Paper 3199-2019

The Power of a Good SUIT. SAS® Unit and Integration Testing

Cameron Lawson, Selerity

ABSTRACT

Test automation is an integral component of modern software development. Many programming languages and the communities built around them either enjoy integrated test libraries as part of the language's standard library or have adopted a de facto standard such as Python's unit test module or Java JUnit libraries. Many libraries that have been developed or promoted in the SAS® community have been adopted to varying degrees. This includes notable projects such as SASUnit and the SAS Operational Quality testing tool. All of these libraries are focused on the programmer and geared toward unit testing approaches. SUIT is a new testing tool offered to the community that not only provides familiar JUnit-style test assertions for the SAS® programmer, but also keyword-driven test-case development more suitable for business-focused SAS® users and testers to adopt automated testing. With a wide range of out-of-the-box test cases for SAS® 9 and Viya™, users can quickly develop robust test cases. The extendable architecture easily enables business units to develop their own keyword libraries. Using the power of SAS®, tests can be developed across the full lifecycle of your development, from data to user interface. Test results integrate into common continuous integration tools such as Jenkins and Bamboo. A modern-browser-based interface provides a seamless interface to SAS middle-tier applications.

INTRODUCTION

As a SAS® consultant I constantly engage with customers performing projects that are building business solutions based on SAS® and Viya™. These projects range from initiatives exploring new revenue streams and services through to highly complex business critical solutions that underpin significant business processes and revenue streams.

Project and delivery teams often have difficulty in testing SAS® based solutions. Quite often the testing team come from different backgrounds and do not have the sufficient skills to write and analyze SAS® code or test teams have difficulty integrating the execution and results from bespoke SAS® test scripts. Delays often occur as these teams navigate the complexities of SAS® code. Often the testing team will get the SAS® developers write the tests for them creating issues with quality control and regulatory processes.

In conjunction, many SAS® programmers do not utilize a testing framework as part of their development lifecycle and either do not automate tests or they continually re-write the same logic over and over through different projects to test new features. Research has shown that test automation not only greatly improves the reliability of software development, but it also improves the speed of delivery (Dustin, Rashka and Paul. 1999 p392.)

The availability of open source SAS® centric frameworks is limited. There are some exceptions such as SASUnit which provides J-Unit style test assertions. The SUIT framework looks to address some of these limitations by making a framework available that is simple to use, improves development efficiency and can be used by both SAS® programmers and non-programmers alike. SUIT does not aim to replace existing test frameworks currently available. It seeks to work alongside these and fill the gaps often encountered by testing teams, user acceptance testers and SAS® developers.

DESIGN PRINCIPLES

The development of SUIT is guided by a series of overarching design principles outlined in Table 1:

Principle	Description
Build an accessible framework	Construct a testing framework that can used by both SAS® and non-SAS® users.
Build an extendable framework	Construct a framework that can be adapted and improved over time following core software design principles.
Make the framework simple	Provide interfaces that are simple to use.
Make the framework play well with others	Allow the framework to integrate into automation frameworks and interface with other systems using well known and supported protocols.

Table 1. SUIT Design Principles

SUIT is open source and will be available on or around the SAS Global Forum 2019 (May 2019). A premium Enterprise version is also in development which will provide Enterprise grade features to SUIT and is suitable for organizations that have an embedded testing culture and want premium level support. Table 2 provides a comparison of features between SUIT and SUIT Enterprise:

Feature	SUIT Core	SUIT Enterprise
SUIT Programming Interface	X	
SUIT File Interface	X	
SUIT Command Line Interface	X	
Core Test Keywords	X	
Web Interface and REST API		X
Viya™ containers / Docker support		X
Ansible Modules and Chef Recipes		X
Group and User Security		X
Suit File Editor		X
Premium Keyword Libraries (UI testing, OWASP)		X
Level 1 and 2 help desk support		X

Table 2. SUIT Core and SUIT Enterprise Features.

SUIT Enterprise is expected to be available in the third quarter of 2019 (July – September).

Selerity is also looking for contributors to SUIT Core. Contributors will have heavily discounted access to SUIT Enterprise. For further information please contact support@seleritysas.com.

SUIT ARCHITECTURE

The core of SUIT is built with the SAS® language. Table 3 outlines the key objects within the SUIT framework:

Object	Description
Keywords	These are the building blocks for test cases. A keyword implements an action or asserts truthiness.
Test Cases	A Test Case is a collection of one or more keywords. Test Cases are typically created to test an object or a process. A test case can produce multiple test results.
Test Suites	A Test Suite is a collection of test cases that are executed as one entity. A Test Suite will output a series of object showing the results of keywords within test cases as well as producing a SUIT results log and additional reporting.
Test Results	A Test Result is produced by executing Test Suites. A Test Result records the outcomes of test keywords executed within each test case and grouped by a test suite.

Table 3. SUIT Components.

