PhUSE 2013

Paper PP01

SASUnit: General overview and recent developments

Patrick René Warnat, HMS Analytical Software GmbH, Heidelberg, Germany
Peter Bewerunge, HMS Analytical Software GmbH, Heidelberg, Germany

ABSTRACT

SASUnit is a unit testing framework for the development, execution and documentation of tests for SAS programs. It consists of SAS macros and a few shell commands and is available for free from HMS Analytical Software GmbH. With SASUnit, tests are implemented as executable source code. These tests can be repeated with very little effort, therefore negative side effects of software changes are spotted easily. Moreover, SASUnit supports the assessment of code coverage. This new feature can be used to find parts of a SAS program that have not been executed during tests.

This paper provides an introduction to SASUnit as well as an update on recent improvements. First, the general concept of SASUnit and its structure is explained. Then, the usage of SASUnit is outlined and demonstrated with a simple example. In addition, the latest features are presented, with a special focus on the assessment of code coverage.

INTRODUCTION

SAS macros are used in order to simplify the reusability of SAS source code. An important part of the development of a macro is to assure that given user requirements are implemented correctly. Thus, testing of macros is inevitable, during first development as well as after every change/enhancement of macros. Using SASUnit, a unit testing framework for SAS, this process can be simplified.

SASUnit is a unit test framework for the development, execution and documentation of tests of SAS macros. Using SASUnit, tests are implemented as executable SAS code. Tests are grouped into test scenarios. Test scenarios are put together into test suites. Basically, SASUnit is a set of SAS macros, enabling the execution of test scenarios and the generation of a report for documentation of the executed tests and its results.

A SASUnit test suite comprises one or more test scenarios. A test scenario is an executable SAS program defining test cases. Every test case tests a certain aspect of the macro to be tested (unit under test). SASUnit is controlled by a SAS program (commonly called 'run_all') which represents a test suite.

The invocation of the macro %initSASUnit, used in a test suite, opens a test database (set of SAS files). The macro %runSASUnit executes each test scenario in a separate SAS session.

In every test case of a test scenario, static test data is defined, the SAS macro to be tested is called, and finally assessment macros are invoked to compare the results generated with predetermined, anticipated results. The test results are stored in the test database. As the last step in a test suite definition, a report is generated from the test results in the test database using the macro %reportSASUnit.

Figure 1 gives an overview of the structure of unit tests with SASUnit.

When creating unit tests, it makes sense to ensure that all source components were executed in at least one test. This will avoid that parts of the source code remain untested. The automated determination of source code parts executed in a given set of tests is called the determination of the test coverage. As of version 1.2.1 SASUnit under SAS 9.3 supports the determination of test coverage.

The following sections provide more information on SASUnit and a general overview of the new features in version 1.2. After that, the option of determining the test coverage is presented, which is supported in the current version 1.2.1 of SASUnit.

