Paper TT15

SAS Unit Validation of your macros to upgrade Calibre!!

Roshan Stanly, Genpro Research, Thiruvananthapuram, India
Siji Elsa Paul, Genpro Research, Thiruvananthapuram, India

ABSTRACT

SDTM, ADaM, and TFLs submissions to the FDA require standardized programming efforts. Few of these aspects on each of these tasks remain almost similar across studies. Macros can help in avoiding repetition of the same efforts and improving efficiency, but validating these macros before using them in live studies is an ordeal. SAS unit testing has been found to be the most effective method for validating macros. SAS Unit® is a unit testing framework for SAS® macros that controls the execution of test scenarios and generates documentation in HTML format. In this paper, we will be discussing the end-to-end process for the validation of SAS macros, starting from the installation of the SAS unit to the final documentation.

INTRODUCTION

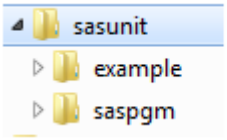
Macros helps reuse code multiple times without having to type the entire code again and again. The benefits of SAS macros are

- It saves time and results in an error-free code.
- Make our job easier by re-using similar code multiple times after defining it once.
- Make changes to a variable in a single place and reflect them at multiple locations.
- A program can be driven based on data values.

SAS Unit is a unit testing framework for SAS macros. It can be used for development, execution, and documentation of unit testing of macros. Validation of these macros is very important as it affects the quality of clinical trial data.

INSTALLATION AND SET UP

Current version of SAS unit can be downloaded from sourceforge.net/projects/sasunit/. We need to unpack the downloaded zip file at the preferred location. The folder structure should look like this after installation.



Each of these folder structures and sub-folders will be explained, and the detailed guide for setting up SAS unit software can be found in
https://sourceforge.net/p/sasunit/wiki/Getting%20started%20with%20unit%20testing%20of%20SAS%20programs/.

SAS UNIT MACROS

Once the installation is complete and folder structures are finalized as detailed in the above link, we need to write unit testing for SAS macros. We will paste the macro in the folder sasunit/<project name>/saspgm and a separate SAS program needs to be generated for testing this macro. From the below screenshot, descriptive is the macro we need to validate, and descriptive_test is the program we need to create for testing the macro. In the next steps, we will see how to develop the unit testing program for macro validation.
There are some in-built macros in SAS Unit that we need to invoke in the unit testing program. Some of these macros are:

initScenario

Start of a new test scenario. We need to give it at the beginning of the unit testing program along with a description of the macro as a macro parameter which will be used for documentation.

initTestcase

Start of a new test case of a program under test. We need to invoke the initTestcase macro before each of the unit tests and it should contain a description of that particular unit test as a macro parameter.

endTestcall
Ends an invocation of a program under test. We need to invoke this macro after the macro call for the particular unit test is done.

endTestcase
Ends a test case. The result and finish time are added to the test repository. We need to invoke this macro once a particular unit test case is done.

endScenario
This macro can be invoked at the end of the unit testing program.

Below example details the usage of the above macros.

```
%initScenario(i_desc=Tests for descriptive.sas);
/*first test case*/
%initTestcase(i_object=descriptive.sas, i_desc=Macro parameter indsn cannot be missing)
%descriptive(indsn     = ,outdsn    = out,
                                  statlist  = n mean std stderr median q1 q3 min max,
                                  var       = weight,
                                  byvars    = age sex,
                                  byvarfmts = $age. $sex.,
                                  compvar   = height);
%endTestcall()
%assertLogMsg(i_logMsg=%str(USER WARNING: descriptive Macro Aborted. At least one required parameter was not specified.))
%endTestcase();

/*second test case*/
%initTestcase(i_object=descriptive.sas, i_desc=Macro parameter outdsn cannot be missing)
%descriptive(indsn     =sashelp.class ,outdsn    = ,
                                  statlist  = n mean std stderr median q1 q3 min max,
                                  var       = weight,
                                  byvars    = age sex,
                                  byvarfmts = $age. $sex.,
                                  compvar   = height);
%endTestcall()
%assertLogMsg(i_logMsg=%str(USER WARNING: descriptive Macro Aborted. At least one required parameter was not specified.))
%endTestcase();

%endScenario;
```