SUIT is written to accommodate SAS® and Viya™ programmers as well as test practitioners who do not have experience with the SAS® language. The typical workflow for SUIT is shown in Figure 1:

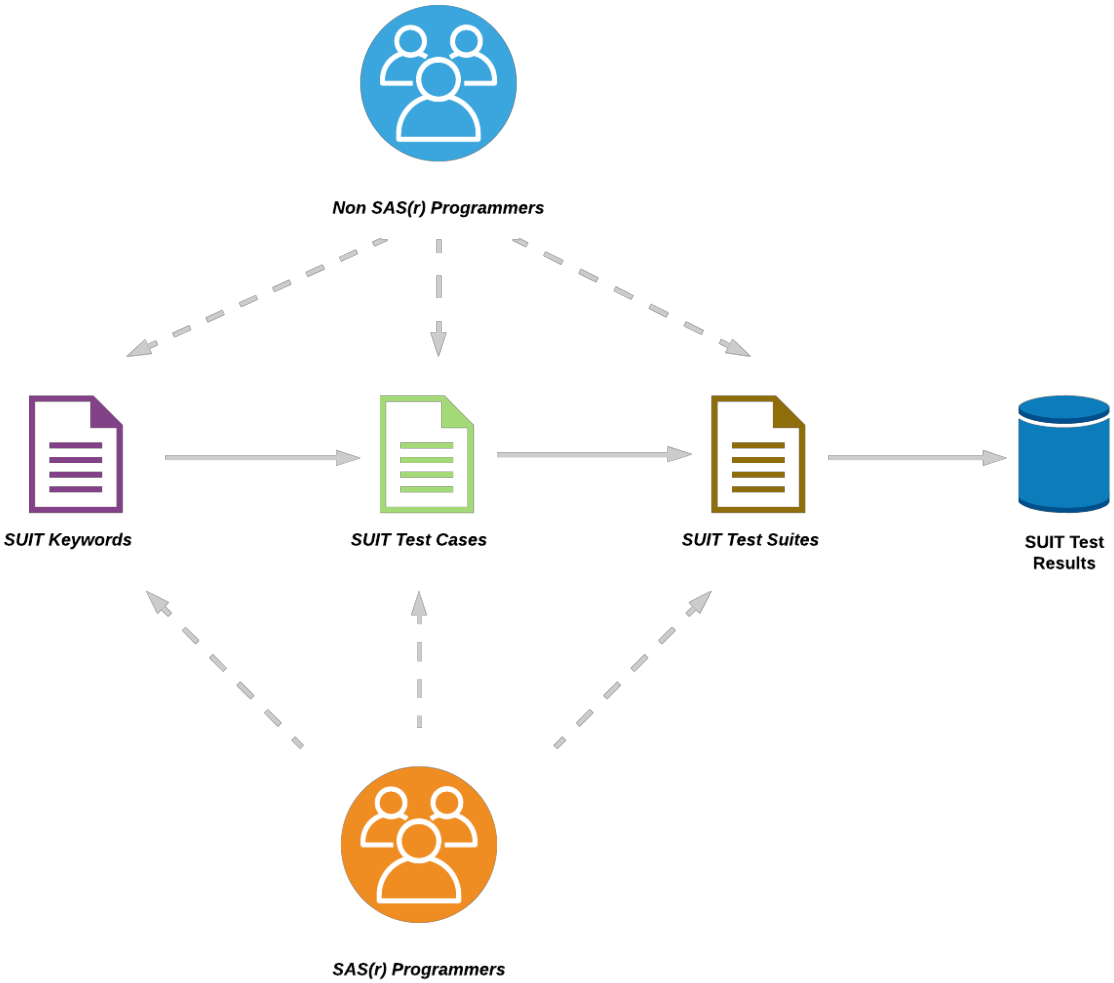


Figure 1. SUIT Workflow.

Interfaces are available for programmers and non-programmers. The execution of SUIT can be invoked manually by a user or incorporated into continuous integration / development processes such as Jenkins, Bamboo or other process orchestration tools like Ansible or Chef.

SUIT implements a plugin-based architecture to extend SUIT functionality, a flexible library-based approach to extend and customize keywords and a range of lifecycle hooks to trigger custom tasks and plugin processes at every stage of the SUIT testing lifecycle.

