, sec=1, max=600, type=w,
canbemissing=y);
%if %sysfunc(exist(smoky.smoke_detector_long))=0 %then %do;
    data smoky.smoke_detector_long;
           length libname $8 memname $32 memtype $8 timestamp tstime tsdate crdate
last_verified filesize nobs 8 compress $8
                  maxvar num_character num_numeric 8 tab_not tab_war tab_err $800;
           format libname $8. memname $32. memtype $8. timestamp crdate
last_verified datetime17. tsdate mmddyyd10. tstime 8.
                  filesize nobs comma15. compress $8. maxvar num_character
num_numeric 8. tab_not tab_war tab_err $800.;
    run;
    %end;
%else %do;
    data smoky.smoke_detector_long;
           set smoky.smoke_detector_long;
           where &where;
    run;
    %end;
&lockclr;
%lockitdown(lockfile=smoky.smoke_detector_lock, sec=1, max=600, type=w,
canbemissing=y);
%if %sysfunc(exist(smoky.smoke_detector_lock))=0 %then %do;
    data smoky.smoke_detector_lock;
           length libname $8 memname $32 memtype $8 lock_war lock_err $800 timestamp
tstime tsdate lock_first lock_last lock_type 8;
           format libname $8. memname $32. memtype $8. lock_war lock_err $800.
timestamp lock_first lock_last datetime17.
                  tstime lock_type 8. tsdate mmddyyd10.;
    run;
    %end;
%else %do;
    data smoky.smoke_detector_lock;
           set smoky.smoke_detector_lock;
           where &where;
    run;
    %end;
&lockclr;

/* BEGIN OUTER LOOP */

%let i=1;
%do %until (%sysevalf(%sysfunc(datetime())>(&starttime+&mintot)));
    %let libloopstart=%sysfunc(datetime());
    proc sql;
           select unique libname
           into :autoliblist separated by ' '
           from dictionary.tables
           where &where;
           quit;
    run;
%if &sqlobs=0 %then %do;
    %let errsmoke=No valid libraries were selected, or selected libraries contained
no valid files;
```

```
    %goto err;
    %end;
data dir;
    length libname $8 memname $32 memtype $8 crdate 8 memnamelong $100 crdate 8 inp
$200;
    format crdate datetime17.;
run;
%if &sysscp=WIN %then %do;
    %let j=1;
    %do %while(%length(%scan(&autoliblist,&j))>1);
           %let lib=%scan(&autoliblist,&j);
           * confirm which tables are present through pipe;
           %let path=%sysfunc(pathname(&lib));
           filename indata pipe "dir %quote(&path) /a-d";
           data modir (keep=libname memname memtype crdate);
                  length libname $8 memname $32 memtype $8 memnamelong $100 crdate 8
inp $200;
                  format crdate datetime17.;
                  infile indata truncover;
                  input inp $200.;
                  if upcase(scan(inp,-1,.))^='SAS7BDAT' then delete;
                  else do;
                         crdate=input(substr(inp,1,20),mdyampm20.);
                         libname=upcase ("&lib");
                         memtype= 'DATA';
                         memnamelong=upcase(substr(inp,39,length(inp)-38));
                         if length(memnamelong)>15 then do;
                                if
upcase(substr(strip(memnamelong),length(strip(memnamelong))-12,5))='COPY.' then
delete;
                                end;
                         memname=strip(substr(memnamelong,1,length(memnamelong)-9));
                         end;
           run;
           proc append base=dir data=modir;
           run;
           %let j=%eval(&j+1);
           %end;
    %end;
* confirm tables present through dictionary;
proc sql;
    create table dic as
    select unique libname, memname, memtype, crdate, nobs, compress, filesize,
maxvar, num_character, num_numeric
    from dictionary.tables
    where &where and memtype= "DATA";
    quit;
run;
proc sort data=dic;
    by libname memname memtype;
run;
proc sort data=dir;
    by libname memname memtype;
run;
```

