Paper 372-2012

# Customising SAS® OQ to Provide Business Specific Testing of SAS Installations and Updates

Steve Huggins, Amadeus Software Limited, Oxford, UK

## ABSTRACT

The SAS Installation Qualification and Operational Qualification tools provide a generic inbuilt method of validating a SAS installation. This paper describes a simple method that allows the Operational Qualification tool to be customized to execute additional SAS code in order to provide a robust method for regression-testing a new SAS installation using business-specific SAS programs. The process involves selecting one or more indicative SAS programs, benchmarking the expected results, and then integrating the SAS programs into the Operational Qualification tool. This paper is aimed at SAS users and administrators wishing to upgrade or update their SAS installation whilst comprehensively ensuring the consistency of their results.

## INTRODUCTION

This paper discusses the use of the inbuilt software qualification tool provided as a part of the standard SAS foundation install. A general overview is provided of both the Installation and Operational Qualification tools however the main focus of this paper is the Operational Qualification tool. The steps required to allow SAS users to integrate their own SAS programs into the Operational Qualification tool are described in detail and consideration is given to how the Operational Qualification tool can be deployed to provide customised software validation across an organisation. Examples given throughout the paper are based on a Windows installation of SAS however the tools described are available on other operating systems and the basic concepts apply equally. For details of how to make use of these tools on all platforms refer to the SAS IQ/OQ Documentation [ref1].

## OVERVIEW OF SAS IQ/OQ

Any installation of SAS Foundation software will automatically include two software qualification tools; the Installation Qualification tool (IQ) and the Operational Qualification tool (OQ). These two tools should be used after installing, upgrading or updating SAS Foundation software in order to ensure the installation has been successful.

### SAS INSTALATION QUALIFICATION TOOL

The Installation Qualification (IQ) tool is used to verify that all of the executable components deployed as part of the SAS installation have been correctly installed. To check this, the IQ tool uses the MD5 hashing algorithm to generate a hash value for each executable file; these are then compared with the corresponding reference values for each file to confirm that the file has been installed correctly. The IQ only checks components installed as part of SAS Foundation and so does not include other application components installed as part of the BI platform.

The IQ can be executed from the start menu in Windows or using a command line statement, in both cases a location for the output report will need to be provided. The IQ process will test all of the installed components and provide a detailed report in the location specified. The report is delivered in the form of a drillable HTML file providing summary and detailed information as well as a complete copy of this information in a PDF file.

### SAS OPERATIONAL QUALIFICATION TOOL

The IQ tool will validate the executable components that have been installed however the Operational Qualification tool (OQ) provides further confidence by using the installed SAS software to run a number of predefined SAS programs. The OQ checks that each test program executes without error and in some cases compares the generated results with predefined benchmark values.

The OQ can be executed via the command line or using the short cut under the SAS utilities folder from the Windows start menu. Running from the command line has the advantage that you can specify exactly which test programs should be executed if a limited list is required. The results are delivered in the form of a drillable HTML report and PDF document providing summary and detailed information regarding each SAS component that is installed.

By using both the IQ and OQ it is possible to verify that the installation of SAS Foundation is not only correctly deployed but also functions as it should. It is also possible to add user written test programs to the OQ process to be

used in addition to the default set of tests, this allows users to validate the installation using the exact processes that will be used within their business providing further assurance of the reliability of their SAS installation.

## CUSTOMISING THE SAS OPERATIONAL QUALIFICATION TOOL

The following sections will provide some general background into the way that the SAS OQ works and will describe the steps required to add custom test programs to the OQ process.

### ANATOMY OF THE SAS OQ TOOL

The various components that make up the SAS OQ tool are all located in a folder named “sastest”, this can be found within the SAS root location, i.e. the same location as the main SAS executable (the components for the SAS IQ are also located here).