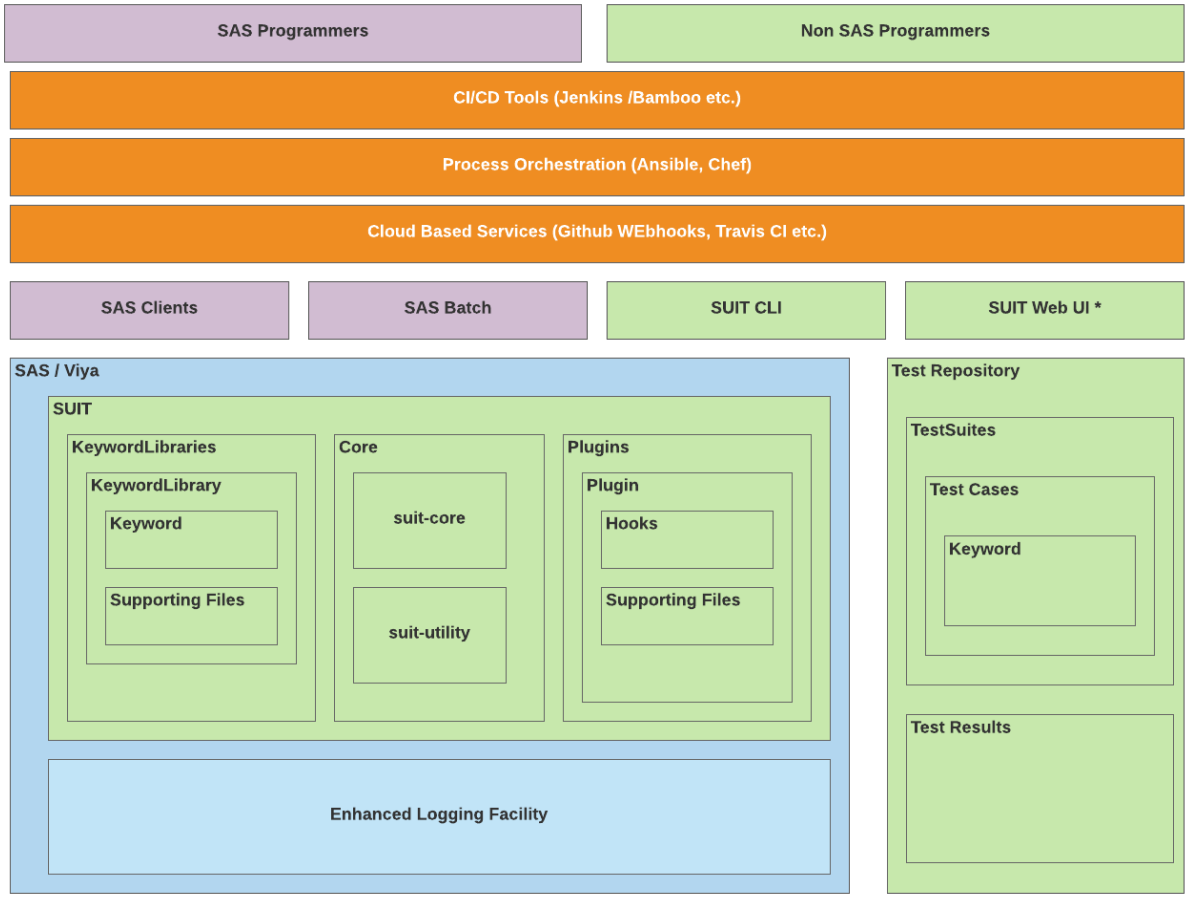


Figure 2. SUIT Services Diagram.

SUIT framework objects can be written using either SAS® Code or using a simple text markup. This markup language is loosely based on the Markdown language and saved with the extension of ".suit" or ".md".

SUIT results are recorded to a test suite log file which is implemented using the *SAS® Enhanced Logging Framework*. This framework is based on the log4j Java library and provides advanced logging capabilities. Results can be streamed to a console, written to databases or to multiple log files. SUIT then parses the result logs into test reports. These reports can be viewed in your browser or integrated into test reporting tools that support the J-Unit XML format.

Tests support the concept of decorators which are macros that alter default behaviors of keywords or provide run-time extensions in functionality. A tester can "decorate" a test by calling the required decorator immediately before a test keyword. Decorators are maintained in a plugin within SUIT and can also be extended. Table 4 outlines the standard decorators SUIT core ships with:

Decorator	SUIT File Keyword	Description
%expect_failure	Expect failure	Will record a failed test as passed. This is useful for performing negative assertions in tests.

Decorator	SUIT File Keyword	Description
%skip_test	Skip test	Will skip execution of the test and record the test as skipped in test results.
%tags	Tags	Will tag tests by the given tags. This is slated for a future release that will allow test execution by tag name.

Table 4. SUIT Decorators

USAGE SCENARIOS

SAS® OR VIYA™PROGRAMMER

SUIT is written in the SAS® language. Using SUIT is no different to writing any other SAS® program. It is recommended however that a user has an understanding of how to invoke SAS® Macros. Typically, a programmer will write test cases either inside projects such as unit and functional tests and some organizations will also maintain separate user acceptance and regression test repositories. A SAS® or Viya™ programmer simply writes their desired test cases and saves them in a location accessible to either the SAS® or Viya™ server and executes test suites using the **%suit_testsuite()** macro.

SOFTWARE TESTER

Software testers not accustomed to the SAS® language can write test keywords and test cases using the suit file format. This format described further in this paper is a simple text file format that is parsed into SAS® code and then executed like any other SAS® program. As testers may not have direct access to SAS or have a limited knowledge in the SAS® interface, a command line tool written in python is also provided called suit-cli. A tester can write tests locally on their client machine and upload them to the SAS® or Viya™ server and then execute test suites. The client simply needs ssh access to the SAS® server and write access to the location of where test keywords, test cases and test suites are stored.

AUTOMATION SCENARIOS

SUIT also supports automation. This is a common scenario where test automation is used as a control gate in the promotion or committing of code. SUIT can be invoked by a CI/CD application such as Jenkins or Bamboo via the *suit-cli* tool. SUIT will produce a J-Unit style xml results file that then be used by Jenkins to report test history and in Bamboo via the J-Unit XML Parser. A premium version of SUIT is also in active development and this will also provide a custom Ansible module and Chef recipe that can be invoked to integrate SUIT testing into configuration management processes.

