

## Automatic Unit Testing of SAS Programs with SASUnit

Andreas Mangold, HMS Analytical Software GmbH, Heidelberg, Germany  
Dr. Patrick Warnat, HMS Analytical Software GmbH, Heidelberg, Germany

### ABSTRACT

Software testing is indispensable, but time-consuming. The concept of unit testing implies that testing should start as early as possible during software development, and that tests themselves should be implemented as executable pieces of source code. As a result, tests can be repeated anytime, and negative side effects of software changes can be spotted in a fast and easy way.

This paper presents SASUnit, a unit testing framework for SAS programs, which is available for free. SASUnit is used for the development, execution and documentation of tests for SAS programs. SASUnit test scenarios extend the specification of user requirements in a formal way and developers can leverage SASUnit for automatic and standardized documentation of test results.

In a SASUnit test scenario, one allocates static test data, invokes the program under test and then calls assertion macros in order to compare actual with expected results. The comparisons are stored and used later to prepare the documentation.

### INTRODUCTION

Unit testing is widely used in software engineering in order to assure software quality in complex systems. When we standardize and reuse SAS programs, especially SAS macros, then unit testing is an imperative requirement, but the SAS system lacks this capability.

That was why we at HMS Analytical Software developed SASUnit for use in our own projects. Our objective for eventually putting it under the GPL license was to encourage other SAS users to adopt and to improve it. SASUnit can be used to test SAS macros and SAS programs. Testing of Stored Processes and DI-jobs after code generation is also possible.

SASUnit is completely written in SAS base code with a few OS commands where necessary. Currently, SASUnit requires SAS 9.1.3 under Microsoft® Windows. We will definitely port it to UNIX and possibly to the mainframe. At the time of preparing this paper, SASUnit was only available for download on redscope.org [1], a German language SAS community on the web, but there will be an English site by the time it is presented.

This paper first gives an overview of the role of unit tests in software development. It then defines the general structure of unit tests and aligns the features of SASUnit with the concepts of unit testing frameworks. Next, usage of SASUnit is outlined and finally, a case study demonstrates a few simple examples.

### THE ROLE OF UNIT TESTS

#### DEFINITION OF UNIT TESTS

According to Wikipedia, “unit testing is a method of testing that verifies the individual units of source code are working properly. A unit is the smallest testable part of an application. ... The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy.” [2]

Often, unit tests are seen as a first stage in testing, with further stages like integration tests (test of the assembled system) or user tests (test within the business process) being needed (see section about GAMP below).

Benefits of unit tests include

- Specifications are clarified and refined in the process of test case development.
- Problems are found early in the development cycle.
- Regression tests (retesting when something has changed) can be run at any time very easily.
- Integration tests are alleviated when units are tested properly.

- Unit tests, when they show adequate test coverage, document the real functionality of an existing unit of code.

## WHEN UNIT TESTS ARE NEEDED FOR SAS PROGRAMS

For simplicity, we consider only the application of SAS programs for clinical studies here. Similar considerations apply to business intelligence applications. Two cases have to be distinguished:

- One-off SAS programs for data management, statistical evaluation and reporting. Those programs are developed (often from templates) and run once for a certain task with a defined set of data. Quality assurance has to be done by log and code reviews, comparison of results to specification, tracking of sample data records and so on. There is no need for unit testing here, because the programs are for one time usage only.
- Standardized SAS macros to be reused in different studies. Those macros must function properly in different contexts with different data and different parameters. This can best be assured with automated, executable tests.

Standardized SAS macros can be controlled by parameter values and can deliver a variety of result types, including macro variable values, SAS datasets, ODS result files or external data files. Therefore, unit tests for SAS macros should make it possible to run SAS macros with different sets of parameter values and to automatically check for correctness of the different result types.

## UNIT TESTS FOR COMPLIANT GXP SYSTEMS (GAMP)

GAMP5 [3], chapter 7.1, states the following about (unit) tests:

“The test specifications, when executed, should demonstrate that all requirements, functionality, and design have been met. ... a complex product may have Module (Unit-) Testing, Integration Testing and System Testing. Test records for each stage should be reviewed and approved and retained ...

This statement stresses the fact that unit tests are only needed for complex systems where SAS macros are standardized and reused. It also makes clear that a unit testing framework should have an adequate reporting component which makes it easy to review test results and retain information about test execution (logs etc.).

## GENERAL STRUCTURE OF UNIT TESTS

Most of the time, unit tests are driven by a unit testing framework [4] and they are structured roughly as follows.