The OQ tool itself consists of an executable file (sasoq.exe) that controls the overall process and a number of sub-folders each of which contains the SAS programs that make up the OQ tests. Each sub-folder contains the test programs for a specific SAS module such as Base, STAT, etc. Also included within each of the folders is a table file (.TAB). The table file contains the instructions for each individual test that is to be run as part of the OQ process; this information is recorded using a specific notation known as the “Operational Qualification Tool Table Language” for a detailed description of this syntax refer to the SAS IQ/OQ Documentation [ref1].

When executed, the OQ reads the table file in each folder which provides a definition of each of the tests that are to be run. Typically each test will be linked to a specific SAS program; SAS programs that are included in a folder but not referred to in a table file will not be executed. The OQ runs each of the tests defined within the table file, on completion each test program will pass a return code back to the OQ process and the value of this code determines whether that test has passed or failed. The default return code for a successfully executed SAS program is zero; errors or warnings generated during the execution of the SAS program will cause different return codes to be returned. The OQ records all of these results and once all of the tests from each folder have been run a SAS program is executed to render the results into the standard format.

The command line statement (Windows) to launch the OQ is as follows:

```
sasoq.exe -tables *:folder1 *:folder2 -outdir C:\Output_Location
```

This must be submitted from the “sastest” folder or a full file path must be included, the options are as follows:

| Option | Description |
|---|---|
| -tables | Instructs the OQ to run all of the tests defined within a subfolder with that name.<br>If a value of “*:*” is entered all subfolders are processed (this is the command submitted via the Windows start menu). |
| -outdir | Specifies the location in which to generate the output reports (if not included this will be prompted for). |
| -testware | This is an additional option that allows a root folder to be defined (the default is the root location of the OQ executable). |

**Table 1. List of Basic OQ command Line Options**

To view the results of the test process open the file named “sasoqtoc.htm” in the output location. This will provide various links to the summaries and detailed reports generated. The same information can also be found in sasoq.pdf within the same location.

### ADDING CUSTOM TESTS TO THE SAS OQ

The Operational Qualification tool allows additional user written tests to be included as part of the standard suite of functional tests. These can be incorporated into the OQ test process and results are delivered as part of the standard report. There are two elements that are required to add new test programs to the OQ run. These are a test program and a test definition (test block) to be included within a table file.

It is possible to add a test program to an existing folder and simply edit the appropriate table file to include that test however the approach described in this paper will involve adding a new test folder with a new table file. The advantage of this approach is that the original folder structure remains in its default state and all additional testing is managed within its own folder. The report process automatically groups the tests by source folder making it clear which tests are included in each group so that the newly added tests can be easily identified.

The sections below describe how to set up a new table file so that a simple SAS program can be added to the OQ process. These steps will assume that a new sub-folder has been added to the sastest folder.

## SETTING UP THE TABLE FILE

The OQ table file serves to define the various tests that are to be executed, the command line that launches the OQ only provides a list of folders to process, and the OQ will look in each of the folders for a table file and execute any tests that are defined within that table file.

A table file contains instructions written in the OQ table language. These statements allow for the definition of various options and variables however the majority of statements are used to define the various tests to be run. This paper describes those statements required to add custom tests into the OQ process however further details can be found in the SAS IQ/OQ Documentation [ref1].

For the purposes of this paper the following statements will be discussed:

| Statement | Description of use |
|---|---|
| &set | This statement allows the creation of variables which can then be used anywhere within the table file. The variable name must consist of alphanumeric values and any parenthesis within the value must be matched. To resolve the value of the variable use "@varname" where varname is the name allocated by the &set. |
| &test | The &test statement is used to define a test block, each test block will define an individual test that will be executed and reported by the OQ process (see example below). |
| &run | This statement is used within a test block and instructs the OQ to execute an application (usually SAS). The &rc() statement must be followed by the &rc() statement .<br>For example: `&run(sas: (-sysin C:\test.sas -autoexec c:\sasautoexec.sas))`<br>would run the SAS program "test" (specified using –sysin) using the "autoexec.sas" file as the autoexec. Other SAS options can also be defined in this way. |
| &infile | This is used to reference a physical file (usually a SAS program), it has three arguments separated by colons in the form: `&infile(suffix:filename:subfolder)`.<br>For example, `&infile (sas:test1:base)` would identify file "test1.sas" in the sub-folder "base". |
| &rc | This statement is used to test the return code value that is returned by the program executed by the preceding &run statement. The single argument is the expected return code. This would be zero for a successfully executed SAS program but can, in principle, be any numeric value. |
| /* text */ | This is a comment block; text within the block will be ignored. |