INSTALLATION AND CONFIGURATION

SUIT is easy to install and flexible to accommodate varying business requirements within SAS® and Viya™ installations. In the simplest form, SUIT requires the following:

- Folders to store Test Cases, Test Suites and Application Logs.
- The setting configuration variables to point to the location of SUIT, and your Test Case, Test Suite and Application Log Paths.
- SAS® Metadata Server details and Viya™ server details if running through Viya™

SINGLE USER

SUIT can be used from SAS® University Edition quite easily. The following steps outline the setup:

1. Clone SUIT from https://github.com/selerity/suit into your shared folders location. If you are using the latest version of SAS® University Edition, you can use the newly available git functions to clone the repository as required.
2. In your shared folder location under SASUniversityEdition/myfolders create the following new folders:
    a. suit_logs
    b. suit_results
    c. suit_testcases
    d. suit_testsuites
3. Edit your autoexec file and add the following:

```
options set = SUIT_HOME "/folders/myfolders/suit";
options set = SUIT_TESTCASE_HOME "/folders/myfolders/suit_testcases";
options set = SUIT_RESULTS_HOME "/folders/myfolders/suit_results";
options set = SUIT_LOG_HOME "/folders/myfolders/suit_logs";
%include "/folders/myfolders/suit/suit_init.sas";
```

Output 1. SUIT Configuration Variables.

SERVER

For server installations, the configuration can be implemented in a number of ways. The installation is essentially the same for SAS® 9 and Viya™ environments as SUIT utilizes the programming interface for Viya™ which initialize SAS® Workspace servers. SUIT Enterprise will also include support for initiating SUIT as a containerized process allowing users to create and teardown test environments easily.

For the Application Context you wish to run SUIT from, it is recommended you place SUIT under your SASEnvironment/ path however it can be installed in any server-side location that you store SAS® Code.

You need to invoke *suit/suit_init.sas* from your *appserver_autoexec_usermods.sas file* or by adjusting the INITSTMT option on your workspace server. You can set configuration variables either in your operating system, sasv9_usermods.cfg file or your appserver_autoexec_usermods.sas file. The *suit/suit_init.sas* file by default looks for the following environment variables. The minimum required configuration variables are outlined below in Table 5:

Variable	Value
SUIT_HOME	The path to where the suit framework is located.
SUIT_TESTCASE_HOME	The top-level path to where test cases will be stored.
SUIT_RESULTS_HOME	The top-level path to where test results are written.

Variable	Value
SUIT_LOG_HOME	The top-level path to where suit application logs is written.

Table 5. SUIT Configuration Variables.

The paths you set for these variables need to have read, write, execute permissions for SAS and suit users. The SUIT_HOME location can be set with read, execute permissions. It is recommended to have separate top-level directories for test cases and test suites. This ensures that test suites do not have circular references to files.

THE SUIT FILE FORMAT

A suit file is a simple text-based markup format that can be used by a tester to create test keywords, test cases and test suites. This markup is parsed by SUIT into SAS® code and executed on the SAS® or Viya™ environment. The SUIT file format is separated into different sections depending on the role of the markup. Sections are marked up using a single hash (#) followed by the section name. The format currently supports the following sections

METADATA

The metadata section contains descriptive information for the object being created. The information in this section is essentially descriptive and the purpose is to inform test results and documentation output. The information in this section typically contains:

- The name of the object
- A description of the object
- Authorship information
- Any other key / value pair that assists with the understanding of the object.

At a minimum, a name is required. All other parameters are optional. An example metadata section is as follows:

```
# Metadata
Name      Example SUIT Test Case
Desc      An example of the different capabilities for a suit file.
Author    Cameron Lawson
Project   SUIT
```

Output 2. SUIT File Metadata Section Example.

VARIABLES

Variables allow the setting of constants that can be used within keywords and tests. Variables are defined as key / value pairs with values separated via a TAB characters. Variables may also reference other variables and they should be referenced using the SAS & token. Output 3 shows a typical Variable structure:

```
# Variables
stagelib          stagelib
sourcetable1      &stagelib..s_assets
stagetable1       &stagelib..s_assets
tfmtable1         &assetlib..t_assets_keys
```

Output 3. SUIT File Variables Section Example.

KEYWORDS

The keywords section allows the creation of custom keywords. Keywords are written as the name of the keyword followed by the parameters of the keyword each separated by TABs followed by subsequent lines indented by a TAB. The following example shows this syntax:

```
# Keywords
Dataset test    library     dataset
    Dataset exists    &library..&dataset
    Variable exists   &library..&dataset      rowid
    Variable exists   &library..&dataset      dt_created
    Variable exists   &library..&dataset      dt_modified
    Format equals     &library..&dataset      dt_created  datetime20.
    Format equals     &library..&dataset      dt_modified datetime20.
    Dataset is not empty    &library..&dataset
    Variable unique   &library..&dataset      rowid
```

Output 4. SUIT File Keywords Section Example.

