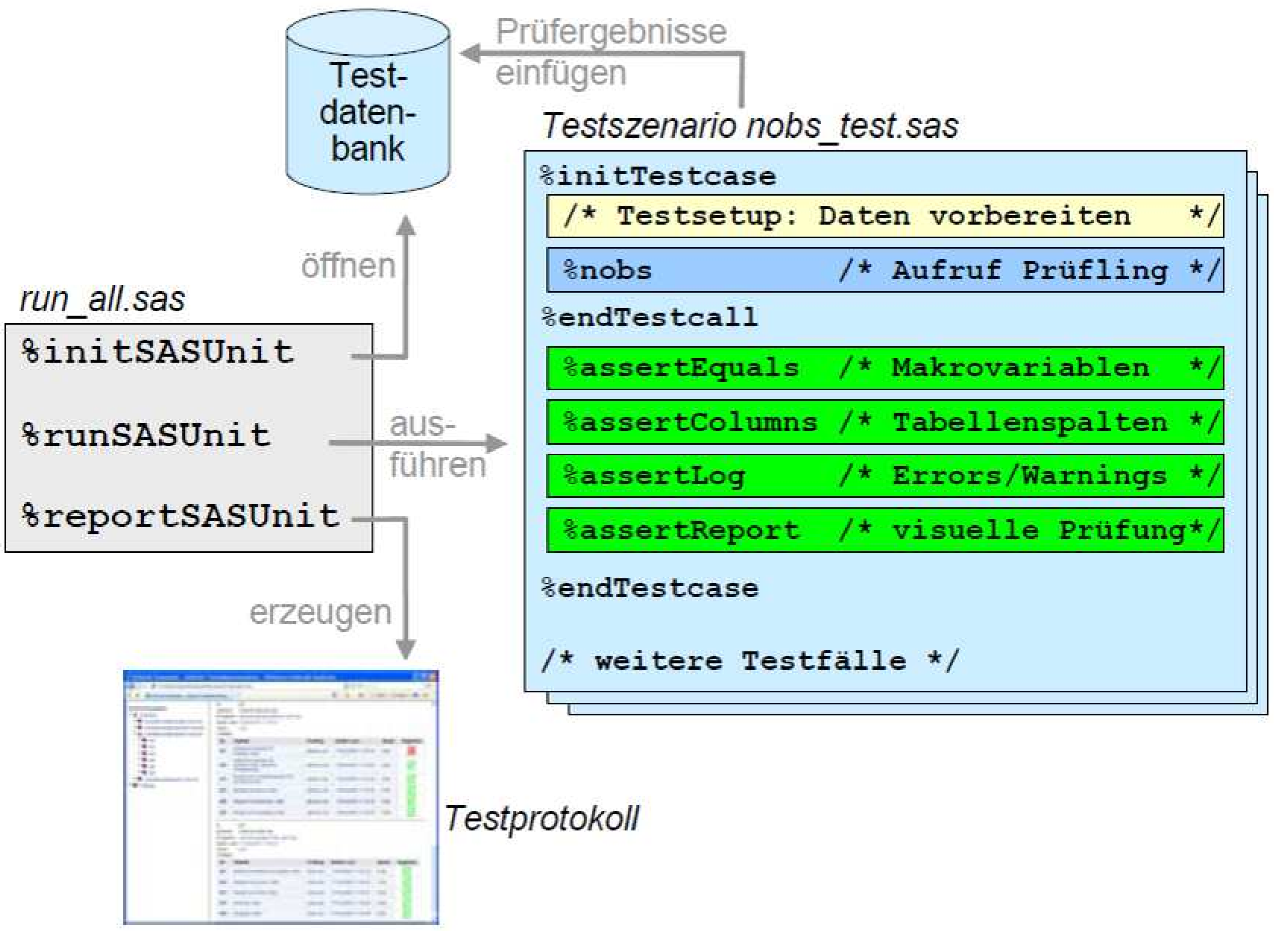
1 How SASUnit 2.0 works

In the following both working methods (batch and interactive) with SASUnit are introduced.

