PharmaSUG 2011 - Paper AD10

Automating the Installation and Operational Qualifications of Your SAS Applications with the SAS System

Joe Perry, Perry & Associates Consulting, Oceanside, CA

ABSTRACT

Many industries require that new software installations go through checks that they have both been installed correctly (Installation Qualification or IQ) as well as operate as intended (Operational Qualification or OQ). Modest software development projects can use one or more brute force methods of performing or ensuring the critical software qualifications of the software placed in the hands of the end-user, the application, even for modest software applications, can tax the patience of any person or company.

As anyone who plans, executes or supports software validation activities knows: preparing, executing and documenting the validation processes can be time consuming and costly. Recognizing this, the SAS Institute has pre-packaged IQ and OQ routines that ease the task of performing installing SAS® in controlled environments, saving users considerable time and money.

With proper planning during the software development process, other SAS-based applications can take advantage of some of these same techniques and you can significantly improve the user's IQ and OQ experiences.

This paper discusses SAS-based automation methods to use for Installation and Operational Qualifications for use in your own software deployments; discussing and demonstrating simple methods to reduce the complexity, uncertainty and cost of installing software at your users' site.

BACKGROUND

Installing, configuring and performing: Installation Qualifications, First Use Operations and Operational Qualifications can be both time consuming and subject to poor work or testing processes: compress time and errors will increase for both documentation and the quality of the work, improve work or testing as well as the time required can bust your budget. Following a manufacturing maxim "never automate a manual process…", automation of these manual processes, though possible, will require some (possibly) significant changes to the overall system's architecture. For example, any application's installation directory must contain all of the files found in the system's 'manifest', whether that list is real or virtual. The Installation Qualification confirms that the expected files and locations are, in fact, where they need to be in order for the application to work correctly.

Also, some SAS installations use a customized startup icon; with users required to manually configure the SAS startup icon for configuration parameters like: SASINITIALFOLDER, AUTOEXEC, etc.; the *automated* configuration routines for this process will have to be accomplished by moving these steps into a code controllable routine, these will then become part of the overall application's code base.

Likewise, the Operational Qualification, most often written out and performed by an independent party, would usually require a modification of the system's code so it could: analyze, capture and report out the results of the, previously, manual tests performed within the OQ.

This paper looks at a typical application and discusses specific tasks and code for the application of SAS-based automation routines to handle the IQ and OQ routines, first we need to look at the process design specifications.

PROCESS DESIGN SPECIFICATIONS: INSTALLATION QUALIFICATIONS

Installation Qualification (IQ) is the documented verification that all key aspects of the installation adhere to approved intentions.[1] First, the IQ assumes that the application has been tested and works so, if automation is possible, for the purpose of this step, the IQ must consist of demonstrating that in fact the system was installed and starts. Operationally, this would imply that certain necessary tasks were performed (or, conversely, did not fail) so, let's look at some of the common failures or errors in performing the installation of an application. Experience in deploying software to users, who then install and *qualify* the software at their sites, shows that many, if not most, installation errors fall into three distinct categories: **extraction**

[1] Source < http://www.omsar.gov.lb/ICTGPG/Web/Activity_24_Installation_Qualification_%28IQ%29.htm >

(from a zip file) to a disk location; **configuration of the system** for first usage of the software; and **documentation** of the state of the software.

Common Installation (Qualification) Errors -- Categories:

1) **Extraction**: Full application package provided in a compressed, or zipped: file, the extraction process was incomplete and some files are missing from the application area;
2) **Extraction**: All files were extracted but, all (some) the extracted files were placed in incorrect location(s);
3) **Extraction**: All files were extracted and were placed in the correct location but, some of the files were corrupted or of a different size than expected;
4) **Configuration**: Critical directories were not properly set up and/or the registered locations do not match the system's designated location(s);
5) **Configuration**: Application configuration activities were not properly performed and some critical tasks were completed incorrectly or omitted;
6) **Documentation**: PASS(ed) status indicated in the worksheet but the attached documentation does not match;
7) **Documentation**: Step Failure/Errors not recorded yet, occurred.