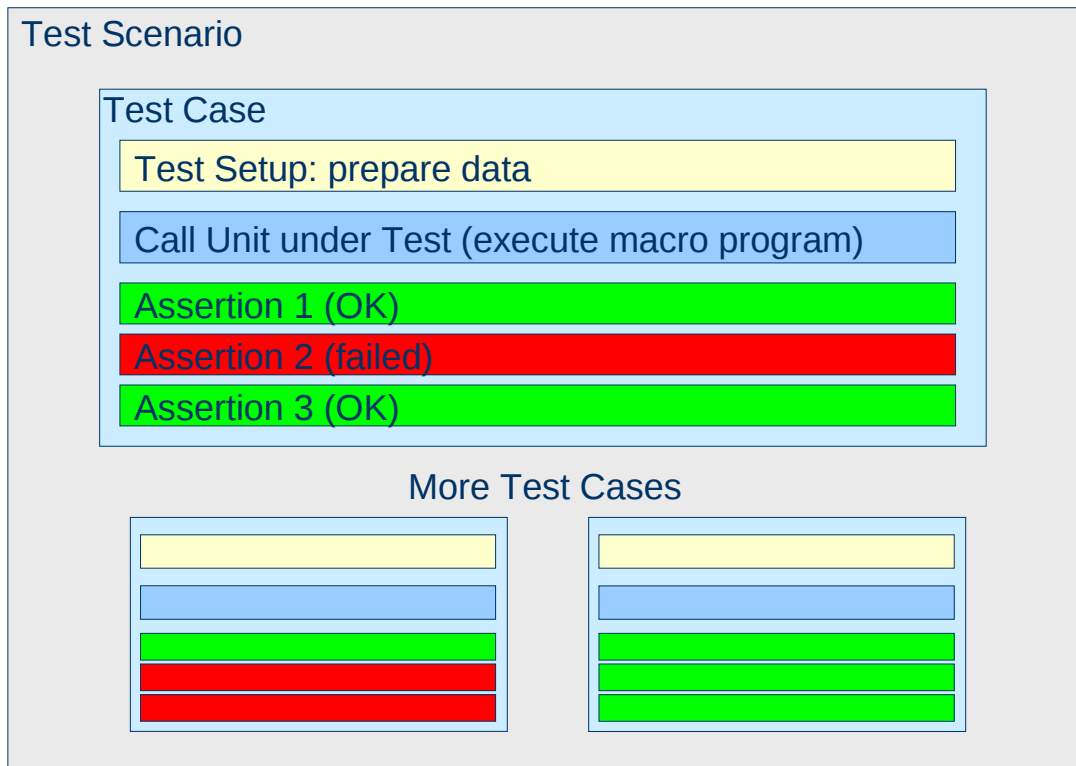


Figure 1: General structure of unit tests

A **test suite** consists of one or more test scenarios. A **test scenario** is an executable program written in the respective programming language and contains a set of test cases. Each **test case** tests a certain aspect of the functionality of a unit under test. Many test cases might be needed to test complete functionality.

A test case needs **static data** so that it can be repeated at any time. Static data can be stored in or referenced from the

testing environment, or it can be generated in the **test setup** section of a test case. Every test case contains one **execution of the unit under test** with a certain constellation of parameters and the usage of the static test data. At the end of each test case, a set of **assertions** check for the correctness of the results delivered by the execution of the unit under test. Each assertion can have the outcome “OK” (signaled by green) or “failed” (red).

## SASUNIT AS A UNIT TESTING FRAMEWORK

| Unit tests in general                                                                                                                                                                                                                                 | Unit tests with SASUnit                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unit testing frameworks control the execution of test scenarios and generate test reports.                                                                                                                                                            | SASUnit is a unit testing framework for SAS programs and SAS macros and is itself implemented as a SAS macro package. It controls the execution of test scenarios (SAS programs) and generates test documentation in HTML format.                                                                                                             |
| Test scenarios are programs in the respective programming language which use a set of functions for the control and execution of the tests.                                                                                                           | Test scenarios are SAS programs. They use a set of SAS macros for the control and execution of the tests.                                                                                                                                                                                                                                     |
| Test cases use static test data as input.                                                                                                                                                                                                             | Static data objects (SAS datasets or external files) can be managed in a special folder within the test environment. They can also be generated as part of the test scenario with SAS programming statements (data steps, PROC SQL, %LET statements etc.).                                                                                    |
| Test cases parameterize and execute the unit under test.                                                                                                                                                                                              | Units under test are SAS macros (or ordinary SAS programs) which are parameterized and executed as part of each test case.                                                                                                                                                                                                                    |
| Test cases use assertions in order to check correctness of execution results.                                                                                                                                                                         | A set of %assert-macros allows for the checking of macro variable values, dataset content, the existence of ODS reports and the presence or absence of log messages.                                                                                                                                                                          |
| A test suite executes all the test scenarios belonging to the test of a system.                                                                                                                                                                       | All necessary test scenarios of a test suite are executed by macro %runSASUnit. Every test scenario runs in its own SAS session in order to avoid side effects.                                                                                                                                                                               |
| When a new program is developed, unit tests are developed in parallel in order to test all requirements, exceptional constellations and the error recovery. The unit under test and the tests are enhanced and modified until no test fails any more. | Functionality of SAS macros can be tested by checking the values of macro variables and the content of SAS datasets.<br>For ODS results, only existence of a file can be checked, but a marker for manual checking is written to the test report.<br>Error recovery and clean logs can be checked by the built in log scanner.                |
| System units have to be changed when requirements change or when defects emerge. Regression tests assure that units that have not been changed still function correctly.                                                                              | Regression tests can be carried out at any time. Only test scenarios that have been changed since their last execution will be executed again.                                                                                                                                                                                                |
| Test reports are generated automatically after test execution. Success or failure is signaled by green or red.                                                                                                                                        | Test scenarios write their results to a test repository from which test reports can be created.<br>Test reports can be browsed in a convenient navigational HTML frame, organized by macro library, test unit, test scenario and test case. The test reports contain links to the programs, SAS logs, generated SAS datasets and ODS results. |
| Unit tests get documented in the source code.                                                                                                                                                                                                         | We use Doxygen [5] as source code documentation tool for SAS programs.                                                                                                                                                                                                                                                                        |