1.1 SASUnit - Batch (as before)

The strength of SASUnit as a batch call is the documentation of the entire test-suite and the automated repetition of tests. In batch calls, the runtime is less important, because this is where the test results are documented and you expect a longer run time.
In some areas, SAS® Display Manager is used to create SAS® programs. Thus, the developer works directly on the environment on which SASUnit runs and generates the documentation. This is where SASUnit can show its full strength.
This has also been the goal in the development of SASUnit.



```
%initSASUnit

%runSASUnit

%reportSASUnit
```

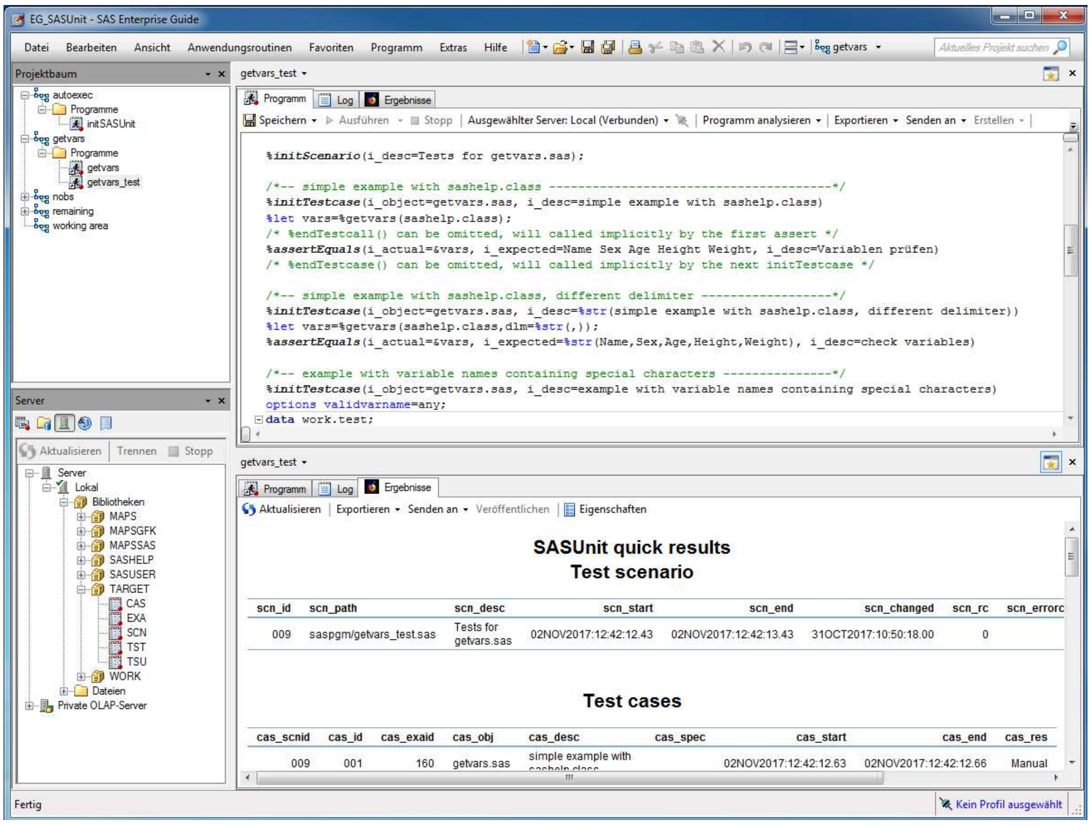
```
%initTestcase
  /* Testsetup: Daten vorbereiten   */
  %nobs            /* Aufruf Prüfling */
%endTestcall
  %assertEquals    /* Makrovariablen  */
  %assertColumns   /* Tabellenspalten */
  %assertLog       /* Errors/Warnings */
  %assertReport    /* visuelle Prüfung*/
%endTestcase

/* weitere Testfälle */
```

Figure 1: SASUnit Architecture (Batch)

1.2 SASUnit - interactive

However, some SAS environments no longer have access to the Display Manager and use a remote SAS session. Up to now there was no adequate possibility to use SASUnit for this field of application.
Starting a batch SAS® session on a remote computer is somewhat more complicated with SAS®/BASE means. This can be achieved, for example, by a stored process that executes a command file on the remote host.
The results must also first be copied from the remote session to the local client.
Why not leave that to the SAS® Enterprise Guide. He can do it already.