Since these are the typical errors encountered during Installation (Qualifications), an automatable process can be designed around these needs; guided by your experiences this list can be modified and then, the new list then becomes the new process specifications. The IQ checking software then must meet these (possibly modified) requirements:

The IQ checking software will:

1) Verify that all files specified in the application's manifest, this manifest must be provided electronically and best practice requires that it be automatically produced by the application deployment routines;
2) Automatically configure, verify and confirm the configuration of the application during first use routines; and
3) Provide documented, printable evidence that each of the enumerated errors has been checked and NO errors were found in the: extraction and configuration of the installed components.

The system has the following steps:

Pre-release:
1) Stage the software in deployment area;
2) Build a system manifest, store in deployment area;
3) Bundle up the software into the deployment package;

At the user's site:
4) Extract the files into the client's application area;
5) Perform a check against the system's manifest, confirming all files have been found in the expected directory;
6) Perform a check against the system's manifest, confirming all file sizes match the manifest;
7) Execute the coded system configuration routines;
8) Start the application and confirm system startup and/or system startup's failure; and
9) Create documentation:
    a. If system failure, create error file describing: missing components and required actions; OR
    b. If installation completed and check successfully, provide electronic documentation of each component's status.

Of special note is Step # 2, that at the time the software is deployed and prepared for packaging, an electronic manifest must be prepared listing the directories and files in the deployment area; the code for this is relatively simple and a sample is shown below:

```
*=========================================================================*;
* Create a manifest of the current data files, use as part of a CHECK SUM *;
*=========================================================================*;
  filename myDirX PIPE 'dir /s /b -A -D "MY_DEPLOYMENT_AREA" ' ;
  data manifest.SysManifest ;
    attrib  AppData length=$80   label = "List of APPLICATION'S Baseline Files" ;
    length dirContents $ 600 ;
    infile myDirX lrecl=600 truncover end=eof;
    input dirContents $1-600 ;
    AppData=upcase(substr(dirContents,index(upcase(dirContents),'KEY_STRING')+10));
  run ;
  filename myDirX clear ;
```

Also, for automation purposes, the configuration routines use SAS' GETOPTIONS functions to identify, copy and automatically configure the SAS installation. This is accomplished by:

1) Executing SAS in batch mode;
2) Identifying the concatenated path for the CONFIG file(s), sasV9.cfg;
3) Extracting the configuration file associated with the currently identified LOCALE; and
4) Copying the configuration file to the application's home, modifying it to contain the appropriate configuration information.

   Any number of SAS configuration options can be programmatically controlled including: MVARSIZE, SASUSER, WORK and SORTSIZE. A partial example of the resultant file's contents is shown below.

```
/*-------------------------------------------------------------\
|   Environmental Controls  (SAS INITIALIZATION)               |
\-------------------------------------------------------------*/
-SET SASINITIALFOLDER 'C:\HOME_DIRECTORY\SASPGM'
-SET APPHOME          'C:\HOME_DIRECTORY'
-AUTOEXEC             'C:\HOME_DIRECTORY\SASPGM\AUTOEXEC.SAS'
-DATAHOME             'C:\DATA_HOME'
```

Other benefits of using the SAS configuration file rather than the SAS Startup icon includes the ability to define your own environmental variables like DATAHOME, this read-only system variable can have a like-named macro variable that can be reset at will and then set back to the base value as follows:

```
%let DATAHOME = %SYSGET(DATAHOME) ;
```

Finally, using code similar to the code in the manifest build routine above, the IQ routines run; comparing the current installation against the system manifest, discrepancies are noted in the application's SASINITIALFOLDER identified above.