```
/* BEGIN INNER LOOP */

* join and compare both directory and dictionary Lables and test all locks;
%do %until (%sysevalf(%sysfunc(datetime())>&libloopstart+&libloopmax));
    %let lockloopstart=%sysfunc(datetime());
    %lockitdown(lockfile=smoky.smoke_detector, sec=1, max=600, type=w,
canbemissing=n);
    data smoky.smoke_detector (drop=z lockexcl lockshared timestamp tstime tsdate
tabref crdate2 nobs2 compress2 filesize2
                maxvar2 num_character2 num_numeric2 lockexcl lockshared
lock_type_old lock_excl lock_shared crdate1)
          smoke_detector_long (keep=libname memname memtype timestamp tstime tsdate
crdate nobs compress filesize
                num_character num_numeric maxvar last_verified tab_not tab_war
tab_err)
          smoke_detector_lock (keep=libname memname memtype timestamp tstime tsdate
lock_type
                lock_first lock_last lock_war lock_err);
          merge smoky.smoke_detector (in=a keep=libname memname memtype crdate nobs
compress filesize num_character num_numeric maxvar
                lock_first lock_last lastlocktest deleted_first deleted_last
last_verified lock_type
                rename=(lock_type=lock_type_old))
          dir (in=b rename=(crdate=crdate1))
          dic (in=c rename=(crdate=crdate2 nobs=nobs2 compress=compress2
filesize=filesize2
                maxvar=maxvar2 num_character=num_character2
num_numeric=num_numeric2));
          by libname memname memtype;
          where ^missing(libname) and &where;
          length tabref $8 tab_not tab_war tab_err lock_not lock_war lock_err $800
lock_first lock_last lock_type lock_type_old 8;
          format libname $8. memname $32. memtype $8. timestamp crdate datetime17.
tsdate mmddyyd10. tstime 8. filesize nobs comma15.
                compress $8. maxvar 8. num_character 8. num_numeric 8. lock_excl
lock_shared lock_type lock_type_old 8. lock_first lock_last
                lastlocktest deleted_first deleted_last last_verified datetime17.;
          in_smoke=a;
          in_dir=b;
          in_dic=c;
          timestamp=&lockloopstart;
          tsdate=datepart(timestamp);
          tstime=hour(timepart(timestamp));
          tabref='tab' || strip(put(_n_,$8.));
          if in_dir=0 then do; *if table has been deleted;
                if missing(deleted_first) then deleted_first=&lockloopstart;
                deleted_last=&lockloopstart;
                tab_err=strip(tab_err) ||
strip(put(timepart(&lockloopstart),hhmm5.)) ||
                      ' - ERROR - table deleted ' || byte(13);
                if deleted_first=deleted_last then output smoke_detector_long; *
only output first time shown deleted;
                lock_first=.;
                lock_last=.;
                end;
```

```
          else do; * if table exists;
                if ^missing(deleted_first) then do; * if deleted table has
reappeared;
                      deleted_first=.;
                      deleted_last=.;
                      end;
                * test for locks;
                lastlocktest=&lockloopstart;
                z=filename(tabref,pathname(libname) || '\' || strip(memname) ||
'.sas7bdat');
                lockexcl=fopen(tabref,"u");
                if lockexcl=1 then do; * exclusive lock gained, no locks
elsewhere;
                      z=fclose(lockexcl);
                      lock_first=.;
                      lock_last=.;
                      lock_type=0;
                      end;
                else do;
                      lockshared=open(libname || '.' || memname);
                      if lockshared=1 then do;
                            z=close(lockshared);
                            lock_type=1; * only a shared lock gained, thus shared
lock elsewhere;
                            end;
                      else do; * no locks gained, thus exclusive lock elsewhere;
                            lock_type=2;
                            end;
                      end;
                if lock_type=1 or lock_type=2 then do;
                      if missing(lock_first) or lock_type^=lock_type_old then
lock_first=&lockloopstart;
                      lock_last=&lockloopstart;
                      end;
                if lock_last-lock_first > &lockerrthresh then
lock_err=strip(put(timepart(&lockloopstart),hhmm5.))
                      || ' - ERROR - table ' || strip(memname) || ' ' ||
ifc(lock_type=1,'read-only','exclusively') || ' locked for '
                      || strip(round((lock_last-lock_first)/60)) || ' minutes';
                else if lock_last-lock_first > &lockwarthresh then
lock_war=strip(put(timepart(&lockloopstart),hhmm5.))
                      || ' - WARNING - table ' || strip(memname) || ' ' ||
ifc(lock_type=1,'read-only','exclusively') || ' locked for '
                      || strip(round((lock_last-lock_first)/60)) || ' minutes';
                else if lock_last-lock_first > 0 then
lock_not=strip(put(timepart(&lockloopstart),hhmm5.))
                      || ' - NOTE - table ' || strip(memname) || ' ' ||
ifc(lock_type=1,'read-only','exclusively') || ' locked for '
                      || strip(round((lock_last-lock_first)/60)) || ' minutes';
                if lock_last-lock_first > &lockwarthresh then output
smoke_detector_lock; * output only warnings and errors;
                if in_dic=1 then do;
                      last_verified=&lockloopstart;
                      if ^missing(filesize2) and (crdate^=crdate2 or nobs^=nobs2
or filesize^=filesize2) then do;
```