In the above screenshot, we have tested the macro for only 2 test cases. The macro parameter indsn should be a required parameter and the macro is written in such a way that if the indsn parameter is specified as missing, a user warning should be populated in the SAS log and the macro should be aborted. So, we have invoked the descriptive macro by giving blank values to the indsn parameter. After the macro call, we invoked the in-built SAS Unit assert macro assertLogMsg to see if a particular message appears in the log. If it appears in the log, the unit testing will be passed or else it will fail. A similar check is done for outdsn as well. This is a simple test program that contains only 2 unit tests, and assertLogMsg is the only assert macro used. Next, we will see what the different assert macros are available in the SAS unit and its purpose.

ASSERTS IN SAS UNIT

Assert macros are used in the SAS unit for testing each scenario. In the above example, we have seen how assertLogMsg is being used to see if a particular message appears in the log. Similarly, we have other assert macros as well, which test for different scenarios like comparing two datasets, checking the value of a macro variable, checking if a dataset exists or not, etc. We will see most of these asserts in detail below.

1) **assertColumns:**

It compares the two datasets passed into i_expected and i_actual using PROC COMPARE. Check fails when there is a mismatch between both datasets.

Macro Parameters:

%assertColumns (i_expected = “SAS dataset having the expected values”
,i_actual = “SAS dataset with the actual values”
,i_desc = “Description of the assertion”
,i_fuzz = “Maximum deviation of expected values from actual values”
,i_allow = “accepted differences between data sets”
,i_id = “Id-column for matching of observations”
,o_maxReportObs = “number of observations that are copied from the expected and actual dataset”

```
,i_include      = “include list of variables to match”
,i_exclude     =” exclude list of variables to match”);
```

Example:

```
%assertColumns (i_expected=work.manual,
                i_actual=work.macro,
                i_desc=%str(Compare the estimated values),
                i_allow=LENGTH)
```

2) assertLogMsg:

Verifies that the expected messages, warnings, and errors are sent to the log file correctly. Check fails if an expected log message is not found.

Macro Parameters:

```
%assertLogMsg (i_logmsg   = “The log message expected to be printed. Special character to
                  be quoted with ‘\’ as a prefix.
               ,i_desc     = “Description of the assertion”
               ,i_not      = “if set to 1, then it negates the assertion”);
```

Example:

%assertLogMsg (i_logMsg=%str(USER WARNING: Macro Aborted, At least one required parameter was not specified.))

3) assertEquals:

Checks the difference between the value of a macro variable or output value of a macro from the expected value. This assertion fails when both i_actual and i_expected values differ.

Macro Parameters:

```
%assertEquals (i_expected = “Expected value for the macro variable”
               ,i_actual   =   “Actual value”
               ,i_desc     =   “Description of the assertion”
               ,i_fuzz     =    “Maximal deviation of expected value from actual value”);
```

Example:

```
%assertEquals (i_expected = %union(set1=one two three four,set2=one two three,unique=T),
        i_actual   =%str(one two three four),
        i_desc     = Compare Values);
```

4) assertLog:

Checks the log for errors and warnings. The check fails when the expected number of errors/warnings are not found.

Macro Parameters:

```
%assertLog (i_errors   = “Numbers of expected errors”
            ,i_warnings = “Numbers of expected warnings”
            ,i_desc     =“Description of the assertion”);
```

Example:

```
%assertLog (  i_errors   = 1,
              i_warnings = 0,
              i_desc= Scan log for errors)
```

5) assertRecordExists:

Checks the existence of records satisfying the specified conditions, or else the assertion fails.

Macro Parameters:

```
%assertRecordExists (i_dataset      = “Name of dataset to be checked”
                    ,i_whereExpr    =   “Subsetting condition”
                    ,i_desc         = “Description of the assertion” );
```

Example:

```
%assertRecordExists (i_dataset  =work.Contracts
                   ,i_whereExpr=%str(ContractType=1)
                   ,i_desc     =Do we have at least one record per contract type 1?);
```

6) assertTableExists:

Checks the existence of specified datasets, views, or catalogues.