**Table 2. List of Commonly Used Table File Statements**

Below is an extract from the table file used in the default tests for Base SAS (only one test block is included):

```
&set ( sasopts : -autoexec &infile(sas:assert:base) -ls 78 -ps 60 -noovp
    -nosyntaxcheck -nodate -nostimer )

&test  tstsql
 {
  &run ( sas : ( -sysin &infile( sas : tstsql : base) @sasopts))
  &rc(0)
 }
```

Note the following points about the way the table language statements are utilised:

1. The &set is used to create a variable called "sasopts". This will be used as a shortcut to define various SAS options to be applied within the &run statements below. The options defined will be set within the SAS session invoked by the test definition. Note that an autoexec option is defined here that will call the program assert.sas which resides in the base folder. This defines a SAS macro called %rccheck which is important for the identification of test failures (details on this later).
2. The &test statement first defines the name of the test as "tstsql". This is the name that will be listed in the final report. All of the arguments to the &test statement are then enclosed within braces.

3. The &run statement is used to execute SAS, the –sysin option specifies the SAS program that SAS will execute, in this case the values are provided by the &infile statement. The SAS file will be resolved to "sastest\base\tstsql.sas".
4. The table file variable "sasopts" is then used to add the SAS options (including the autoexec call) that are to be used within the SAS session. These include –ls, –ps etc. as defined in the &set statement above.
5. Finally the &rc statement is used to test the value of the return code provided by SAS on completion. In this case the SAS program simply executes a Proc SQL step. If this runs without error the return code from SAS will be zero. The &rc statement then has the value of zero as its argument (this is the expected value). If a return code of zero is returned by SAS the &rc statement will record a pass for this test block, any other value will record a fail.

The simplest way to approach the creation of a new table file is to start with the table file used in the default base test folder and modify it to meet the needs of the new custom test(s). It is recommended that the sasopts variable is used and assert.sas program is used as an autoexec. It is then a simple matter of editing the test blocks to include the new tests (edit one test block and then use it as a template). The test name, the file name and folder referred to in the &infile statement as well as the expected value in the &rc statement will need to be updated. The new tests will then be executed and the results reported as part of the standard report. The tests will be grouped under a common heading derived from the name of the folder where the table file was located and each test result will be individually reported under the name defined in the test block.

## SETTING UP THE TEST PROGRAMS

At a simple level all that is required in order to include a test program is to ensure that the SAS program is available to be executed by the OQ. In general, test programs would be stored in a folder with the table file that calls them. There is however great scope in the way that the test programs can be designed, test programs can be a simple data step or a complex process involving multiple data and proc steps. The key consideration to keep in mind is how the test will interact with the OQ process. This is always via the return code that the SAS session returns to the OQ on completion and there are a number of ways in which this can be controlled.

As mentioned above when a SAS program completes without error it passes a return code of zero back to the OQ, this is then tested using the &rc(0) statement within the test block that evokes the SAS program. When a test program generates an error, a different return code will be passed to the OQ and &rc will register a fail for that test. If however the test program is looking to generate specific values from a process such as Proc Summary then a return code of zero would be returned even if the results are wrong as no errors are generated. To manage this scenario the OQ provides a useful macro called %rccheck, this allows a value to be tested within the test program itself and if the value is not correct then the return code sent to the OQ can be changed to register the failure.