```
                                if missing(filesize) then tab_war=strip(tab_war) ||
strip(put(timepart(&lockloopstart),hhmm5.))
                                      || ' - WARNING - new table' || byte(13);
                                else do;
                                      * errors;
                                      if num_character^=num_character2 then
tab_err=strip(tab_err) ||

    strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - ERROR - # of char vars changed from
' || strip(num_character) ||
                                            ' to ' || strip(num_character2) ||
byte(13);
                                      if num_numeric^=num_numeric2 then
tab_err=strip(tab_err) ||

    strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - ERROR - # of num vars changed from
' || strip(num_numeric) ||
                                            ' to ' || strip(num_numeric2) ||
byte(13);
                                      * warnings;
                                      if nobs2=0 then tab_war=strip(tab_war) ||
strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - WARNING - table has zero
observations' || byte(13);
                                      if filesize2<filesize then
tab_war=strip(tab_war) || strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - WARNING - table decreased in size
from ' || strip(filesize) || ' to ' ||
                                            strip(filesize2) || byte(13);
                                      if nobs2<nobs then tab_war=strip(tab_war) ||
strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - WARNING - number of observations
decreased from ' || strip(nobs) || ' to ' ||
                                            strip(nobs2) || byte(13);
                                      if compress^=compress2 then
tab_war=strip(tab_war) ||

    strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - WARNING - compression status
changed from ' || strip(compress) ||
                                            ' to ' || strip(compress2) || byte(13);
                                      * notes;
                                      tab_not=strip(tab_not) ||
strip(put(timepart(&lockloopstart),hhmm5.)) ||
                                            ' - NOTE - table updated from ' ||
strip(put(crdate,datetime17.)) || byte(13);
                                      end;
                                crdate=crdate2;
                                nobs=nobs2;
                                filesize=filesize2;
                                compress=compress2;
                                num_character=num_character2;
                                num_numeric=num_numeric2;
```

```
                              maxvar=maxvar2;
                              lock_type=lock_type_old;
                              output smoke_detector_long; * output if table
updated;
                              end;
                        end;
                  else do; * when table is both new and locked;
                        if crdate^=crdate1 then do;
                              tab_war=strip(tab_war) ||
strip(put(timepart(&lockloopstart),hhmm5.))
                                    || " - WARNING - new table" || byte (13);
                              if nobs2=0 then tab_war=strip(tab_war) || strip
(put(timepart(&lockloopstart),hhmm5.)) ||
                                    " - WARNING - table has zero observations" ||
byte(13);
                              crdate=crdate1;
                              output smoke_detector_long;
                              end;
                        end;
                  end;
            output smoky.smoke_detector;
      run;
      &lockclr;
      %lockitdown(lockfile=smoky.smoke_detector_lock, sec=1, max=600, type=w,
canbemissing=n);
      proc append base=smoky.smoke_detector_lock data=smoke_detector_lock;
      run;
      &lockclr;
      %lockitdown(lockfile=smoky.smoke_detector_long, sec=1, max=600, type=w,
canbemissing=n);
      proc append base=smoky.smoke_detector_long data=smoke_detector_long;
      run;
      &lockclr;
      * put processor to sleep until outer loop should begin again;
      %do %until (%sysevalf(%sysfunc(datetime()) > &lockloopstart + &lockloopmax));
            %if &sysscp=WIN %then %let z=%sysfunc(sleep(5));
            %else %do;
                  data _null_;
                        call sleep(5,1);
                  run;
                  %end;
            %end;
      %end;
      %lockitdown(lockfile=smoky.smoke_detector_lock, sec=1, max=600, type=w,
canbemissing=n);
      proc sort data=smoky.smoke_detector_lock;
            by libname memname memtype tsdate tstime;
            where ^missing(libname) and (timestamp > intnx("hour",&lockloopstart, -
&hrsretain));
      run;
      data smoky.smoke_detector_lock_rpt (drop=timestamp lock_first lock_last
lock_type lock_first_old lock_war_old lock_err_old
                  lock_war_tmp lock_err_tmp);
            set smoky.smoke_detector_lock;
            by libname memname memtype tsdate tstime;
```