The above output will create a SAS® Macro called **%dataset_test** and will look as follows:

```
/* Variables */
%let saslib = %nrbquote(mylib);
%let dataset1 = %nrbquote(data001);
%let dataset2 = %nrbquote(data002);

/* Keywords */
%dataset_test(library,dataset) / des='Custom Keyword created in suit test
file xxxx.suit by cameron';
    %dataset_exists(&library..&dataset);
    %variable_exists(&library..&dataset, rowid);
    %variable_exists(&library..&dataset, dt_created);
    %variable_exists(&library..&dataset, dt_modified);
    %format_equals(&library..&dataset, dt_created,datetime20.);
    %dataset_is_not_empty(&library..&dataset);
    %variable_unique(&library..&dataset,rowid);
%mend;

/* Tests */
...
```

Output 5. SUIT File Keyword Parsed Output.

Keywords are generally written using the SAS® Macro language. The main use case for creating keywords in a suit file is to combine multiple keywords into a single keyword.

TESTS

The tests section contains either the test case or test suite you wish to execute. Tests are written as one line per keyword with each parameter for the keyword separated by a tab. Variables used in the tests and keyword sections are referenced using the common SAS® syntax of an ampersand (&). Keyword parameters can either be referenced variables or absolute values. Keywords that are defined in a keywords section can also be used:

```
# Tests
Library exists  &saslib
Variables exist metric1     metric2     metric3
Dataset test    &saslib     &dataset1
Dataset test    &saslib     &dataset2
```

Output 6. SUIT File Tests Section Example.

COMPLETE EXAMPLE

```
# Metadata
name  test suit format
desc  this is a description of the test
author      Cameron Lawson
version     1.0
tags  unittest

# Variables
stagelib          stagelib
sourcetable1      &stagelib..s_assets
stagetable1       &stagelib..s_assets
tfmtable1         &assetlib..t_assets_keys

# Keywords
run daily staging jobs
      run sas program         /opt/foo                stage_*.sas
      run sas program         /opt/foo/stats.sas

run daily transform jobs
      run sas program         /opt/foo                transform_*.sas

# Tests
run daily staging jobs
assert sas log                /saslogs/staging.log    ERROR             0
count rows                    &sourcetable1                 sourcerowcount
count rows                    &stagetable1                  stagerowcount
assert equals                 &sourcerowcount               &stagerowcount

run daily transform jobs
assert sas log                /saslogs/transform.log  ERROR             0
count rows                    &tfmtable1                    tfmrowcount
assert equals                 &stagerowcount                &tfmrowcount
```

Output 7. SUIT File Example.

SAS® PROGRAMMING INTERFACE

The suit file format is primarily designed for test practitioners who do not have detailed SAS® programming knowledge. Those with SAS® programming knowledge may wish instead to use the programming interface. The interface uses the SAS® Macro language and suit objects can be written just like any other SAS® program. This allows an end user to easily integrate the process of test-driven development into the development lifecycle.

TEST CASES

Tests are written using available keywords. A tester can also write new keywords directly within tests or by extending the available global keyword libraries. It is recommended to write tests separately to the SAS® objects being tested. If your tests have dependencies such as the execution of a certain program, you can include the program to be executed in the test case. Keywords available to suit files are also available to the SAS® programmer with the difference being that keywords must be referenced using valid SAS® macro naming conventions. Keywords in the suit file format will replace spaces with underscores, so a test keyword called ***dataset exists*** is referenced as ***%dataset_exists*** in the programming interface:

```
/**
Test Program for something critical.
:author: Cameron Lawson
:project: SUIT
*/
%let logpath = %nrbquote(/path/to/log.log);
proc printto log = ”&logpath.” new;
run;
%include “/my/program/i/want/to/test.sas”;
proc printto;
run;

/* Start of test keywords */
%saslog_errors_equal(&logpath,0);
%saslog_warnings_equal(&logpath,0);
%expect_failure;
%dataset_exists(mylib.mydata);
```

Output 8. SUIT Programming Test Case Example.

KEYWORDS

Keywords can be defined either locally or by extending the global SUIT keyword library. Keywords may or may not generate a test result. A Test Result will write a result to the SUIT log and will record a test as either Passed or Failed. To implement this, SUIT provides two macros which can be used by a developer:

- %suit_pass_test
- %suit_fail_test

Each macro takes the following parameters:

Parameter	Value
testname	The name of the test
msg	A custom message to write to the SUIT log.

Table 6. SUIT Log Message Parameters

Keywords should implement the following parameters:

Parameter	Purpose
testname	A placeholder for the name of the test. This should be given a default value of &sysmacroname.

Table 7. SUIT Keyword Required Parameters.

Keyword macros should also include the DES=" option as this is used to assist documenting the keyword. The naming of keywords should reflect a natural language style syntax and words separated with an underscore.

In conjunction to the above, there are some additional Global variables which can be set to influence the behavior of SUIT. For further details of these see the developer documentation at https://suit.seleritysas.com.

SUIT ships with a J-Unit style assertion keyword library. This library provides test result creation for numerous operations and can be used to simplify the creation of higher-level keywords. For example, most keywords in the built-in libraries use the **%assert** keyword to produce a test result.