Macro Parameters:

```
%assertTableExists (i_libref = “Name the library where the search is done”
                    ,i_memname =” Name of the item to be searched from library”
                    ,i_target  = “Explicit test for existence”
                    ,i_desc    = “Description of the assertion”
                    ,i_not     = “if set to 1, then it negates the assertion”);
```

Example:

```
%assertTableExists(i_libref  = OUT,
                   i_memname = LB,
                   i_target  = DATA,
                   i_desc    = Dataset with ZERO observations Should not exist - LB,
                   i_not     = 1);
```

7) assertReport:

Checks the existence of an expected report file. The assertion fails if the report is not created in the current SAS session. For the comparison of both reports, we require a manual check.

Macro Parameters:

```
%assertReport (i_expected = “Name of the expected file(including full path).”
                    ,i_actual =” Name of the file created manually(including full path).”
                    ,i_desc    = “Description of the assertion”
                    ,i_manual     = “1: in case of a positive check result, write an entry indicating manual     check
                   (empty rectangle).0: in case of a positive check result, write an entry indicating OK (green hook).”);
```

Example:

```
%assertReport (i_actual= &path \manual.xlsx,
                   i_expected=&path \macro.xlsx,
```

i_desc=%str(Compare both XLSX outputs))

8) assertRecordCount:

Checks whether the expected number of records exist in the specified dataset.

Macro Parameters:

%assertRecordCount (i_libref =" Name of the library which contains the required dataset"
,i_memname = "Name of the dataset to be checked"
,i_operator = "Logical operator to compare.eg., EQ"
,i_recordsExp = "Expected number of records"
,i_where = "where condition"
,i_desc = "Description of the assertion");

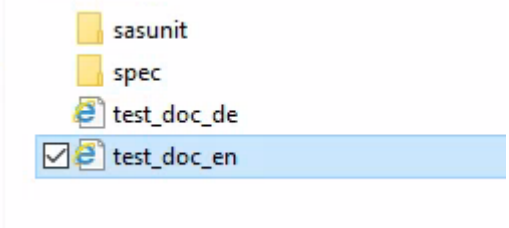
Example:

%assertRecordCount (i_libref = outmacro,
i_memname = _list,
i_recordsExp = 0,
i_desc = Check for a specific number of records);

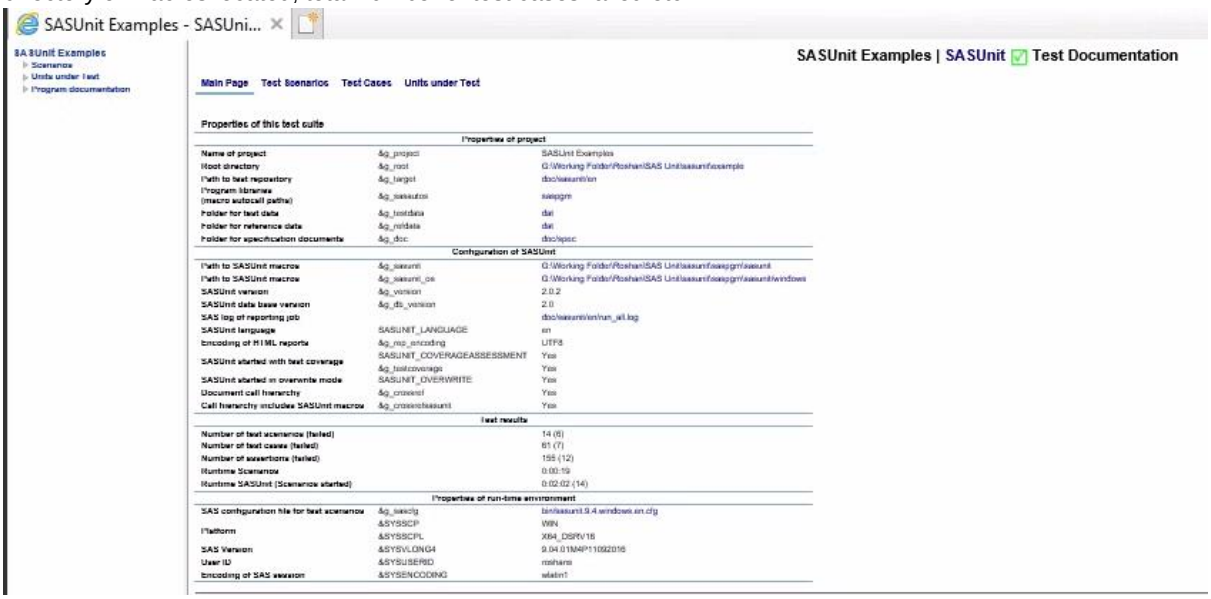
Above are the most common assert macros we might need to validate the SAS macros, but there are additional asserts as well, which can be found after downloading the SAS unit package.