When creating a test scenario, it is more important to quickly obtain the results of the process. Test coverage or program documentation is less important. These can also be generated in a batch call if the scenario is fully implemented.
With a fast interactive mode SASUnit can also be used for test-driven development / design.
This was the starting signal for the interactive mode of SASUnit.
The following solution is a first step into a new direction. There is certainly still a lot to expand and/or improve here. For this we want to use the feedback of the community, in order not to miss the need in developement.


EG_SASUnit - SAS Enterprise Guide
Datei Bearbeiten Ansicht Anwendungsroutinen Favoriten Programm Extras Hilfe
getvars
Aktuelles Projekt suchen
Projektbaum
autoexec
Programme
initSASUnit
getvars
Programme
getvars
getvars_test
nobs
remaining
working area
Server
Aktualisieren Trennen Stopp
Server
Lokal
Bibliotheken
MAPS
MAPSGFK
MAPSSAS
SASHELP
SASUSER
TARGET
CAS
EXA
SCN
TST
TSU
WORK
Dateien
Private OLAP-Server
getvars_test
Programm Log Ergebnisse
Speichern Ausführen Stopp Ausgewählter Server: Local (Verbunden) Programm analysieren Exportieren Senden an Erstellen
Aktualisieren Exportieren Senden an Veröffentlichen Eigenschaften
Fertig
Kein Profil ausgewählt

```
%initScenario(i_desc=Tests for getvars.sas);

/*-- simple example with sashelp.class --------------------------------------*/
%initTestcase(i_object=getvars.sas, i_desc=simple example with sashelp.class)
%let vars=%getvars(sashelp.class);
/* %endTestcall() can be omitted, will called implicitly by the first assert */
%assertEquals(i_actual=&vars, i_expected=Name Sex Age Height Weight, i_desc=Variablen prüfen)
/* %endTestcase() can be omitted, will called implicitly by the next initTestcase */

/*-- simple example with sashelp.class, different delimiter -----------------*/
%initTestcase(i_object=getvars.sas, i_desc=%str(simple example with sashelp.class, different delimiter))
%let vars=%getvars(sashelp.class,dlm=%str(,));
%assertEquals(i_actual=&vars, i_expected=%str(Name,Sex,Age,Height,Weight), i_desc=check variables)

/*-- example with variable names containing special characters --------------*/
%initTestcase(i_object=getvars.sas, i_desc=example with variable names containing special characters)
options validvarname=any;
data work.test;
```

SASUnit quick results
Test scenario

scn_id	scn_path	scn_desc	scn_start	scn_end	scn_changed	scn_rc	scn_errorc
009	saspgm/getvars_test.sas	Tests for getvars.sas	02NOV2017:12:42:12.43	02NOV2017:12:42:13.43	31OCT2017:10:50:18.00	0	

Test cases

cas_scnid	cas_id	cas_exaid	cas_obj	cas_desc	cas_spec	cas_start	cas_end	cas_res
009	001	160	getvars.sas	simple example with		02NOV2017:12:42:12.63	02NOV2017:12:42:12.66	Manual

Figure 2: SASUnit Architecture (Interactive)

2 SASUnit 2.0 - New interactive mode

This section explains the approach to the interactive mode of SASUnit.

2.1 Goal

The goal is to write **and execute** a test scenario in the Enterprise Guide. A compact report on the success or failure of the test cases should also be available. This makes it possible to use SASUnit in the TDD (Test-driven Development / Design) environment.

2.2 Architecture / Approach

Not much has changed in the basic architecture. The test database is the same and you still need to call initSASUnit. However, this call only executes part of the functionality. The checks for the existence of the autocall paths are still performed. However, since the log and report are returned directly to the client, for example, the directories for logs and reports are not deleted.
At present, the same SASUnit test database is used for the interactive call as in the batch call. Thus, the test database always reflects the current state of development.