An example keyword macro can be seen below which implements the following:

```
%macro assert(actual,expected,operator,msg=,testname=&sysmacroname.) /
        des = "Tests that an expected value meets an actual value by a given
operator";


    %local _testname  _passmsg _failmsg actualc expectedc;
    %let _testname = &testname.;
    %let _passmsg = %nrbquote(&actual. &operator. &expected. &msg.);
    %let _failmsg = %nrbquote(&actual. was expected to be &operator.
&expected.  &msg.);


    %if (&SUIT_IGNORE_CASE_IN_STRINGS) %then %do;
        %let actual = %upcase( &actual. );
        %let expected = %upcase( &expected. );
    %end;


    %if %datatyp(&expected) eq CHAR %then %do;
        %let actualc = %sysfunc(md5(&actual),$hex32.);
        %let expectedc = %sysfunc(md5(&expected),$hex32.);
        %if &actualc &operator &expectedc %then %do;
            %suit_pass_test(test=&_testname,msg=&_passmsg.);
        %end;
        %else %do;
            %suit_fail_test(test=&_testname,msg=&_failmsg.);
```

```
        %end;
        %return;
    %end;
    %else %do;
        %if %eval( &actual. &operator &expected. ) %then %do;
            %suit_pass_test(test=&_testname,msg=&_passmsg.);
        %end;
        %else %do;
            %suit_fail_test(test=&_testname,msg=&_failmsg.);
        %end;
        %return;
    %end;
%mend assert;
```

Output 9. SUIT Keyword Programming Example.

In order to obtain a listing of available keywords in your session you can run the **%suit_libdoc** macro. This will produce a listing of available keywords, their parameters and the description of the macro recorded in the DES attribute. Table x shows the output of running **%suit_libdoc(assert)**:

Assert Test Keyword Library
Keyword Library Metadata

Obs	attr	value
1	Description	The assert library provides a low level J-Unit style assertions that can be used either as is or as part of higher level composite keyword libraries.
2	Commit ID	2c89266409c8917977d6c68876caabf8af8e4165
3	Author	Cameron Lawson
4	Email	royalsouvenir@users.noreply.github.com
5	Commit Message	Initial commit
6	Time of Commit	1540610723

Available Keywords

Test Keyword	Parameters	Description
Assert	actual,expected,operator,msg=,testname=&sysmacroname.	Tests that an expected value meets an actual value by a given operator
Assert Equal	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts the equality of two given values. The assertion operates for numeric and character values.
Assert Greater Or Equal	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts an actual value is greater or equal to an expected value. Both numeric and character values are supported.
Assert Greater Than	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts an actual value is greater than an expected value. Both numeric and character values are supported.
Assert Less Or Equal	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts an actual value is less or equal to an expected value. Both numeric and character values are supported.
Assert Less Than	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts an actual value is less than an expected value. Both numeric and character values are supported.
Assert Not Equal	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts an actual value is not equal to an expected value. Both numeric and character values are supported.
Assert Number	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts a given value is numeric
Assert Sas Log	actual,expected,operator,msg=,testname=&sysmacroname.	Asserts the count of a SAS Log Message type matches a given operator

Figure 3. SUIT LibDoc Example Output.

TEST SUITES

To execute a test suite, a SAS® programmer can call the **%suit_testsuite** macro. This macro operates in a similar fashion to a suit file except it accepts a name and description for the test suite followed by an optional list of paths and search patterns for tests to discover and include. In the SUIT configuration file, parameters are set for the default test case path and default search pattern for test names. The default values for these are as follows:

Parameter	Default Value
SUIT_DEFAULT_TEST_CASE_PFX	test_*
SUIT_TESTCASE_HOME	<User Specified>

Table 8. SUIT Test Case Default Parameters.

If a user does not include any paths or search patterns, these default values are used. The below examples show invocations of **%suit_testsuite** using default parameters and then supplying multiple paths for test cases:

```
/* Test Suite with Default Parameters */
%suit_testsuite(
standard_testsuite,
tests executed from the default testcase path and pattern
);

/* Test Suite with a Custom List of Paths */
%suit_testsuite(
custom_testsuite,
A custom test suite with multiple paths,
/home/cameron/private_tests/unittest_*,
test_critical*,
test_smoke*,
test_uat*
);
```

Output 10. SUIT Test suite Code Examples.

TEST RESULTS

By default, SUIT will output results of tests into the SUIT_TESTRESULTS_HOME path specified in the configuration. The results directory is structured as follows:

Path	Purpose
/logs	SAS® program logs and listings and the SUIT results log.
/report	SUIT html and j-unit xml reports.
/tests	A copy of each suit and sas program executed in the test suite.
/data	SAS Datasets containing parsed test results and test suite metadata.

Table 9. SUIT Test Result Structure

Each test result directory by default is named by the test suite executed and a datetime to ensure uniqueness. The output name of this directory can be changed within the SUIT configuration file.