DOCUMENTATION

When the unit testing program is done, we can find the results in the doc folder as an html file completing batch run.

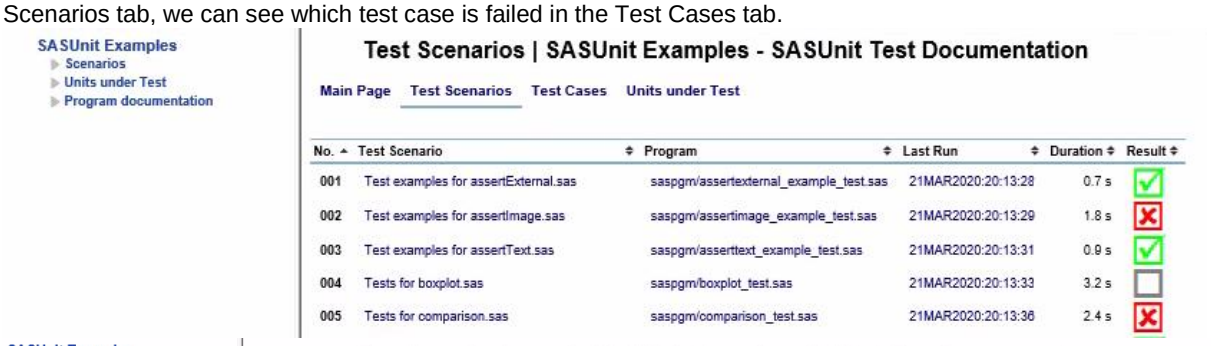


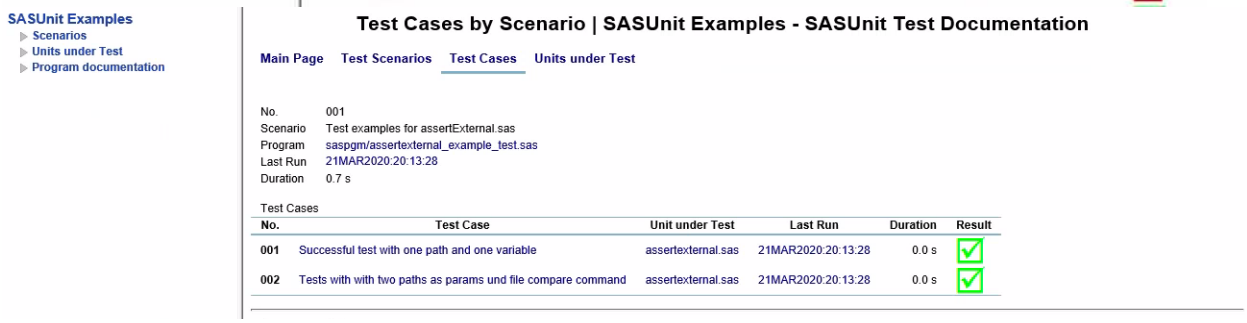
Main Page of SASUnit Test Documentation contains general information regarding the total number of macros tested, directory of macros located, total number of test cases failed etc.



Test Scenarios tab gives information regarding whether a particular macro is validation has passed or not. Test cases tab gives information on all the different checks performed inside a macro. If a particular macro is failed in Test

Scenarios tab, we can see which test case is failed in the Test Cases tab.