Since each test scenario now reads its own scenario ID from the test database, it must be adapted for the interactive call. (For details see 3.3)
At the beginning, a call to initScenario must be inserted.

```
/**
   \file
   \ingroup    SASUNIT_EXAMPLES_TEST

   \brief      Tests for nobs.sas - has to fail!

               Example for a test scenario with these features:
               - create simple test scenario
               - check value of macro symbol with assertEquals.sas

*/ /** \cond */

%initScenario(i_desc=Tests for nobs.sas - has to fail!);

/*-- simple example with sashelp.class -----------------------*/
%initTestcase(i_object=nobs.sas
             ,i_desc=simple example with sashelp.class
             )

%let nobs=%nobs(sashelp.class);

%endTestcall()

%assertEquals(i_actual=&nobs
             ,i_expected=19
             ,i_desc=number of observations in sashelp.class
```

```
            )
%assertLogMsg (i_logMsg=.let nobs=.nobs.sashelp.class.);
%assertLogMsg (i_logMsg=NOTE: .* NOBS );
%endTestcase()
```

At the end a short report should be generated. This task is performed by the macro ***endScenario***.

```
/*-- invalid dataset -------------------------------------------*/
%initTestcase(i_object=nobs.sas, i_desc=%str(invalid dataset))

%let nobs=%nobs(xxx);

%endTestcall()

%assertEquals(i_actual=&nobs
             ,i_expected=
             ,i_desc=number of observations with invalid dataset
             )

%endTestcase()

proc datasets lib=work memtype=DATA nolist;
   delete big empty;
run;quit;

%endScenario();
/** \endcond */
```

The focus of the interactive call is to support test-driven development/design. The execution should be fast and therefore limited to the minimum functionality required during implementation.

For the creation of the complete test documentation, the test coverage, the program documentation and the complete testing of all test examinees the batch call is still indispensable!

With the current architecture, the interactive call should also be executable in environments without XCMD. With the exception of the checks that use operating system commands (***assertExternal***, ***assertImage*** and ***assertText***).

2.3 General conditions and implications

Since there is no longer a main program that controls the call and passes information, each scenario must know what it is called (name of the SAS program that is currently running) to update the test database correctly. This information was previously transferred from the calling batch session and has now been moved to the current test scenario.

These include:

- the name of the called test scenario
- the scenario ID from the test database

Depending on how the test scenario is called (batch or interactive), this information is stored in different places in the SAS session. The knowledge about this is encapsulated in a new macro _readEnvMetadata.

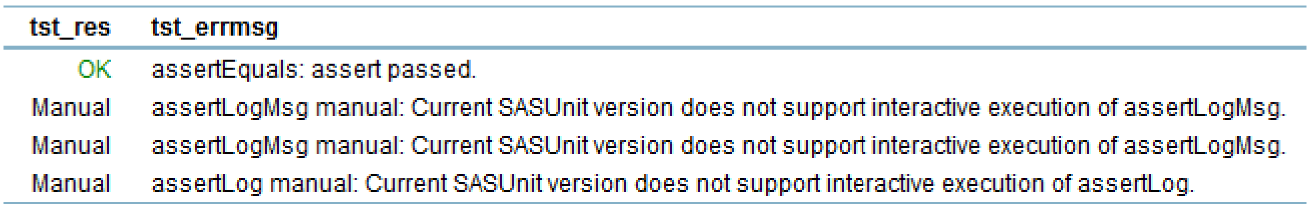
Due to the architectural changes and the type of call in the Enterprise Guide, note the following:

With the interactive call there is only one(!) SAS session.

- All scenarios share the work library.
    - Since the temporary tables remain at the end of the scenario, undesirable side effects may occur. Therefore, it is necessary to clean up at the end of the scenario.
- All scenarios share the runtime environment and thus also the macro variables.
    - Here, too, there can be side effects between the scenarios.
    - More importantly, however, there may be side effects even if a scenario is called repeatedly. It is necessary to include all macro variables used in the test setup.

After calling a scenario in the Enterprise Guide, the complete log should be available there.