**Using the %RCCHECK Macro within test programs**

The %rccheck macro tests the value which is passed to it as a parameter (this is usually a return code from a previous step) and if the comparison is not as expected then the SAS session is aborted and an appropriate return code (not zero) is sent back to the host operating system. The syntax for %rccheck is:

```
%rccheck(actual, expected, operator);
```

The arguments are used as follows:

| Argument | Description of use |
|---|---|
| Actual | The return code value to be inspected, usually a return code or value passed from the preceding step, such as &sysinfo. |
| Expected | The expected value, the value that the return code in the actual argument will be compared with. This has a default value of zero. |
| Operator | The comparison operator to be used, this defaults to NE (i.e. if actual value is not equal to the expected a fail is registered). |

**Table 3. Arguments for the %RCCHECK macro**

Below is an example of a test program that uses the %rccheck macro:

```
* Create test data;
data test_data;
  input a b c;
cards;
1 2 3
run;

* Test functions and put the results into macro variables;
data _null_;
  set test_data;
  test1 = sum(a,b,c);              * Test Sum function;
  call symputx('sum_test',test1);  * Copy result to a macro variable;
  test2 = mean(a,b,c);             * Test Mean function;
  call symputx('mean_test',test2); * Copy result to a macro variable;
run;

%rccheck(&sum_test,6);  * Check the result from the Sum function;
%rccheck(&mean_test,3); * Check the result from the Mean function;
```

In the example data step above two data step functions are being tested. The results they generate must be passed into macro variables so that they can be validated using the %rccheck macro. In the first case the sum value will match the expected result (6), the %rccheck macro will then take no action and if this were the only test then the SAS session would end normally with a return code of zero. However the mean value tested by the second %rccheck will in fact be 2, since %rccheck is expecting this value to be 3 the macro will abort the SAS session. Internally this is done using the abort statement which will terminate SAS and pass a non-zero return code back to the OQ in order to register a failure, (i.e. the return code will be validated by the &rc table file statement ).

**Using the %RCCHECK Macro with PROC COMPARE**

A second application of the %rccheck macro is to validate the return code generated by a SAS procedure, in practical terms this is most usefully applied to the &sysinfo automatic macro variable generated by PROC Compare. When using the compare procedure to compare two SAS datasets a return code is stored in the &sysinfo macro variable. The value of &sysinfo depends on the results found by PROC Compare, the value of most interest here is the zero which is returned when no differences are found between the two datasets. The example below demonstrates how PROC Compare can be applied within the OQ:

```
* Define raw data;
data test1;
  input a b c;
cards;
2 12 23
5 34 23
4 23 23
1 10 23
run;

* Run test process;
proc means data = test1 mean std;
  output out = test2 (drop= _type_ _freq_) mean= std= / autoname;
run;

* Define benchmark data;
data bench;
  input  a_Mean b_Mean c_Mean a_StdDev b_StdDev c_StdDev;
cards;
3   19.75   23   1.8257418584    11.086778913      0
run;

* Compare results with the benchmark data;
proc compare base=bench compare=test2 noprint
             method=absolute
             criterion=0.000000001;
run;

* Check the return code;
%rccheck(&sysinfo,0)
```

Here the means procedure is used to generate some basic statistics from known test data, the results are written to a dataset so that they can be examined later. The input data is rigidly defined therefore the expected output is also known and is (in this case) hard coded in the test program. PROC Compare is then used to compare the actual results generated with those that are expected from the given input data. If the datasets are found to be identical then &sysinfo will have a value of zero and this is then tested using the %rccheck macro, again a non-zero value will cause the SAS session to abend indicating a failed test.

In situations where the bench mark values and the tested values are run on different platforms (such as when upgrading to new hardware) it is not unusual to find small differences in the generated values. As this does not represent an issue with the SAS software itself it is good practice to use the absolute method for comparing the values, add the criterion option to provide an acceptable margin of error to account for these differences. What is considered to be an acceptable level of error will depend on the individual business.

## USING SHARED TEST RESOURCES

In the examples above all of the resources required to run the test programs have been included within the programs themselves, and these in turn have been deployed to the machine that is being tested. This approach is fine for validating a single machine but is less appropriate when all SAS installations across an entire business need to be validated using the same criteria.