LOGS DIRECTORY

Test results are written to a custom SAS® log which is created via the *SAS® Enhanced Logging Facility*. Each result contains the following information:

Attribute	Value
Datetime	Datetime of the entry
Result	PASS or FAIL
SUIT_SESSION_ID	A unique ID for the invocation of SUIT.
Test Start Datetime	Datetime Test Keyword initiated.
Test End Datetime.	Datetime Test Keyword terminated.
Test Suite	Name of the Test Suite
Test Case	Filename of the Executing Test Case.
Test	Name of the executing Test Keyword.
Level	The SAS Logging Level (ERROR,WARN,NOTE,DEBUG)
Msg	The custom message passed to the Test Keyword.

Table 10. SUIT Result Log Attributes

This log is parsed by the SUIT and then turned into results reporting and stored in the SUIT_RESULT table.

TESTS DIRECTORY

Each test discovered is copied to the results directory in order to allow the auditing of tests at time of execution. If a suit file is included in the test suite, two files are written to the *tests* directory:

- The original suit file.
- The parsed suit file in .sas format.

DATA DIRECTORY

Test Suite metadata, test case discovery and test results are stored in SAS® datasets within the */data* directory. The SUIT Enterprise product collects these tables and stores them into a central database to allow results reporting and analytics on test results across an organization.

The tables within the directory are as follows.

Dataset	Purpose
SUIT_TESTSUITE	Metadata about the test suite including name, description, execution time, author
SUIT_TESTCASE	A list of test cases executed by a test suite.
SUIT_RESULT	A listing of executed keywords that produce a test result and their execution time and pass / failure status.

Dataset	Purpose
SUIT_HOOKS	A list of executed plugins by test lifecycle stage.

Table 11. SUIT Output Datasets.

For full information on the attributes within each table please consult the SUIT online help documentation at https://suit.seleritysas.com.

RESULTS DIRECTORY

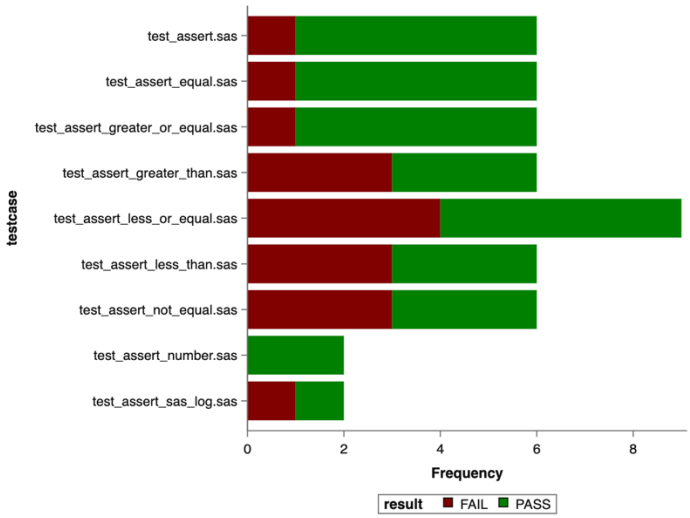
The results directory contains an ODS html report containing a listing of all test cases and test keywords executed and their results. The output report contains links to the test cases executed and the full SAS Log for each test case.

In conjunction, a J-Unit formatted XML report called *report.xml* is also outputted to the directory. This document can then be used to integrate into CI/CD systems.

SUIT Test Results

The following report documents the test results for the testcase assert_library_unittests whose purpose is to Unittests for the assert test keyword library . Test results are stored i
/folders/myfolders/suit_results/suitresults_assert_library_unittests.24MAR19:19:12:46.600 . The SUIT ID for this test execution is ba65623d-6186-e745-b5e7-480c47d4

- Total Tests Executed: 49
- Total Tests Passed: 32
- Total Tests Failed: 17
- Failure Rate: 34.69%



Results by Test Case

Test Case	Test Case Path	Test Case Duration	Tags	Tests Executed	Percent Passed	Tests Passed	Tests Failed
test_assert.sas	/folders/myfolders/suit_testcases	0:00:00		6	83.33%	5	1
test_assert_equal.sas	/folders/myfolders/suit_testcases	0:00:00		6	83.33%	5	1
test_assert_greater_or_equal.sas	/folders/myfolders/suit_testcases	0:00:00		6	83.33%	5	1
test_assert_greater_than.sas	/folders/myfolders/suit_testcases	.		6	50.00%	3	3
test_assert_less_or_equal.sas	/folders/myfolders/suit_testcases	0:00:00		9	55.56%	5	4
test_assert_less_than.sas	/folders/myfolders/suit_testcases	0:00:00		6	50.00%	3	3
test_assert_not_equal.sas	/folders/myfolders/suit_testcases	0:00:00		6	50.00%	3	3
test_assert_number.sas	/folders/myfolders/suit_testcases	0:00:00		2	100.00%	2	0
test_assert_sas_log.sas	/folders/myfolders/suit_testcases	0:00:00		2	50.00%	1	1

Test Case test_assert.sas

Test Case test_assert.sas

Test Suite		assert_library_unittests
Test Case		test_assert.sas
Run By		sasdemo
Test Case		Test Case
SAS Log		SAS Log
Test Case		
Test Steps		
PASS	Test	Assert
Test Log Output		
`1 ge 1 test for numeric input`		
PASS	Test	Assert
Test Log Output		

Output 11. SUIT HTML Output.