```
          length lock_first_old 8 lock_war_old lock_err_old lock_war_tmp
lock_err_tmp $1000;
          format lock_first_old datetime17.;
          * lock warnings and errors are continuously output, so only the final
should appear;
          if ^missing(lock_first) then do;
                if first.tstime then do;
                      lock_war_old='';
                      lock_err_old='';
                      lock_war_tmp='';
                      lock_err_tmp='';
                      lock_first_old='';
                      end;
                if lock_first=lock_first_old or missing(lock_first_old) then do;
                      if ^missing(lock_war) and ^missing(lock_err) then
lock_war='';
                      lock_war_old=lock_war;
                      lock_err_old=lock_err;
                      end;
                else do;
                      lock_war_old='';
                      lock_err_old='';
                      end;
                if (^missing(lock_first_old) and lock_first^=lock_first_old) or
last.tstime then do;
                      if length(lock_war_old)>1 then
lock_war_tmp=strip(lock_war_tmp) || strip(lock_war_old) || byte(13);
                      if length(lock_err_old)>1 then
lock_err_tmp=strip(lock_err_tmp) || strip(lock_err_old) || byte(13);
                      end;
                if last.tstime then do;
                      lock_war=strip(lock_war_tmp);
                      lock_err=strip(lock_err_tmp);
                      end;
                end;
          lock_first_old=lock_first;
          retain lock_first_old lock_war_old lock_err_old lock_war_tmp
lock_err_tmp;
          if last.tstime then do;
                lock_war=strip(lock_war_tmp);
                lock_err=strip(lock_err_tmp);
                len=length(lock_err);
                output;
                end;
    run;
    &lockclr;
    %lockitdown(lockfile=smoky.smoke_detector_long, sec=1, max=600, type=w,
canbemissing=n);
    proc sort data=smoky.smoke_detector_long;
          by libname memname memtype tsdate tstime;
          where ^missing(libname) and (timestamp> intnx("hour",&lockloopstart,-
&hrsretain));
    run;
    * subsets data so only last observation per hour is displayed, but retains
comments from all obs;
```