One good way to resolve this issue is to store the test programs and bench marking data in a central repository. This can then be situated on a shared network drive so that all users have access to the testing resources. Read only folders should be created to hold standard test programs, standard input data and standard results data for benchmarking. With all of the custom tests stored centrally the only components required to be stored on the SAS test machines would be a custom folder, a table file and some test programs. The deployed test programs would simply call the standard test programs from the central location (the -testware option could also be used for this). The programs would then be executed on the test machine using the standard data and the results validated against the standard benchmark data. Using this approach all testing can be standardised across all SAS machines.

These ideas can be taken even further by using test programs that write their results to a date stamped folder. This could be a stored in a centralised location so that an audit of the status of all SAS installations across the business can be easily maintained. Finally it is relatively simple to schedule the OQ process to run at regular intervals to ensure continuity over time.

## USING THE OQ TO VALIDATE BUSINESS SPECIFIC SAS PROGRAMS

Below are some general guidelines on the steps to take in order to add custom test programs to the SAS Operational Qualification Tool:

**1) Select the Test Programs:**

Select the business critical SAS programs that are to be used to validate the SAS installation(s). Care should be taken to ensure that all of the different types of SAS process are covered. Consideration should also be given to how the results will be validated within the OQ process.

**2) Create the infrastructure:**

Set up the folders on all test machines. Where there are to be many test machines a folder could be set up with all of the required files and simply copied into the sastest folder on each machine. If centrally stored resources are to be used, set up these folders and ensure the access rights are correctly configured.

**3) Capture Input and Benchmark Data:**

If specific input data is to be used for the tests, store these tables in a stable read-only location so that the test programs will be able to read it. Where the results from the tests are in the form of datasets that will be checked using PROC Compare, run the test processes and store the output datasets in a permanent library. Ultimately ensure that this location is also read only.

**4) Configure the Table File:**

Set up the table files that will be used to control the different tests, use the table file from the base folder as a start point and ensure that the autoexec option is configured to run “assert.sas” from the base folder, this will ensure that the %rccheck macro is defined for each SAS test session.

**5) Deploy Test Programs:**

Copy the test programs to the new test folder, this may be a case of simply copying the test programs to the new folder on the test machine, or may need the test programs to be copied to a central location and separate SAS programs added to the new folder on each machine to call the centrally stored programs.

**6) Edit the test programs:**

Finally make any edits to the test program to ensure that the test results are properly captured, this may include the addition of PROC Compare steps and use of the %rccheck macro.

## CONCLUSION

The SAS Operational Qualification Tool serves to validate a SAS installation in its “out of the box” state however it can be enhanced to include many more tests as specifically required by a business. In the sections above it has been seen:

- How the OQ works to execute multiple functional tests in the form of suites of SAS programs;
- The configuration of the table files and how this allows for the simple organisation of many test programs;
- How any SAS program can be used as a functional test and how by using the compare procedure and the %rccheck macro, real processes and results from the business can be employed as part of the validation process;
- How, with a little preparation the whole qualification process can be centrally managed.

The OQ provides a set of tools for executing and then validating any number of custom SAS programs, incorporating the results into the standard set of reports. With these tools and a little imagination a comprehensive and varied set of additional business specific tests can be included as part of the OQ test process providing increased confidence when updating or upgrading SAS Foundation software.

## REFERENCES


| | |
|---|---|
| Ref1: SAS IQ/OQ Documentation: | SAS® 9.1.3 Qualification Tools. User's Guide<br>SAS® 9.2 Qualification Tools. User's Guide<br>SAS® 9.3 Qualification Tools. User's Guide |


## CONTACT INFORMATION

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

Steve Huggins
Amadeus Software Limited
Mulberry House
9 Church Green
Witney.
OX28 4AZ
UK.

Work Phone: +44(0) 1993848010
Email: steve.huggins@amadeus.co.uk
Web: www.amadeus.co.uk