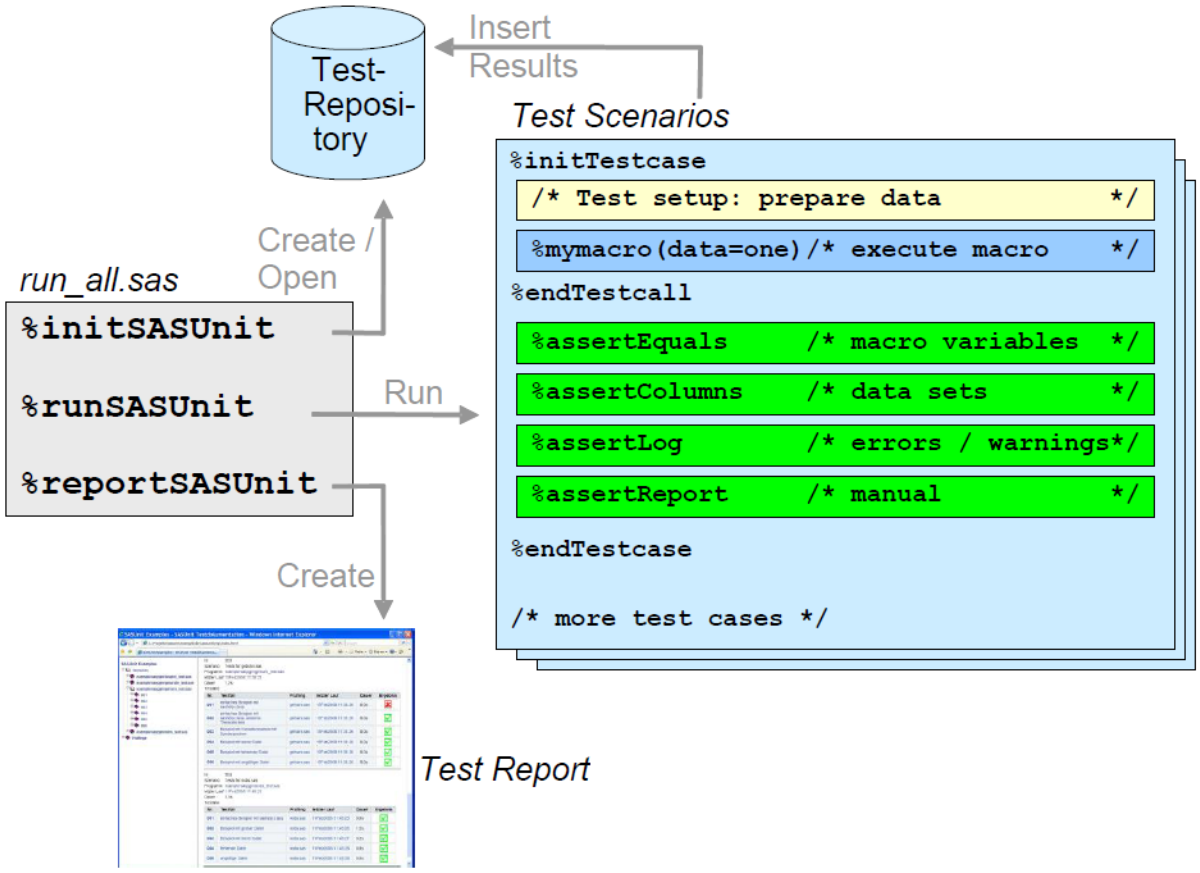

Figure 1: Structure of units tests with SASUnit. Diagram from the documentation of SASUnit [1].

SASUNIT

ADVANTAGES OF SASUNIT AND HOW TO GET IT

SASUnit is open source software and free to download from sourceforge.org [1].
The benefits of SASUnit can be summarized as follows:

- Test scenarios and tests of SAS macros are written as SAS source code
- "assertion" macros support the testing of results: values of macro variables, contents of SAS files, external files existence or emergence of certain log messages
- Test execution and documentation of test results can be run in batch mode
- The automatically generated test result documentation is consistently structured and combined with the documentation of the SAS macros
- Measurements of macro run times are performed in addition
- Non-automated tests can be included in test scenarios. In this case, a message appears in the test results documentation, e.g. that manual inspection of generated output has to be made
- Available for SAS 8.2, 9.1, 9.2 and 9.3 on Windows, Linux and Solaris

NEW FEATURES OF SASUNIT

With version 1.2, the following changes were made to SASUnit:

- Version 1.2 is the first release of SASUnit whose development was managed entirely on open-source project management platform SourceForge. New releases are now provided faster and the development team can better work together with the users.
- Support for different execution environments (SAS versions, operating systems, etc.) has been systematized. There are now two batch files per execution environment provided: one for the complete execution of a test suite and one that performs only changed scenarios.
- Projects can be moved to another location in the file system, without the need to adjust file paths.
- Batch scripts now pause, if needed, to display error messages.
- The test procedures were improved. SASUnit is tested in a special self-test environment and a detailed test plan is executed.