## USAGE OF SASUNIT

## OVERVIEW

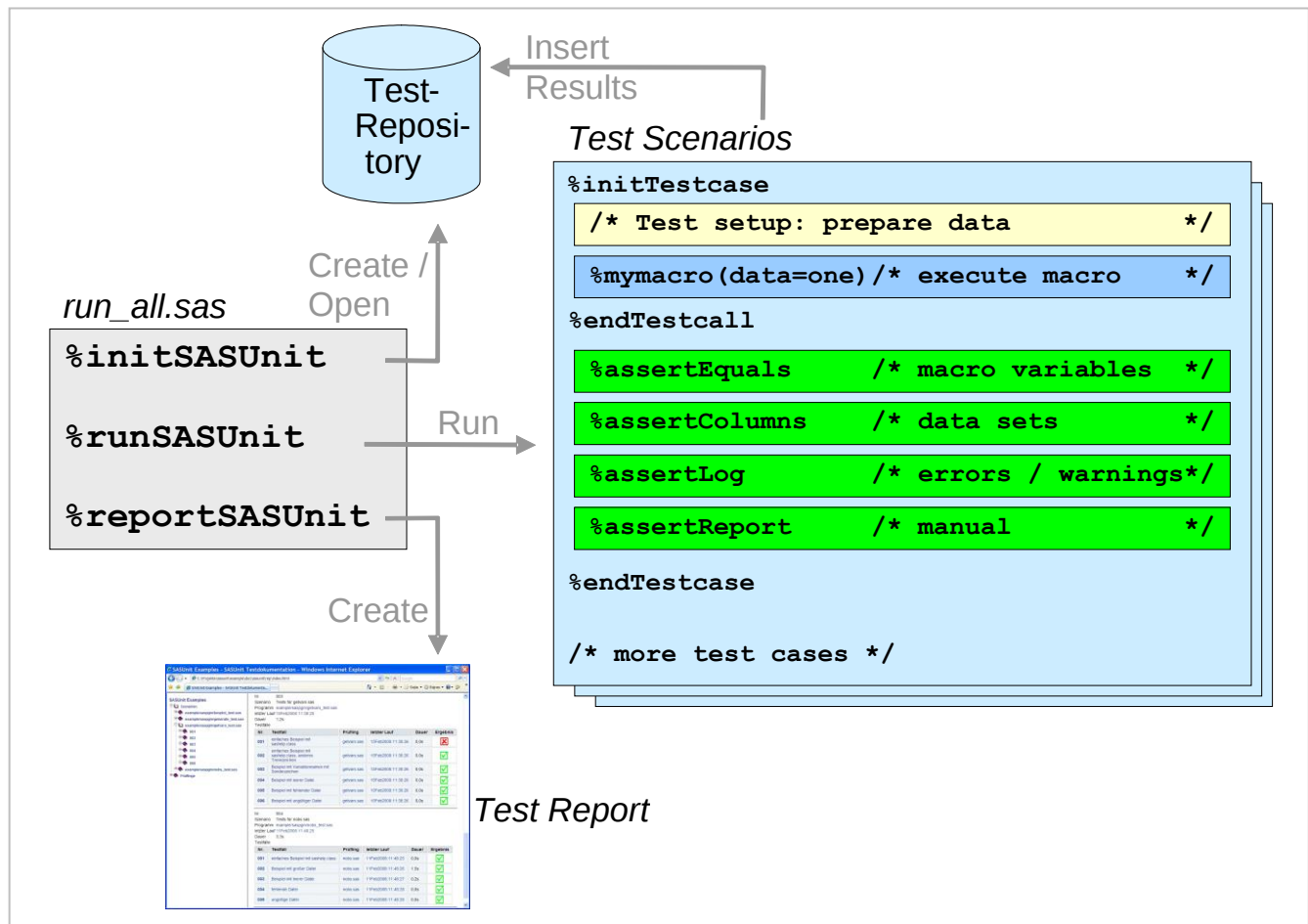


Figure 2: Overview of SASUnit usage

SASUnit is controlled by a program usually called `run_all.sas` (see Figure 2 and source code example in Figure 4).