```
    data smoky.smoke_detector_long_rpt (rename=(tab_not_tmp=tab_not
tab_war_tmp=tab_war tab_err_tmp=tab_err)
                  drop=max_date max_time last_lib last_mem last_nobs last_size
last_crdate last_compress last_maxvar
                        last_char last_num max_nobs max_size max_compress
max_maxvar
                        max_char max_num i j newdate last_date tab_not tab_war
tab_err max_tab_not
                        max_tab_war max_tab_err lock_war_tmp lock_err_tmp
lock_err_old lock_war_old
                        lock_first_old last_date last_time exp_date exp_time
last_type
                        max_timestamp max_crdate flyover_tmp first_mem);
           set smoky.smoke_detector_long;
           by libname memname memtype tsdate tstime timestamp;
           length tab_not_tmp tab_war_tmp tab_err_tmp lock_war_tmp lock_err_tmp
lock_war_old lock_err_old
                        max_tab_not max_tab_war max_tab_err $2000 flyover
flyover_tmp $5000 libhref $200
                        lock_first_old last_date last_time exp_date exp_time
max_date max_time 8
                        last_lib last_compress last_type $8 last_mem $32;
           format lock_first_old datetime17. exp_date max_date last_date mmddyy10.;
           libhref="<a href='" || "&outputloc.smoke_detector_" || strip(libname) ||
".htm'>" || strip(libname) || "</a>";
           * reset variables;
           if first.memtype then do;
                  last_date=.;
                  last_time=.;
                  last_nobs=.;
                  last_size=.;
                  last_crdate=.;
                  last_compress='';
                  last_maxvar=.;
                  last_char=.;
                  last_num=.;
                  first_mem=0;
                  end;
           if first.tstime then do;
                  tab_not_tmp='';
                  tab_war_tmp='';
                  tab_err_tmp='';
                  lock_war_tmp='';
                  lock_err_tmp='';
                  lock_war_old='';
                  lock_err_old='';
                  lock_first_old=.;
                  end;
           if length(tab_not)>1 then tab_not_tmp=strip(tab_not_tmp) ||
strip(tab_not);
           if length(tab_war)>1 then tab_war_tmp=strip(tab_war_tmp) ||
strip(tab_war);
           if length(tab_err)>1 then tab_err_tmp=strip(tab_err_tmp) ||
strip(tab_err);
           if last.tstime then do;
```

```
                if length(tab_err_tmp)>1 then flyover=strip(flyover) || '***
ERRORS ***' || byte(13) ||
                      strip(tab_err_tmp) || byte(13);
                if length(tab_war_tmp)>1 then flyover=strip(flyover) || '***
WARNINGS ***' || byte(13) ||
                      strip(tab_war_tmp) || byte(13);
                if length(tab_not_tmp)>1 then flyover=strip(flyover) || '*** NOTES
***' || byte(13) ||
                      strip(tab_not_tmp) || byte(13);
                if first_mem=0 then first_mem=.;
                else do;
                      i=0;

    exp_date=datepart(intnx('hour',dhms(last_date,last_time,0,0),1));

    exp_time=hour(intnx('hour',dhms(last_date,last_time,0,0),1));
                      do while((^(tsdate=exp_date and tstime=exp_time) and
(^(tsdate=max_date and
                                  tstime=max_time and i^=0))) and i<100);
                           if i=0 then do;
                                  max_date=tsdate;
                                  max_time=tstime;
                                  max_nobs=nobs;
                                  max_timestamp=timestamp;
                                  max_size=filesize;
                                  max_crdate=crdate;
                                  max_compress=compress;
                                  max_maxvar=maxvar;
                                  max_char=num_character;
                                  max_num=num_numeric;
                                  max_tab_not=tab_not_tmp;
                                  max_tab_war=tab_war_tmp;
                                  max_tab_err=tab_err_tmp;
                                  flyover_tmp=flyover;
                                  end;
                           i=i+1;
                           tsdate=exp_date;
                           tstime=exp_time;
                           if ^(tsdate=max_date and tstime=max_time) then do;
                                  nobs=last_nobs;
                                  filesize=last_size;
                                  crdate=last_crdate;
                                  compress=last_compress;
                                  maxvar=last_maxvar;
                                  num_character=last_char;
                                  num_numeric=last_num;
                                  timestamp=.;
                                  tab_not_tmp='';
                                  tab_war_tmp='';
                                  tab_err_tmp='';
                                  flyover='';
                                  output;
                                  end;