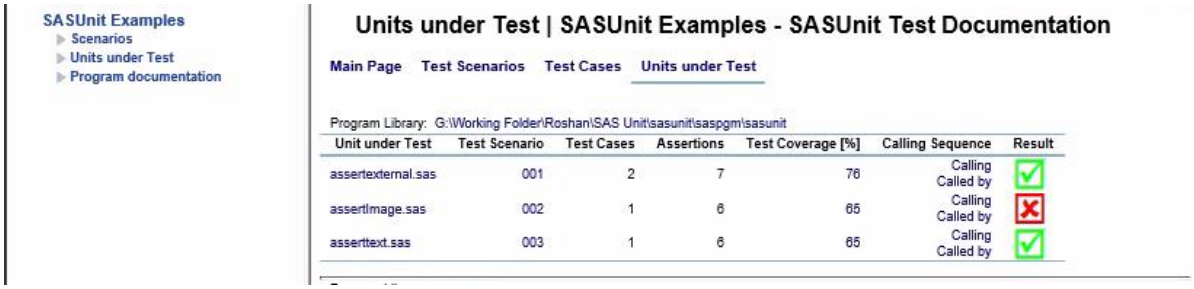
The checkbox gives the results as follows

 - test turned out correct

 - test needs a manual check

 - test result is not correct

The units under Test tab gives the overall information regarding all macros. Test Coverage refers to the actual percentage of macro code covered during validation. The non-covered codes will be highlighted in Orange and we can write additional test cases to cover those codes or remove the code if it is not necessary.



MACRO VALIDATION CHECKLIST TO EASE THE PROCESS

Sometimes, as per our conventional process, it might be convenient to create a macro validation checklist in excel format to track all the checks for a macro. This might be more comfortable to ensure that all the checks have been added for a macro and visually appealing as well for review compared to SAS programs. In this section, we explain how we can create a macro validation checklist to add all the checks for a macro in an excel file and a program we have created to generate the sas unit testing program from the macro validation checklist. The structure of macro validation checklist is given below.

S. No	Testcase	Macro Check	Expected Result
1	* Macro parameter indsn cannot be missing	%count(indsn =, outdsn = out1, var = age);	Log message - "USER WARNING: MACRO ABORTED!!! THE MACRO PARAMETER INDSN IS NOT SPECIFIED!!!".
2	* Macro parameter outdsn cannot be missing	%count(indsn = sashelp.cars, outdsn = , var = age);	Log message - "USER WARNING: MACRO ABORTED!!! THE MACRO PARAMETER OUTDSN IS NOT SPECIFIED!!!".
3	* One existing dataset is passed as macro parameter indsn * Input dataset contains 0 record.	%count(indsn = adae0_count, outdsn = out, var = age);	Log message - "USER WARNING: COUNT Macro Aborted, The dataset (adae0_count) has zero observations.".
4	*One existing dataset is passed as macro parameter indsn.	%count(indsn = adae_count, outdsn = out, var = age);	Derive the dataset in a separate program and compare both the datasets

S. No -> Serial number for each test case.
Testcase -> Explanation of the check we are performing for a particular test.
Macro Check -> Macro call corresponding to the test we are performing.
Expected Result -> Expected result for a particular test. Should follow standard convention as defined in the chklisttosas macro to perform appropriately.

The chklisttosas macro can be called to produce the initial version of the SASUnit test program once the macro validation checklist has been created as we saw above. From there, we may only need to add a few more stages to the program. The chklisttosas macro was used to create an SASUnit test program from the preceding checklist, which is pasted below.

```
Program Name:                count_sasunit_test
SAS Version:                 SAS V9.4
Short Description:           Program to validate the reporting macro count.
Author:                      <Author>
Date:                        <Date of development>
Input:                       <Add input datasets here>
Output:                      HTML file for sas unit test documentation

*************************************************************************************************************************************

%initScenario(i_desc=Tests for count.sas);
/*Check 1*/
%initTestcase(i_object=count.sas, i_desc=%str(Macro parameter indsn cannot be missing))
%count(indsn    =,
       outdsn   = out1,
       var      = age);
%endTestcall()
%assertLogMsg(i_logMsg=%str(USER WARNING: MACRO ABORTED!!! THE MACRO PARAMETER INDSN IS NOT SPECIFIED!!!))
%endTestcase();

/*Check 2*/
%initTestcase(i_object=count.sas, i_desc=%str(Macro parameter outdsn cannot be missing))
%count(indsn    = sashelp.cars,
       outdsn   =,
       var      = age);
%endTestcall()
%assertLogMsg(i_logMsg=%str(USER WARNING: MACRO ABORTED!!! THE MACRO PARAMETER OUTDSN IS NOT SPECIFIED!!!))
%endTestcase();

/*Check 3*/
%initTestcase(i_object=count.sas, i_desc=%str(One existing dataset is passed as macro parameter indsn,Input dataset contains 0
              record.))
%count(indsn    = adae0_count,
       outdsn   = out,
       var      = age);
%endTestcall()
%assertLogMsg(i_logMsg=%str(USER WARNING: COUNT Macro Aborted, The dataset \(adae0_count\) has zero observations. .))
%assertLogMsg(i_logMsg=%str(USER WARNING: COUNT Macro Aborted, The dataset \(adae_count\) has zero observations.))
%endTestcase();

/*Check 4*/
%initTestcase(i_object=count.sas, i_desc=%str(One existing dataset is passed as macro parameter indsn.))
/*<DERIVE THE DATASET MANUALLY HERE>*/
%count(indsn    = adae_count,
       outdsn   = out,
       var      = age);
%endTestcall()
%assertColumns(i_actual= , i_expected= , i_desc=%str(One existing dataset is passed as macro parameter indsn.),i_allow=LENGTH)
%endTestcase();

%endScenario;
```