• The main page of the test documentation now contains more information about the execution environment (SAS version, configuration, user id, etc.).
• The SASUnit logo was changed to a green OK icon and now contains a link to SourceForge.
• There is a new assertion for performance testing (assertperformance.sas).
• The example regression_test.sas has been improved to make it more understandable and is now available on SAS for Linux.
• Test cases that redirect log files are now supported. There is the new global macro variables g_logfile and g_printfile.
• Some bug fixes were done, in particular, an error was corrected that affected the incremental test execution (to run only modified tests) in English configurations of Windows.

In the SASUnit Version 1.2.1, an option for assessment of test coverage was introduced. This is described in the following sections. Version 1.2.1 is the current release of SASUnit.

In the approaching new version 1.3 of SASUnit, which is currently in development, the following new features will be included:

- Write results of assertions to saslog
- New assertions asserttableexists, assertrowexpression, assertrecordexists, assertrecordcount, assertforeignkey
- Integrate SASUnit with continuous integration tools like Jenkins

ASSESSMENT OF CODE COVERAGE

WHAT IS IT ABOUT?

The assessment of the code coverage of unit tests is a formal approach to determine how well a code was tested. The determination of test coverage helps to evaluate the quality of test cases. There are different approaches to determine the coverage of source code by unit tests.
Examples are:
• "statement coverage": describes whether each statement / each line of code was executed during the tests performed
• "decision coverage": assesses whether all Boolean expressions used in control structures (such as% IF-statements) were resolved at least once to true and false in the set of tests performed
• "path coverage": check that all logically possible execution paths were executed during the tests

For a detailed description and discussion of the different approaches to determine test coverage, see [2].

WHY IS IT USEFUL TO ASSESS TEST COVERAGE

Using test coverage assessment, it can be determined if there are source code components that are never executed in the associated unit tests. As unit tests are one way to ensure the quality of source code, the determination of test coverage is helpful to check the quality of the test cases themselves. By determining the test coverage, it can be easily prevented that parts of source code remain untested and thus potential errors remain undetected in the source code.

IMPLEMENTATION OF TEST COVERAGE ASSESSMENT IN SASUNIT – GENERAL CONCEPT

In the current version 1.2.1 of SASUnit the determination of test coverage is implemented as a variant of the "statement coverage": the test coverage of each line of code is checked. The determination of the test coverage is switched on by setting the parameter i_testcoverage of macro initSASUnit to the parameter value = 1 (prerequisite: SAS 9.3). To implement determination of the test coverage the SAS MCOVERAGE option is used that is available in SAS from SAS version 9.3 onwards.
With the MCOVERAGE option and the associated MCOVERAGELOC option, it can be logged during a SAS session, which lines of code are executed and which lines of code are "non contributing code" (especially comment lines, but also, for example, jump target marks for %GOTO commands). Since SASUnit creates a separate SAS session per test scenario, a text file for each test scenario is created. In these text files is logged with a unique MCOVERAGE syntax which lines of code were executed and which represent "non-contributing code".
After execution of all test scenarios, all MCOVERAGE log files are analyzed in order to determine the union of the lines of code executed in all test cases per macro. This is compared with the number of lines of code that potentially represent executable code. Thus, a percentage is calculated, which represents the test coverage: number of executed lines / total number of executable lines. Figure 2 shows an example of the part of the SASUnit results report showing the test coverage.
In addition, an HTML representation of the tested source code is created, in which the individual source code lines are colored accordingly to their test coverage.

Unit under Test	Test Scenario	Test Cases	Assertions	Test Coverage [%]	Result
boxplot.sas	001	22	45	97	☐
generate.sas	002	6	9	100	☑
getvars.sas	003	6	12	100	☑
nobs.sas	004	6	12	100	☒
regression.sas	005	1	6	100	☐

Figure 2: Part of the SASUnit result report showing the test coverage of the units under test.