`%initSASUnit` creates or opens the test repository. `%runSASUnit` runs each test scenario in its own SAS session.

`%reportSASUnit` creates the report from the test repository.

#### INITIALIZE SASUNIT ENVIRONMENT (`%initSASUnit`)

`%initSASUnit` opens the test repository or creates it, if it does not yet exist. The test repository consists of a set of SAS datasets with metadata for the tests, as well as SAS logs and different results from the test executions.

#### RUN TEST SCENARIOS (`%runSASUnit`)

`%runSASUnit` receives a file specification as a parameter, which can be a single test scenario or a group of scenarios. For example, the file specification `"saspgm\*_test.sas"` executes all programs with suffix `"_test.sas"` in the respective folder as test scenarios. Only test scenarios, where the scenario itself or one of the units under test has been changed since last execution, will be executed. For each scenario executed, the general result is consolidated from the test case results.

#### CREATE REPORT (`%reportSASUnit`)

`%reportSASUnit` creates the test report from the contents of the test repository. The test report is structured by test scenarios, test cases, assertions and units under test (see Figure 3). Reporting is available in German and in English.

The screenshot shows a web browser window titled "SASUnit Examples - SASUnit Test Documentation - Windows Internet Explorer". The address bar shows the URL "C:\projekte\sasunit\example\doc\sasunit\rep\index.html". The page has a navigation menu on the left with "Scenarios" and "Units under Test". The main content area is titled "Details for Test Case 004.001 | SASUnit Examples - SASUnit Test Documentation". It displays the following information:

- Scenario No. 004
- Scenario Tests for nob.sas
- Program example/saspgm/nobs\_test.sas
- Last Run 19Aug2008:14:47:25
- Duration 1,0s
- Test Case No. 001
- Test Case simple example with sashelp.class
- Unit under Test nob.sas
- Last Run 19Aug2008:14:47:25

Below this information is a table of assertions:

| No. | Assertion    | Description                             | Expected               | Actual                 | Result |
|-----|--------------|-----------------------------------------|------------------------|------------------------|--------|
| 001 | assertEquals | number of observations in sashelp.class | 19                     | 19                     | ✓      |
| 002 | assertLog    | Log prüfen                              | Errors: 0, Warnings: 0 | Errors: 0, Warnings: 0 | ✓      |

At the bottom of the page, it says "Generated on Tuesday, 19 August 2008, 14:47:31 by SASUnit Version 0.904 (32)".

Figure 3: Test report with details for a test case

```

/* open test repository or create when needed */
%initSASUnit(
  i_root      = c:\projekte\sasunit /* root path, all other paths can then be
                                     relative paths */
  ,io_target   = example\doc\sasunit /* Output of SASUnit: test repository, logs,
                                     results, reports */
  ,i_overwrite = 0                  /* set to 1 to force all test scenarios to be run,
                                     else only changed scenarios or scenarios with
                                     changed unit under test will be run*/
  ,i_project   = SASUnit Examples  /* Name of project, for report */
  ,i_sasunit   = saspgm\sasunit    /* SASUnit macro library */
  ,i_sasautos  = example\saspgm    /* Search for units under test here */
  ,i_testdata  = example\dat       /* test data, libref testdata */
  ,i_refdata   = example\dat       /* reference data, libref refdata */
)

/* Run specified test scenarios. There can be more than one call to runSASUnit */
%runSASUnit(i_source = example\saspgm\*_test.sas)

/* Create or recreate HTML pages for report where needed */
%reportSASUnit()

```

Figure 4: Example for run\_all.sas

## STRUCTURE OF A TEST CASE

Every test scenario contains a number of test cases which are structured as follows: (See Figure 5 for an example test of a simple macro which counts observations in a SAS dataset, see Figure 3 for resulting test report.)