OPERATIONAL QUALIFICATIONS

DISCUSSION

The OQ provides "The documented evidence that the system or equipment performs as intended throughout all anticipated operating ranges. The Operational Qualification protocol contains the procedures to verify specific dynamic attributes of a system or equipment throughout its operating range, which may include worst-case conditions."[2] Since the software must be verified 'throughout its operating range'[3], OQ testing can and does become tedious and it is not uncommon for an OQ script to run into the 100+ pages; as a consequence the OQ becomes prone to errors in the test cases, as well as the documentation, of the test routines. Consequently, the automation of the OQ processes can usually result in the greatest savings for the users of your software.

---

2 Source < http://www.novasep.com/misc/glossary.asp?defId=247&lookfor=&search=O >

3 See Carnegie Mellon's Software Engineering Institute's standard, SEI-CM-12-1-1, data flow coverage requirements for a more in-depth discussion of the requirements for software testing across the full operational range of the system.

Also, it is important to decide what automation of the Operational Qualification is, and is not. Automation of the OQ still requires the development or production of everything one would normally expect to be done for any OQ: SOPs still need to be in place; training of personnel must take place; test plans and a traceability matrices need to be developed… critical system or data tests must be identified, thorough testing planned and test 'scripts' developed, etc. So, automation is *not* the elimination of these tried and true practices used in the software world today; those practices are still part and parcel of what needs to be done to properly qualify a package, operationally.

What automation is then, is changing the '***how it is performed***' rather than changing the '***that it is performed***' characteristics of some of the qualification tasks. Manually executed script steps that *can* be automated, *should* be automated; manually executed steps that can*not* be automated, should NOT be automated. The crux of the problem then is; what steps can, or should, be automated when preparing an OQ?

Now, let's go back to the savings mentioned above, these savings can be magnified as an application is ported to multiple operating systems or SAS versions; SAS has identified and implemented similar functionality when they began to offer expanded of their products and/or began supporting multiple platforms on compressed timeframes.

It seems obvious that, once the automation modifications have taken place, it takes only incrementally more time to run an OQ on multiple SAS versions or operating systems compared to the multiples of the time required to run the same manual OQ script for each SAS version or operating system; odds are, if one hundred pages of scripts must be run for platform release #1 then each additional release will require approximately the same number of pages of scripts.

Without a doubt, qualifying analytical software at an operational level presents definite challenges, not the least being that there can be many test cases that are correct but, due to data dependencies, a few test cases will break the system and result in failures.

Another important factor is that, typically, many software applications' specifications are user-centric, i.e. rather than specifying all designed functionality, the specifications address the functionality that the *user* will see. In the case of the requirements of the system's design to support automation of the OQ, typically these 'derived' requirements will not be at the top of the user's list of their priorities. Therefore, OQ automation requirements, i.e. those derived from the ability to automate a test or testing protocol, tend to not be specified when the software's requirements are being formulated. If they are specified, they are usually formulated late in the software development process; a point at which the programming may require substantial modifications to support the new automation paradigm. This late-phase requirements definition raises the development costs and tends to discourage implementation of much of this functionality.

However, the ability to automate this process can have a major impact on the usability of, and desire to use, an application since the user experience is enhanced with the improved performance capabilities of the system.

Interestingly, experience shows that the new, automatable system architecture and code increases: testability, extensibility and maintainability characteristics with the system. In hindsight, this makes sense in that the changes significantly improve the modular structure of the code, thereby forcing the code towards these reusable structures found in modern development structures.

PROCESS DESIGN SPECIFICATIONS: OPERATIONAL QUALIFICATIONS

That said, what *are* some of the major requirements for software; requirements that are solely derived from the automation of testing or Operational Qualification process? Based on successful automation efforts, the following are suggested requirements for the OQ automation sub-system:

1) Environmental controls, embedded system resets to 'base' configuration with automatic system 'resets' executed after each run iteration;
2) If reports are produced:
   a. Separate data production from data rendering (so that the data can be electronically checked by the system);
   b. Maintain separate, manual reviews of the formatting of the reports but, use the error trapping system to detect that no error occurred in the rendering process;
3) If the system employs a user interface to perform its functions, pass interactive run requests into batch-able routines;
4) Eliminate dependency on error logs to determine the error status of an executed routine;
5) Surface error status codes to the application's shell, i.e. if an error occurs during operational runs test case, the system must pass out distinct error codes; and
6) Once an error occurs, the application must be capable of extracting and permanently capturing the error code before exiting the application's step, i.e. no ENDSAS-type commands or uncontrolled abends.