Only Check 4 in the SASUnit test program mentioned above has to be updated in order to manually build the dataset and compare it to the output produced by the macro. Please be aware that the chklisttosas macro was designed to handle the assertions assertLogmsg and assertColumns just for the time being. It can be improved to handle other asserts. The chklisttosas macro is pasted below.

```
%macro chklisttosas(macroname=,in_file=);

/*Importing checklist file into sas dataset*/
 %let file1 = %kscan(%sysfunc(kcompress(&in_file,"")),-1,%str(.));
 %if &file1 = %klowcase(xlsx) or &file1 =  %klowcase(xlsm)%then %do; %let file=xlsx ;%end;
    %else  %let file=&file1;
    %if  &file1 = %klowcase(xls) or &file1 = %klowcase(xlsx) %then %do;
filename infile &in_file;
        proc import datafile=infile
                out=list1
                dbms=&file
                replace;
                getnames=yes;
       run;
    %end;
%let nobs = &sysnobs.;

/*Handling when multiple asserts within single testcase*/
data list1a;
  set list1;
  length exp1 exp2 $10000.;
  exp1 = tranwrd(expected_result,"!","@#@");
  exp2 =translate(exp1,'!','0A'x,'0D'x);
  var = countw(exp2,"!");
  call symput(cats("n",strip(put(_n_,best.))),strip(put(_n_,best.)));
  call symput(cats("v",strip(put(_n_,best.))),strip(put(var,best.)));
run;

/*Appropriate Assert macros being called, can be updated if more macros required.
Structing the test Program with comments and statements*/
data list2(keep=progline);
  set list1a;
  array dumvar[50] $10000;
  array dvar[50] $10000;
  array resvar[50] $10000.;
  length progline result2 $10000.;
  logic1 = tranwrd(substr(strip(testcase),2),"?",",");
  logic1= compbl(compress(logic1,'0D0A'x));

  %do i = 1 %to &nobs.;

  if _N_ eq &&n&i. then do;
    %if &&v&i. gt 1 %then %do;
  do i = 1 to &&v&i.;

    dumvar[i] = strip(substr(scan(exp2,i,"!"),3));
if substr(strip(dumvar[i]),length(strip(dumvar[i])),1) eq "." then resvar[i] =
substr(strip(dumvar[i]),1,length(strip(dumvar[i]))-1);
        else resvar[i] = dumvar[i];
        if find(resvar[i],"log message","i") then do;
          dvar[i] = substr(resvar[i],find(resvar[i],"log message","i")+14);
       dvar[i] = tranwrd(dvar[i],"","");
dvar[i] = cats('%assertLogMsg(i_logMsg=%str(',tranwrd(tranwrd(tranwrd
(tranwrd(dvar[i],"\","\\"),"(","\("),")","\)"),"^","\^"),'))');
            end;
```

```
else if find(resvar[i],"Derive the dataset in a separate program and compare both the datasets","i") then do;
dvar[i] = cats('%assertColumns(i_actual= , i_expected= ,i_desc=%str(',strip(logic1),'), i_allow=LENGTH)' );
         end;
         else do;
           dvar[i] = resvar[i];
         end;
    end;
    result2 = catx('0A'x,of dvar:);
    %end;
    %else %do;
 if substr(strip(expected_result),length(strip(expected_result)),1) eq "." then result1 =
substr(strip(expected_result),1,length(strip(expected_result))-1);
        else result1 = expected_result;
        result1= compbl(compress(result1,'0D0A'x));
        if find(result1,"log message","i") then do;
          result2 = substr(result1,find(result1,"log message","i")+14);
   result2 = tranwrd(result2,"","");
result2 = cats('%assertLogMsg(i_logMsg=%str(',tranwrd(tranwrd(tranwrd
(tranwrd(result2,"\","\\"),"(","\("),")","\)"),"^","\^"),'))');
      end;
 else if find(result1,"Derive the dataset in a separate program and compare both the datasets","i") then do;
result2 = cats('%assertColumns(i_actual= , i_expected= , i_desc=%str(',strip(logic1),'), i_allow=LENGTH)' );
      end;
      else do;
        result2 = result1;
      end;
%end;
  end;
  %end;
  result2 = tranwrd(result2,"@#@","!");

  progline = cat("/*Check ",cats(_n_),"*/");
  output;
  progline = cat('%initTestcase(i_object=',"&macroname..sas, ",'i_desc=%str(',strip(logic1),'))');
  output;
if find(expected_result,"Derive the dataset in a separate program and compare both the datasets") then do;
    progline = "/*<DERIVE THE DATASET MANUALLY HERE>*/";
    output;
    progline = strip(macro_check);
    output;
  end;
  else do;
    progline = strip(macro_check);
    output;
  end;
  progline = '%endTestcall()';
  output;
  progline = strip(result2);
  output;
  progline = '%endTestcase();';
  output;
  progline = "";
  output;
run;

/*Header part for the sasunit tes program*/
data list3;
  length progline $10000.;
  progline =
"/*************************************************************************************************************************";
```

```
  output;
  Progline = "Program Name:               &macroname._sasunit_test";
  output;
  Progline = "SAS Version:               SAS V9.4";
  output;
  Progline = "Short Description:          Program to validate the reporting macro &macroname..";
  output;
  Progline = "Author:                    <Author>";
  output;
  Progline = "Date:                      <Date of development>";
  output;
  Progline = "Input:                     <Add input datasets here>";
  output;
  Progline = "Output:                    HTML file for sas unit test documentation";
  output;
  Progline = "";
  output;
  Progline =
"*****************************************************************************************************************************/";
  output;
  Progline = "";
  output;
  progline = cat('%initScenario(i_desc=Tests for ',"&macroname..sas);");
  output;
run;

data list4;
  length progline $10000.;
  progline = '%endScenario;';
  output;
run;

data list5;
  set list3 list2 list4;
run;

/*Path and file name of the sasunit program*/
filename sascode "&path.\&macroname._sasunit_test.sas";

data _null_;
  set list5;
  file sascode;
  put progline;
run;

%mend chklisttosas;
```