- %initTestcase initializes the test case. The name of the unit under test (i\_object) and the description of the test are inserted into the test repository. The SAS log is routed to a separate file. A timer is started for performance checks.
- The test setup prepares test data when needed (not in Figure 5).
- The unit under test is called.
- %endTestcall (optional) ends the call of the unit under test, reroutes the SAS log and writes the execution time to the test repository.
- Calls of various %assert macros check for correct functionality and write results to the test repository (see next section).
- %endTestcase (optional) ends the test case and consolidates the detail results from the assertions to a general result for the test case.

```
%initTestcase(i_object=nobs.sas,
              i_desc=simple example with sashelp.class)
%let nob = %nobs(sashelp.class);
%endTestcall()
%assertEquals(i_actual=%nob, i_expected=19,
              i_desc=number of observations in sashelp.class)
%endTestcase()
```

Figure 5: Example for a simple test case

## ASSERTIONS

Assertions always compare an expected value with the actual value delivered by the call of the unit under test. For every assertion you will find a corresponding section in the test report, where the expected value and the actual value are compared and the result is judged for success or failure.

%assertEquals checks the value of a macro symbol.

| No. | Assertion    | Description                             | Expected | Actual | Result                                                                                |
|-----|--------------|-----------------------------------------|----------|--------|---------------------------------------------------------------------------------------|
| 001 | assertEquals | number of observations in sashelp.class | 19       | 19     |  |
| 001 | assertEquals | number of observations in sashelp.class | 20       | 19     |  |

%assertColumns checks SAS datasets or a subset of columns in SAS datasets. In the test report, links to dataset listings and to a report generated by PROC COMPARE can be found. Furthermore, differences between the two datasets with respect to various features are checked (see documentation of PROC REPORT).

| No. | Assertion     | Description              | Expected                               | Actual                               | Result                                                                                |
|-----|---------------|--------------------------|----------------------------------------|--------------------------------------|---------------------------------------------------------------------------------------|
| 002 | assertColumns | compare estimated values | Table listing<br>DSLABEL LABEL COMPVAR | Table listing<br>Comparison<br>LABEL |  |

%assertLibrary compares all the datasets or a subset of datasets in SAS libraries. The report links to directory listings of the two libraries and to a comparison report.

| No. | Assertion     | Description     | Expected | Actual                | Result                                                                                |
|-----|---------------|-----------------|----------|-----------------------|---------------------------------------------------------------------------------------|
| 001 | assertLibrary | check libraries | Library  | Library<br>Comparison |  |

%assertReport checks whether an external file (usually generated by ODS) has been created or updated in the current run of the test case. The test report contains a link to the generated file in column "Actual". Optionally, the test report can also link to a specification document or a reference version of the ODS file in column "Expected", so that the checking for correctness can be done manually. If the assertion succeeds, an empty square is shown on the test report signaling the need to check file contents manually.

## PhUSE 2008

| No. | Assertion    | Description                               | Expected | Actual | Result                   |
|-----|--------------|-------------------------------------------|----------|--------|--------------------------|
| 001 | assertReport | please compare SAS chart with Excel chart | .xls     | .rtf   | <input type="checkbox"/> |

%assertLog scans the SAS log for error and warning messages. Usually, it is used to assert that no errors and no warnings are present, but it can also assert a certain number of error or warning messages. By default, %assertLog is always applied as the last assertion in every test case.

| No. | Assertion | Description | Expected               | Actual                 | Result                              |
|-----|-----------|-------------|------------------------|------------------------|-------------------------------------|
| 005 | assertLog | Scan log    | Errors: 0, Warnings: 0 | Errors: 0, Warnings: 0 | <input checked="" type="checkbox"/> |

%assertLogMsg scans the SAS log for presence or absence of a certain message.

| No. | Assertion    | Description       | Expected                                                        | Actual        | Result                              |
|-----|--------------|-------------------|-----------------------------------------------------------------|---------------|-------------------------------------|
| 001 | assertLogMsg | Logmeldung prüfen | Message 'ERROR: boxplot: Data set XXXXX does not exist' present | Message found | <input checked="" type="checkbox"/> |

### CASE STUDY

The following case study is part of the example project, which is contained in the SASUnit distribution (see subfolder "example"). It shows how statistical results calculated by different software tools can be compared with SASUnit. A simple linear regression is calculated in Microsoft Excel® and in SAS.

#### SAS MACRO AS UNIT UNDER TEST

```

%MACRO regression(
    data    = /* input dataset                      */
    ,x      = /* variable for x axis, must be numeric */
    ,y      = /* variable for y axis, must be numeric */
    ,out    = /* output dataset, contains &x, &y, &yhat */
    ,yhat   = /* name of the variable with estimated values */
    ,parms  = /* output dataset with regression parameters */
    ,report = /* report file (file extension must be .rtf) */
);

%local dsid;

ods _all_ close;
ods rtf file="&report";

goptions ftext=Swiss;
proc reg data=&data outest=&parms;
    model &y = &x;
    output out=&out (keep=&x &y &yhat) p=&yhat;
    plot &y * &x;
run; quit;

ods rtf close;

%MEND regression;

```

Figure 6: Example of a SAS macro as unit under test

## TEST SCENARIO



Figure 7: Example for a test scenario

1. Initialize and describe test case, name the unit under test.
2. Extract test data (work.testdata) and reference data (work.reldata, work.refparm) from Excel sheet (see Figure 8).
3. Call the unit under test (see Figure 6) and extract regression parameters calculated by Excel and by SAS.
4. Check if report has been created and write a note for manual inspection into the test report.
5. Check estimated values for y-variable (yhat). Due to different floating point arithmetic in SAS and Excel, i\_fuzz specifies the tolerance.
6. Check estimate for intercept regression parameter.
7. Check estimate for slope regression parameter.