    exp_date=datepart(intnx('hour',dhms(tsdate,tstime,0,0),1));
```

```
    exp_time=hour(intnx('hour',dhms(tsdate,tstime,0,0),1));
                                end;
                          if i>0 then do;
                                tsdate=max_date;
                                tstime=max_time;
                                nobs=max_nobs;
                                timestamp=max_timestamp;
                                filesize=max_size;
                                crdate=max_crdate;
                                compress=max_compress;
                                maxvar=max_maxvar;
                                num_character=max_char;
                                num_numeric=max_num;
                                tsdate=max_date;
                                tstime=max_time;
                                tab_not_tmp=max_tab_not;
                                tab_war_tmp=max_tab_war;
                                tab_err_tmp=max_tab_err;
                                flyover=flyover_tmp;
                                end;
                          end;
                    if last.memtype then do;
                          max_date=datepart(&lockloopstart);
                          max_time=hour(&lockloopstart);
                          output;
                          j=0;
                          do while (^(tsdate=max_date and tstime=max_time) and
j<100);
                                newdate=intnx('hour',dhms(tsdate,tstime,0,0),1);
                                tsdate=datepart(newdate);
                                tstime=hour(newdate);
                                j=j+1;
                                timestamp=.;
                                tab_not_tmp='';
                                tab_war_tmp='';
                                tab_err_tmp='';
                                flyover='';
                                output;
                                end;
                          end;
                    else output;
                    last_lib=libname;
                    last_mem=memname;
                    last_type=memtype;
                    last_date=tsdate;
                    last_time=tstime;
                    last_nobs=nobs;
                    last_size=filesize;
                    last_crdate=crdate;
                    last_compress=compress;
                    last_maxvar=maxvar;
                    last_char=num_character;
                    last_num=num_numeric;
                    end;
```

```
          retain tab_not_tmp tab_war_tmp tab_err_tmp last_lib last_mem last_type
last_date last_time
                last_nobs last_size last_crdate last_compress last_maxvar
last_char last_num first_mem;
          run;
          &lockclr;
    data smoky.smoke_detector_historical;
          merge smoky.smoke_detector_long_rpt (in=a)
                smoky.smoke_detector_lock_rpt (in=b);
          by libname memname memtype tsdate tstime;
          if length(lock_err)>1 then flyover=strip(flyover) || '*** LOCK ERRORS
***' || byte (13)
                || strip(lock_err) || byte(13);
          if length(lock_war)>1 then flyover=strip(flyover) || '*** LOCK WARNINGS
***' || byte (13)
                || strip(lock_war) || byte(13);
    run;
%let lockloopstart=%sysfunc(dhms("01OCT2014"d,18,0,0));
%put &lockloopstart;

    * create table suset only by library, not by table name;
    proc sort data=smoky.smoke_detector_historical out=smoke_detector_lib_rpt;
          by libname tsdate tstime;
    run;
    data smoky.smoke_detector_lib_rpt (keep=libname libhref tsdate tstime
                tab_not_long tab_war_long tab_err_long lock_war_long lock_err_long
flyover performance)
                lib_temp (keep=libname tabtot sizetot obstot);
          set smoke_detector_lib_rpt;
          by libname tsdate tstime;
          length tabtot sizetot obstot dtg_cut 8 tab_not_long tab_war_long
tab_err_long lock_war_long
                lock_err_long $800 performance $20;
          format dtg_cut datetime17.;
          dtg_cut=intnx('hour',&lockloopstart,-10);
          if first.tstime then do;
                tab_not_long='';
                tab_war_long='';
                tab_err_long='';
                lock_war_long='';
                lock_err_long='';
                tabtot=0;
                sizetot=0;
                obstot=0;
                end;
          tabtot=tabtot+1;
          sizetot=sum(sizetot,filesize);
          obstot=sum(obstot,nobs);
          if length(tab_not)>1 then tab_not_long=strip(tab_not_long) ||
strip(tab_not);
          if length(tab_war)>1 then tab_war_long=strip(tab_war_long) ||
strip(tab_war);
          if length(tab_err)>1 then tab_err_long=strip(tab_err_long) ||
strip(tab_err);
```