SUIT CLI

A command line utility is also available. The cli provides a client-side interface for non-SAS® users to upload, execute and download SUIT keywords, test cases and test suites.

The cli is written in Python and operates on both Linux and Windows. The cli is pre-compiled with an embedded Python interpreter and all the required libraries to operate. The cli uses the *Pycrypto* library which is a C based extension for Python. As a result, when downloading the cli, ensure you obtain the right distribution for your platform.

To see the available options in the cli simply run *suit-cli* using the *-h* option:

```
suit-cli -h
suit-cli –help
```

The output from this produces the following:

```
**************************************************************************
***

                 _____         _               _  _
   ***          /  ___|       | |             (_)| |
 ********       \ `--.    ___  | |  ___   _ __   _ | |_  _   _
***********      `--. \ / _ \| | / _ \| '__|| || __|| | | |
 ********       /\__/ /|  __/| ||  __/| |   | || |_ | |_| |
   ****         \____/  \___||_| \___||_|   |_| \__| \__, |
                                                      __/ |
                                                     |___/


SUIT(TM) - SAS(r) Unit and Integration Testing
------------------------------------------------------------------------
---
version : 0.1
build   : 1
------------------------------------------------------------------------
---


positional arguments:
  {libdoc,testcase,testsuite}
                        Available Sub-Commands
    libdoc              Execute Keyword Library Documentation
    testcase            Manage SUIT Test Case Files
    testsuite           Manage SUIT Test Suites

optional arguments:
  -h, --help            show this help message and exit
```

```
  -v, --version         show program's version number and exit
  -p PROFILE, --profile PROFILE
                        Specify which stored profile to use to connect to the
                        server side instance of SUIT. By default, SUIT will
                        look for a .suitprofile file under your home
                        directory.

------------------------------------------------------------------------
---
Don't like the purity of command line?
Try SUIT Enterprise. Go To https://suit.seleritysas.com
```

Output 12. SUIT CLI Help Screen

The SUIT cli provide three main options:

- Libdoc: Display Keyword Documentation
- TestCase: Upload Test Cases to the Server
- TestSuite: Define and Run SUIT Test Suites.

Full help documentation is available at https://suit.seleritysas.com. SUIT comes with pre-compiled binaries for Linux and Windows.

EXTENDING SUIT

If you are using SUIT we encourage you to extend the framework to match the needs of your organization and processes. To assist, a number of utilities are available and Selerity maintain detailed developer documentation at https://suit.seleritysas.com

EXTENDING PLUGINS.

A new plugin can be created by calling **%suit_create_plugin** and passing in the name of the plugin you wish to create. This will create a new empty plugin structure in the SUIT_PLUGIN_HOME path.

Plugins utilize SUIT's hook script functionality to execute processes at a required lifecycle stage. The available lifecycle states are shown in table 12:

Lifecycle Hook	Description
suit_pre_init	Executes prior to the suit framework initialization
suit_post_init	Executes after suit framework initialization
testsuite_startup_pre_init	Executes prior to Test Suite initialization
testsuite_startup_post_init	Executes after Test Suite initialization

Lifecycle Hook	Description
testsuite_teardown_pre_init	Executes prior to teardown activities for a Test Suite
testsuite_teardown_post_init	Executes after teardown activities complete for a Test Suite
testcase_startup	Executes prior to Test Case execution
testcase_teardown	Executes after Test Case execution
test_startup	Executes prior to each test keyword
test_teardown	Execute after each test keyword
results_startup	Executes prior to generating results reporting
results_teardown	Executes after generating results reporting.

Table 12. SUIT Lifecycle Hooks

EXTENDING KEYWORDS.

A new keyword library can be created by calling **%suit_create_lib** and passing in the name of the keyword library to create. New keywords can be added using the **%suit_create_keyword** macro and passing in the name of the library, and the name of the file. A file can contain one or more macros and the content of the macros should follow the keyword creation conventions outlined previously in this paper.

After adding the keywords, you should maintain the keyword from the library as part of the creation process for both keywords and plugins is to store them inside version control.

CONCLUSION

SUIT provides a simple to use interface that operates over both SAS® 9 and Viya™ environments and presents interfaces that are simple, flexible and targeted to both SAS® programmers and non-programmers. SUIT does not aim to replace existing test frameworks. It is the intent to provide integration points to incorporate some of these frameworks and augment existing testing gaps in organizations.

SUIT is open source. SUIT Enterprise which provides Enterprise grade functionality will will ship around the third quarter of 2019.

To learn more about SUIT go to https://suit.seleritysas.com or feel free to contact the author.

REFERENCES

Dustin, E., Rashka, J. and Paul, J. 1999. "Automated Software Testing: Introduction, Management, and Performance" Addison-Wesley Professional.

SAS Institute Inc. 2018. SAS® 9.4 Logging: Configuration and Programming Reference, Second Edition. https://documentation.sas.com/?docsetId=logug&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en

ACKNOWLEDGMENTS

Many thanks to Michael Dixon for SAS® Programming consultation. Thanks to Melissa Jones and Jafreen Hossain for reviewing this document.

CONTACT INFORMATION

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

Cameron Lawson
Selerity
support@selerity.com.au
https://seleritysas.com