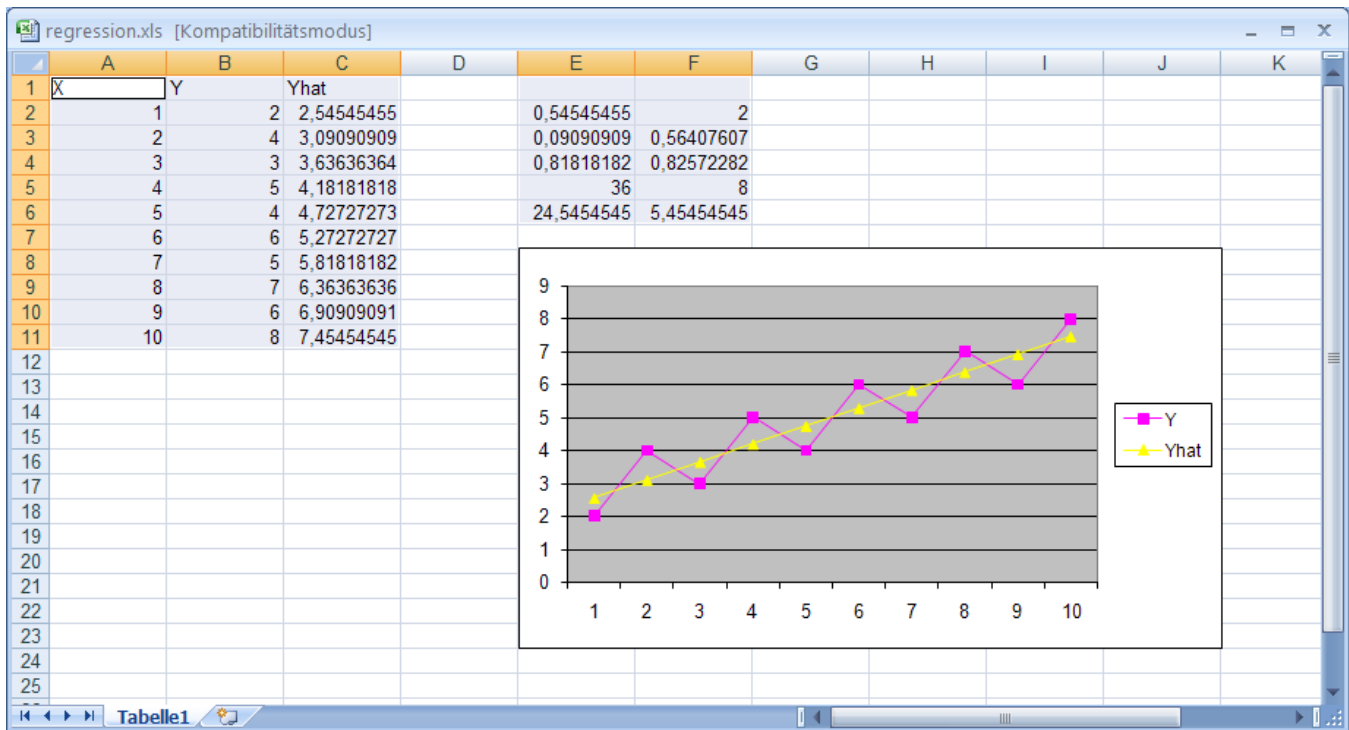


Figure 8: Excel sheet with test data and reference data for example test case

## TEST REPORT

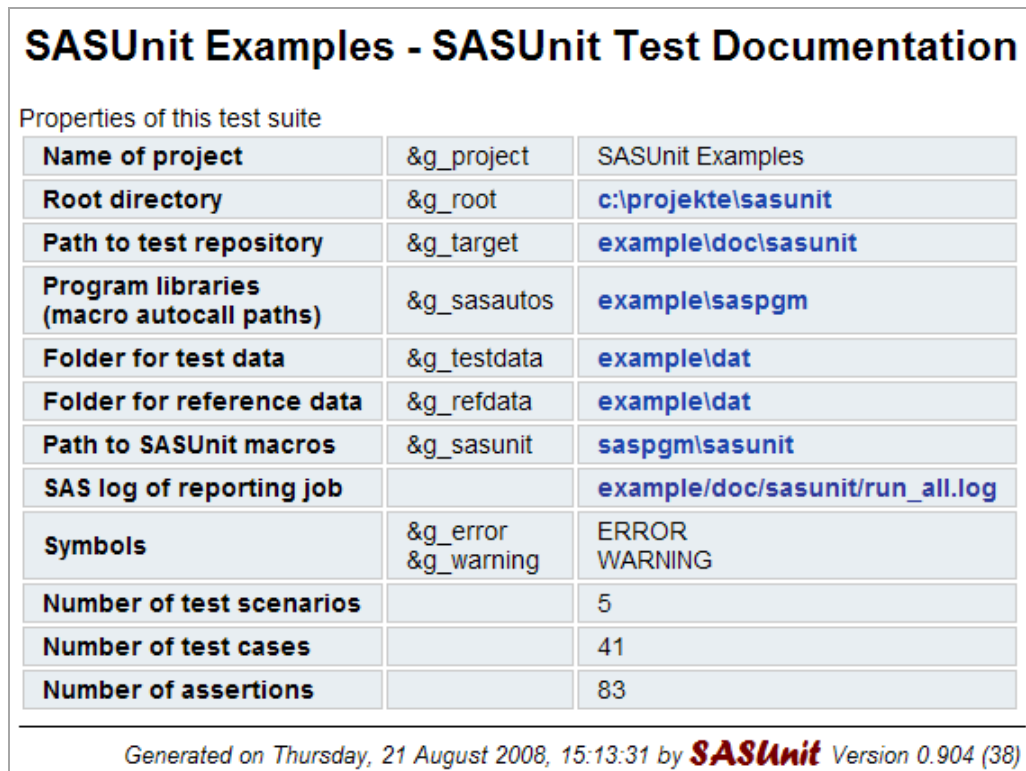


Figure 9: Main page of SASUnit report

The main page of the SASUnit report shows the environment of the test suite as found in the test repository. The project name and paths are set in %initSASUnit. Click on the paths to open folders in the windows explorer. The SAS log of the reporting job is the log of run\_all.sas. Symbols, depending on language settings, are detected automatically. Number of test scenarios, cases and assertions are counted from the test repository.

| No.                 | Test Scenario                            | Program                                            | Last Run                           | Duration | Result                                                                              |
|---------------------|------------------------------------------|----------------------------------------------------|------------------------------------|----------|-------------------------------------------------------------------------------------|
| <a href="#">001</a> | <a href="#">Tests for boxplot.sas</a>    | <a href="#">example/saspgm/boxplot_test.sas</a>    | <a href="#">21AUG2008:15:13:23</a> | 7.0s     |  |
| <a href="#">002</a> | <a href="#">Tests for generate.sas</a>   | <a href="#">example/saspgm/generate_test.sas</a>   | <a href="#">21AUG2008:13:58:49</a> | 1.8s     |  |
| <a href="#">003</a> | <a href="#">Tests for getvars.sas</a>    | <a href="#">example/saspgm/getvars_test.sas</a>    | <a href="#">21AUG2008:13:58:51</a> | 1.0s     |  |
| <a href="#">004</a> | <a href="#">Tests for nob.sas</a>        | <a href="#">example/saspgm/nobs_test.sas</a>       | <a href="#">21AUG2008:13:58:52</a> | 1.1s     |  |
| <a href="#">005</a> | <a href="#">Tests for regression.sas</a> | <a href="#">example/saspgm/regression_test.sas</a> | <a href="#">21AUG2008:14:44:29</a> | 1.7s     |  |

Figure 10: Test scenario overview of SASUnit report

In the test scenario overview, you will find one line per test scenario. Click on the scenario number or scenario name to see the test cases of that scenario (see Figure 11). Clicking on the program name opens the test scenario SAS program in the editor assigned to the extension .sas. Clicking on the last run datetime opens the SAS log of the scenario run.

| No.                 | 005                                                             |                                |                                    |          |                                                                                       |
|---------------------|-----------------------------------------------------------------|--------------------------------|------------------------------------|----------|---------------------------------------------------------------------------------------|
| Scenario            | <a href="#">Tests for regression.sas</a>                        |                                |                                    |          |                                                                                       |
| Program             | <a href="#">example/saspgm/regression_test.sas</a>              |                                |                                    |          |                                                                                       |
| Last Run            | <a href="#">21AUG2008:14:44:29</a>                              |                                |                                    |          |                                                                                       |
| Duration            | 1.7s                                                            |                                |                                    |          |                                                                                       |
| Test Cases          |                                                                 |                                |                                    |          |                                                                                       |
| No.                 | Test Case                                                       | Unit under Test                | Last Run                           | Duration | Result                                                                                |
| <a href="#">001</a> | <a href="#">Compare linear regression between Excel and SAS</a> | <a href="#">regression.sas</a> | <a href="#">21AUG2008:14:44:29</a> | 0.7s     |  |