```
          if length(lock_war)>1 then lock_war_long=strip(lock_war_long) ||
strip(lock_war);
          if length(lock_err)>1 then lock_err_long=strip(lock_err_long) ||
strip(lock_err);
          if first.libname then lib=1;
          if last.libname then lib=2;
          if last.tstime then do;
                flyover='';
                if length(tab_err_long)>1 then flyover=strip(flyover) || '***
ERRORS ***' || byte(13) ||
                      strip(tab_err_long) ||    byte(13);
                if length(lock_err_long)>1 then flyover=strip(flyover) || '***
LOCK ERRORS ***' || byte(13) ||
                       strip(lock_err_long) || byte(13);
                if length(tab_war_long)>1 then flyover=strip(flyover) || '***
WARNINGS ***' || byte(13) ||
                       strip(tab_war_long) || byte (13);
                if length(lock_war_long)>1 then flyover=strip(flyover) || '***
LOCK WARNINGS ***' || byte(13) ||
                      strip(lock_war_long) || byte(13);
                if length(tab_not_long)>1 then flyover=strip(flyover) || '***
NOTES ***' || byte(13) ||
                      strip(tab_not_long) || byte(13);
                output smoky.smoke_detector_lib_rpt;
                if lib=2 then output lib_temp;
                end;
          retain tabtot sizetot obstot tab_not_long tab_war_long tab_err_long
lock_war_long lock_err_long lib;
    run;
    data smoky.smoke_detector_lib_rpt;
          merge smoky.smoke_detector_lib_rpt lib_temp;
          by libname;
    run;
    * determine how many days and hours are present;
    proc sql;
          select count(distinct tsdate), count(distinct tstime)
          into :days, :hrs
          from smoky.smoke_detector_lib_rpt;
          quit;
    run;
    * create main report *;
    ods listing close;
    ods noproctitle;
    ods html path="&outputloc" file="smoke_detector.htm" style=sketch;
    proc report data=smoky.smoke_detector_lib_rpt nocenter nowindows missing
nocompletecols
                style(report) = [foreground=black backgroundcolor=black]
                style(header) = [font_size=2 backgroundcolor=white
foreground=black];
          title h=2 "Libraries: %sysfunc(datetime(),datetime17.)";
          column libname libhref tabtot obstot sizetot tsdate,tstime, (tab_not_long
tab_war_long tab_err_long
                lock_war_long lock_err_long flyover performance) dummy;
          define libname / group noprint '';
          define libhref / group 'Library';
```

```
          define tabtot / group 'Tables';
          define obstot / group 'Observations' format=comma15.;
          define sizetot / group 'File Size (MB)' format=comma15.;
          define tsdate / across '';
          define tstime / across '';
          define tab_not_long / group '' noprint;
          define tab_war_long / group '' noprint;
          define tab_err_long / group '' noprint;
          define lock_war_long / group '' noprint;
          define lock_err_long / group '' noprint;
          define performance / group '';
          define flyover / group '' noprint;
          define dummy / computed noprint '';
          compute dummy;
          %put DAYS HOURS &days &hrs;
          %do a=1 %to &days;
                %do b=1 %to &hrs;
                if length (_c%sysevalf(5+((&b-1) * 7)+1+((&a-1)*&hrs*7))_)>1 then
do;
                      call define("_c%sysevalf(5+(&b * 7)+((&a-
1)*&hrs*7))_",'style',
                            'style=[backgroundcolor=very light green flyover="'||
                            strip(_c%sysevalf(5+((&b-1) * 7)+6+((&a-
1)*&hrs*7))_)||'"]');
                      end;
                if length (_c%sysevalf(5+((&b-1) * 7)+4+((&a-1)*&hrs*7))_)>1
                      or length (_c%sysevalf(5+((&b-1) * 7)+2+((&a-
1)*&hrs*7))_)>1 then do;
                      call define ("_c%sysevalf(5+(&b * 7) + ((&a-
1)*&hrs*7))_",'style',
                            'style=[backgroundcolor=very light yellow
flyover="'||
                            strip(_c%sysevalf(5+((&b-1) * 7)+6+((&a-
1)*&hrs*7))_)||'"]');
                      end;
                if length (_c%sysevalf(5+((&b-1) * 7)+3+((&a-1)*&hrs*7))_)>1
                      or length(_c%sysevalf(5+((&b-1) * 7)+5+((&a-1)*&hrs*7))_)>1
then do;
                      call define("_c%sysevalf(5+(&b * 7)+((&a-
1)*&hrs*7))_",'style',
                            'style=[backgroundcolor=very light red flyover="'||
                            strip(_c%sysevalf(5+((&b-1) * 7)+6+((&a-
1)*&hrs*7))_)||'"]');
                      end;
                      %end;
                %end;
          endcomp;
    run;
    ods html close;
    ods listing;