IMPLEMENTATION OF TEST COVERAGE ASSESSMENT IN SASUNIT – PROBLEMS WITH SAS OPTION MCOVERAGE

Unfortunately, the output generated by the MCOVERAGE option is faulty. The label of "non-contributing code" is inconsistent in some cases. Thus, for example, % LOCAL% IF,% ELSE statements that are executed are not marked as "non-contributing code" in most cases, in some cases however, these statements are incorrectly marked as "non-contributing code", although they were executed. Also, in some cases, PROC or DATA steps or parts of it are marked as "non-contributing code", although the corresponding code block is definitely executable and was executed during a test. These problems have already been reported to the SAS support and were confirmed by the SAS support as errors of the MCOVERAGE option. Unfortunately it is not yet known when this problem is solved by manufacturer SAS. Within the lines of code that are classified by MCOVERAGE as "contributing code", the classification of executed code almost always works correctly (again, errors were observed, but very rare). So, in the majority of cases the lines of code are labeled correctly, thus one can work well with the MCOVERAGE option, but the results, especially the markings of "non-contributing code", currently still should be checked for plausibility.

IMPLEMENTATION OF TEST COVERAGE ASSESSMENT IN SASUNIT – EXAMPLE OF USAGE

A very simple example macro is used to illustrate the assessment of test coverage:

```
/*Code Coverage Test Macro*/
%MACRO ccTestMacro1(binaryInput);
  %LOCAL printTxt;
  %IF &binaryInput EQ 1 %THEN %DO;
   %LET printTxt = A value equal to 1 was given.;
  %END;
  %ELSE %DO;
   %LET printTxt = A value not equal to 1 was given.;
  %END;
  %PUT &printTxt;
%MEND ccTestMacro1;
```

Let us assume there exists only one test case with the following macro call:

```
%ccTestMacro1(0);
```

In this case, a report is generated that colors the source code of the macro according to the execution of the individual lines of code (in addition line numbers have been added). The orange highlighted line (line 5) is source code not executed, comments are highlighted in gray (line 1), executed code is shown in green (remaining rows). Line 1 contains no executable code and is therefore not used to determine the test coverage. Thus, 10 lines remain, of which one is not executed in the test, the result is a test coverage of 9/10 = 90%:

```
000001  /*Code Coverage Test Macro*/
000002  %MACRO ccTestMacro1(binaryInput);
000003   %LOCAL printTxt;
000004   %IF &binaryInput EQ 1 %THEN %DO;
000005     %LET printTxt = A value equal to 1 was given.;
000006   %END;
000007   %ELSE %DO;
000008     %LET printTxt = A value not equal to 1 was given.;
000009   %END;
000010   %PUT &printTxt;
000011  %MEND ccTestMacro1;
```

If a test case with the macro call %ccTestMacro1 (1) is added to the test suite, all source code lines will be colored green (except the comment line) and there will be a test coverage of 100%.

CONCLUSION

The use of SASUnit means some extra effort in the initial development of SAS macros, but saves a lot of work in the preparation of modified macro-versions because unit tests can be repeated easily. The use of the assessment of test coverage contributes to the quality of the implemented test cases, thus also improve the actual macro code. In addition, the documentation of the test results is automatically generated, which alone makes it worthwhile to integrate SASUnit in the development process of SAS macros.

REFERENCES

[1] SASUNIT at sourceforge: http://sourceforge.net/projects/sasunit/
[2] Wikipedia: Code Coverage, http://en.wikipedia.org/wiki/Code_Coverage, as requested at 21st August 2013

CONTACT INFORMATION

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

Dr. Patrick R Warnat
HMS Analytical Software GmbH
Rohrbacher Str. 26
69115 Heidelberg
Germany
http://www.analytical-software.de/en/start/