Figure 11: Details for test scenario of SASUnit report, lists all test cases

The top part of the test scenario details page shows all the details about the test scenario. The bottom part lists all the test cases of the test scenario. Clicking on the test case number or description opens the test case details page (see Figure 13). Clicking on the unit under test opens the corresponding program. Clicking on the last run datetime opens the log view (see Figure 12).

|                                                                                                |              |
|------------------------------------------------------------------------------------------------|--------------|
| <b>Errors : 1 ==&gt; <a href="#">1</a></b>                                                     |              |
| <b>Warnings: 0</b>                                                                             |              |
| NOTE: PROCEDURE PRINTTO used (Total process time):                                             |              |
| real time                                                                                      | 0.00 seconds |
| cpu time                                                                                       | 0.00 seconds |
| 93        %boxplot(data=XXXXX, x=visit, y=sbp, group=med, report=&g_work\report8.rtf)          |              |
| ERROR: boxplot: Data set XXXXX does not exist                                                  |              |
| 94        %assertLogMsg(i_logMsg=ERROR: boxplot: Data set XXXXX does not exist)                |              |
| MPRINT(ENDTESTCALL):    PROC PRINTTO LOG="c:\projekte\sasunit\example\doc\sasunit\log\001.log" |              |
| PRINT="c:\projekte\sasunit\example\doc\sasunit\tst\001.lst" ;                                  |              |
| MPRINT(ENDTESTCALL):    RUN;                                                                   |              |

Figure 12: Log view in the SASUnit test report with hyperlinks for errors and warnings

The log view shows the SAS log which has been translated into the HTML format. For every error message and every warning message, a hyperlink appears at the top of the page for your convenience.

Scenario No. 005  
 Scenario Tests for regression.sas  
 Program example/saspgm/regression\_test.sas  
 Last Run 21Aug2008:14:44:29  
 Duration 1,7s  
 Test Case No. 001  
 Test Case Compare linear regression between Excel and SAS  
 Unit under Test regression.sas  
 Last Run 21Aug2008:14:44:29  
 Assertions

| No. | Assertion     | Description                               | Expected                                  | Actual                               | Result                              |
|-----|---------------|-------------------------------------------|-------------------------------------------|--------------------------------------|-------------------------------------|
| 001 | assertReport  | please compare SAS chart with Excel chart | .xls                                      | .rtf                                 | <input type="checkbox"/>            |
| 002 | assertColumns | compare estimated values                  | Table listing<br>DSLABEL LABEL<br>COMPVAR | Table listing<br>Comparison<br>LABEL | <input checked="" type="checkbox"/> |
| 003 | assertEquals  | compare intercept parameter               | 2                                         | 2                                    | <input checked="" type="checkbox"/> |
| 004 | assertEquals  | compare slope parameter                   | 0.545455                                  | 0.545455                             | <input checked="" type="checkbox"/> |
| 005 | assertLog     | Scan log                                  | Errors: 0,<br>Warnings: 0                 | Errors: 0,<br>Warnings: 0            | <input checked="" type="checkbox"/> |

Figure 13: Details of a test case in SASUnit, lists all assertions

The top part of the test case details page shows all the details about the test case itself. The bottom part lists all assertions of the test case. Depending on the assertion type, information about expected and actual results or hyperlinks to them can be found in the corresponding columns.

## CONCLUSION

SASUnit is available for free at [www.redscope.org/sasunit](http://www.redscope.org/sasunit). Any suggestions and field reports will be greatly appreciated.

## REFERENCES

- [1] SASUnit download on Redscope.org, see <http://www.redscope.org/sasunit/en>.
- [2] Wikipedia: Unit testing, [http://en.wikipedia.org/wiki/Unit\\_test](http://en.wikipedia.org/wiki/Unit_test), retrieved 2008-08-22.
- [3] ISPE: GAMP 5: A Risk-Based Approach to Compliant GxP Computerized System, 2008.
- [4] Wikipedia: List of unit testing frameworks, [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks), retrieved 2008-08-22.
- [5] van Heesch, Dimitri: Doxygen source code documentation generator tool, <http://www.stack.nl/~dimitri/doxygen>.

## ACKNOWLEDGMENTS

Acknowledgements go to Klaus Landwich for his contribution of %assertLibrary.

## CONTACT INFORMATION

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

Andreas Mangold  
 HMS Analytical Software GmbH  
 Rohrbacher Str. 26  
 69115 Heidelberg  
 Germany  
 Fax: +49 6221 60 51 - 0  
 Email: [andreas.mangold@analytical-software.de](mailto:andreas.mangold@analytical-software.de)  
 Web: <http://www.analytical-software.de>

Brand and product names are trademarks of their respective companies.