* create library-specific reports;
    %let j=1;
    %do %while(%length(%scan(&autoliblist,&j))>1);
          %let lib=%scan(&autoliblist,&j);
```

```
          ods listing close;
          ods noproctitle;
          ods html path="&outputloc" file="smoke_detector_&lib..htm" style=sketch;
          proc report data=smoky.smoke_detector_historical nocenter nowindows
missing nocompletecols
                       style(report) = [foreground=black backgroundcolor=black]
                       style(header) = [font_size=2 backgroundcolor=white
foreground=black];
                 where upcase(libname) ="&lib";
                 title h=2 "Tables: %sysfunc(datetime() ,datetime17.)";
                 column libname memname tsdate, tstime, (tab_not tab_war lock_war
tab_err
                       lock_err flyover nobs) dummy;
                 define libname / group 'Library';
                 define memname / group 'Table';
                 define tsdate / across '';
                 define tstime / across '';
                 define tab_not / group '' noprint;
                 define tab_war / group '' noprint;
                 define lock_war / group '' noprint;
                 define tab_err / group  '' noprint;
                 define lock_err / group '' noprint;
                 define flyover / group '' noprint;
                 define nobs / display sum '' format=comma15.;
                 define dummy / computed noprint;
                 compute dummy;
                 %do a=1 %to &days;
                       %do b=1 %to &hrs;
                             if length(_c%sysevalf(2+((&b-1) * 7)+1+((&a-
1)*&hrs*7))_)>1 then do;
                                   call define("_c%sysevalf(2+((&b-1) *
7)+7+((&a-1)*&hrs*7))_",
                                         'style','style=[backgroundcolor=very
light green flyover="'||
                                         strip(_c%sysevalf(2+((&b-1) *
7)+6+((&a-1)*&hrs*7))_)||'"]');
                                   end;
                             if length(_c%sysevalf(2+((&b-1) * 7)+2+((&a-
1)*&hrs*7))_)>1
                                   or length(_c%sysevalf(2+((&b-1) * 7)+3+((&a-
1)*&hrs*7))_)>1 then do;
                                   call define("_c%sysevalf(2+((&b-1) *
7)+7+((&a-1)*&hrs*7))_",
                                         'style','style=[backgroundcolor=very
light yellow flyover="'||
                                         strip(_c%sysevalf(2+((&b-1) *
7)+6+((&a-1)*&hrs*7))_)||'"]');
                                   end;
                             if length(_c%sysevalf(2+((&b-1) * 7)+4+((&a-
1)*&hrs*7))_)>1
                                   or length(_c%sysevalf(2+((&b-1) * 7)+5+((&a-
1)*&hrs*7))_)>1 then do;
                                   call define("_c%sysevalf(2+((&b-1) *
7)+7+((&a-1)*&hrs*7))_",
```

```
                                                    'style','style=[backgroundcolor=very
light red flyover="'||
                                                    strip(_c%sysevalf(2+((&b-1) *
7)+6+((&a-1)*&hrs*7))_)||'"]');
                                          end;
                                          %end;
                                  %end;

                    endcomp;
                    run;
                    ods html close;
                    ods listing;
               %let j=%eval(&j+1);
               %end;
     %let i=%eval(&i+1);
     %end;
%err: %put &errsmoke;
*proc printto;
%mend;

%smoky (mintot=3, libloopmax=120, lockloopmax=60, lockwarthresh=60,
lockerrthresh=300, hrsretain=24,
     outputloc=/folders/myfolders/smoky/,
     liblist=test,
     tabexclude=smoky.smoke_detector);
```

REFERENCES

[i] Hughes, Troy Martin. 2014. *Why Aren't Exception Handling Routines Routine? Toward Reliably Robust Code through Increased Quality Standards in Base SAS*. Midwest Users of SAS (MWSUG).

[ii] Hughes, Troy Martin. 2014. *When Software Development is Your Means Not Your End: Abstracting Agile Methodologies for End-User Development and Analytic Application*. Western Users of SAS Software (WUSS).

[iii] Hughes, Troy Martin. 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).

CONTACT INFORMATION

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

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