CONCLUSION

Macros can help reduce programming effort and improve efficiency and quality. It is important to validate these macros properly before using them on live studies as we are dealing with clinical data and the analysis results should be accurate. SAS Unit is a great platform to validate these macros as the way the tool is designed for validation and documentation is quite fascinating. Proper validation of macros saves a huge time and effort while working on real projects.

REFERENCES

1. Details of SASUnit documentation can be found in https://sourceforge.net/p/sasunit/wiki/Getting%20started%20with%20unit%20testing%20of%20SAS%20programs/.
2. Chklisttosas program is also available in https://github.com/roshanstanly22/SAS-Macro/blob/main/chklisttosas.sas.

CONTACT INFORMATION

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

Author Name : Roshan Stanly
Company : Genpro Research Inc
Address : Second Floor, Nila Building
Technopark Campus
Thiruvananthapuram, Kerala, India
Pin : 695581
Work Phone : +91.4714.026.700
Email : roshan.stanly@genproresearch.com
Website : https://genproresearch.com/

Author Name : Siji Elsa Paul
Company : Genpro Research Inc
Address : Second Floor, Nila Building
Technopark Campus
Thiruvananthapuram, Kerala, India
Pin : 695581
Work Phone : +91.4714.026.700
Email : siji.paul@genproresearch.com