Other issues to consider when writing OQ test scripts and then automating these tests are the common OQ errors encountered in other Operational Qualification efforts. The following section lists some of these but, as with any checklist, the list is dynamic and should be added or subtracted from as experience dictates.

TECHNICAL SPECIFICATION CONSIDERATIONS

The technical specifications for any application controls the development of the application at a level appropriate for programming or coding, as such, it is appropriate to review considerations appropriate for guiding the programming activity; some of these are a high level descriptions of what must be performed during the scripting steps and others the common operational qualifications errors found in the review of multiple OQ processes and documents.

The script writing and execution process follows a narrow range of actions and any automated system must do or produce the same results. The following list must be included in the OQ automation requirements, all test runs must:

1) Perform full setups within the system, including the test data – setups must be specific and repeatable;
2) Capture all resultant: output, status and the relevant log information and 'package' these results into an auditable format;
3) Compare results from each run to expected value(s) with a clear decision rule controlling the PASS/FAIL criteria;
4) Determine if the test is a PASS or FAIL; and
5) Capture, reference and 'attach' documentation of the above to the script's step; include screen shots, output, logs, etc.

Since the OQ is designed to produce 'The documented evidence that the system or equipment performs as intended throughout all anticipated operating ranges', it should be assumed that the software is tested for both what it is designed to do but also to prevent errors from impacting its operation. Therefore, it behooves us to look at and avoid the common errors found during operational use, in a general sense. Since the following statements are framed in the negative, e.g. 'corrupted input files', the statements must be incorporated into the specifications using different language such as 'Corrupted input files must be not cause critical user errors', that said, a list of some of the common operational errors follows.

Common Operational Errors:

1) The system really does not work as designed[4];
2) Analysis routines yield erroneous results;
3) Un-tested or incorrectly tested design functionality;
4) Corrupted input files;
5) Input data incorrectly formatted or structured;
6) Data dependent errors, i.e. errors generated from data in with values/ranges that cause failures;
7) 'Lost' configuration/environmental variables, i.e. macro variables 'here now, gone later';
8) Leftover or retained values from previous runs;
9) Configuration files have errors (User accounts, control data, missing system directories, system data pointing to the wrong locations, etc.);
10) Error trapping routines improperly reflect a failure within the system;
11) Environmental factors: Operating System not validated for the application, SAS versions and Hot Fix levels, etc.;
12) Documentation errors, recorded status incorrect or supporting documentation inconsistent with recorded status;
13) Messaging errors: failures occurred but are not caught by user.

SUMMARY

Manual OQ scripts can be tedious and run into the hundreds of steps, many of them a repeat of a previous test but, with different input data and/or test conditions; automating these tests make sense in many ways. Also, many of the same checks performed during unit and integration testing are duplicated during the operational qualification process, therefore the time expended planning for the automation of the operational qualifications can be leveraged throughout the software development process.

This fact alone speaks for a commitment for the use of automated OQ processes, since automation increases in speed, extensibility and increased product quality are the icing on the cake that is valuable, in their own right.

Also, compared to the requirements for most applications the additional effort necessary to generate OQ specific requirements with the added functionality more than pays for itself during the full life cycle of a software application.

---

[4] These issues should be caught before release of the system for validation.

The author can be contacted at:

Joe Perry
Perry & Associates Consulting
342 Spring Canyon Way
Oceanside, CA 92057
Phone: 760-822-7959
email: PerryAJoE@cox.net