- Splitting the log into individual sections by PROC PRINTTO is no longer displayed.
- The log is streamed back into the Enterprise Guide. This log is not evaluated in the current version. This means that the two macros ***assertLog*** and ***assertLogMsg*** cannot work as usual. Both macros take this into account and set the test result to manual and write a suitable message in the test database:



tst_res	tst_errmsg
OK	assertEquals: assert passed.
Manual	assertLogMsg manual: Current SASUnit version does not support interactive execution of assertLogMsg.
Manual	assertLogMsg manual: Current SASUnit version does not support interactive execution of assertLogMsg.
Manual	assertLog manual: Current SASUnit version does not support interactive execution of assertLog.

Figure 3: Message (interactive) of assertLog and assertMsg

Due to the limitations of some SAS environments, support for NOXCMD environments is also implemented.
However, as already mentioned, this means that the external checks cannot work.

This applies to the following macros:
- assertExternal
- assertImage
- assertText

If one of these macros is used in an NOXCMD environment, it sets the check result to manual and writes a corresponding message to the test database.

```
%IF %_handleError(&l_macname.
                 ,NOXCMD
                 ,(%sysfunc(getoption(XCMD)) = NOXCMD)
                 ,Your SAS Session does not allow XCMD%str(,)
                  therefore assertExternal cannot be run.
                 ,i_verbose=&g_verbose.
                 )
```

2.4 Supported application scenarios

The interactive mode is designed to develop the test scenarios faster and according to TDD methodology.
Below are the application scenarios we have in mind.

2.4.1 Call scenarios

Use with a local SAS session in the Enterprise Guide works best. This is where you have the right to issue operating system commands and SASUnit is the easiest to do.

But you can also work with a remote session.
Here you can differentiate between several variants depending on the location of the test scenarios and the SASUnit macros.

Testszenarien	SASUnit Makros
Lokal	Lokal
Lokal	Remote
Remote	Remote

Table 1: SASUnit interactive call scenarios

All scenarios are possible with SASUnit 2.0.

2.4.2 Clients for SASUnit 2.0

The preferred client for SASUnit 2.0 is clearly the SAS® Enterprise Guide. It is used for development and testing at HMS.

SASUnit 2.0 also runs on SAS® Studio and Jupyter notebooks.

Jupyter Notebooks had to be adapted.

Due to the type of SAS® call by Jupyter Notebooks, it is not clear in the SAS® session which program is currently being executed. Thus the information for updating the test database is missing. To make up for this shortcoming, there is an additional parameter ***i_object*** for the macro ***initScenario***.

```
%MACRO initScenario(i_object =_AUTOMATIC_
                   ,i_desc   =_NONE_
                   );

   %global g_inScenario g_inTestCase g_inTestCall g_scnID;
   %local l_scenarioPath;
```

Thus it is also possible to use SASUnit 2.0 with Jupyter notebooks.

2.5 Lack of backward compatibility

SASUnit 2.0 carefully carries this version number. The previous version is 1.7. To avoid making SASUnit unnecessarily complicated, both call variants (batch and interactive) use the same logic.

Starting with version 2.0, the Id scenario is also determined using ***initScenario*** in batch mode.
In individual cases it may be necessary to adjust a test scenario so that it is still working in batch mode.

When ***initTestcase*** is called, the system checks whether ***initSenario*** has already run. If not, it is called. So there is no compulsion to adapt all scenarios.

If SASUnit macro variables are accessed in a scenario during the test setup, as is the case with some of our test scenarios, then you must explicitly **insert the *initScenario* call before** (!).

```
/**
   \file
   \ingroup    SASUNIT_TEST

   \brief      Test of assertTableExists.sas

*/ /** \cond */

%initScenario(i_desc =Test of assertTableExists.sas);

%let scnid = %substr(00&g_scnid, %length(&g_scnid));

/* test case 1 ------------------------------------ */
```

```
libname hugo1 "X:/TEST";

%initTestcase(
             i_object=assertTableExists.sas
            ,i_desc=call with invalid library
)
```

For the batch call of SASUnit 2.0, nothing more needs to be adapted to the scenarios.

3 Suggested way of working in interactive mode

The Enterprise Guide is the preferred client for SASUnit 2.0, which is why the approach suggested by HMS is based on it.
SASUnit 2.0 provides a working Enterprise Guide project (with local and remote SAS session) for the example project. It shows how to use the Enterprise Guide and SASUnit 2.0.

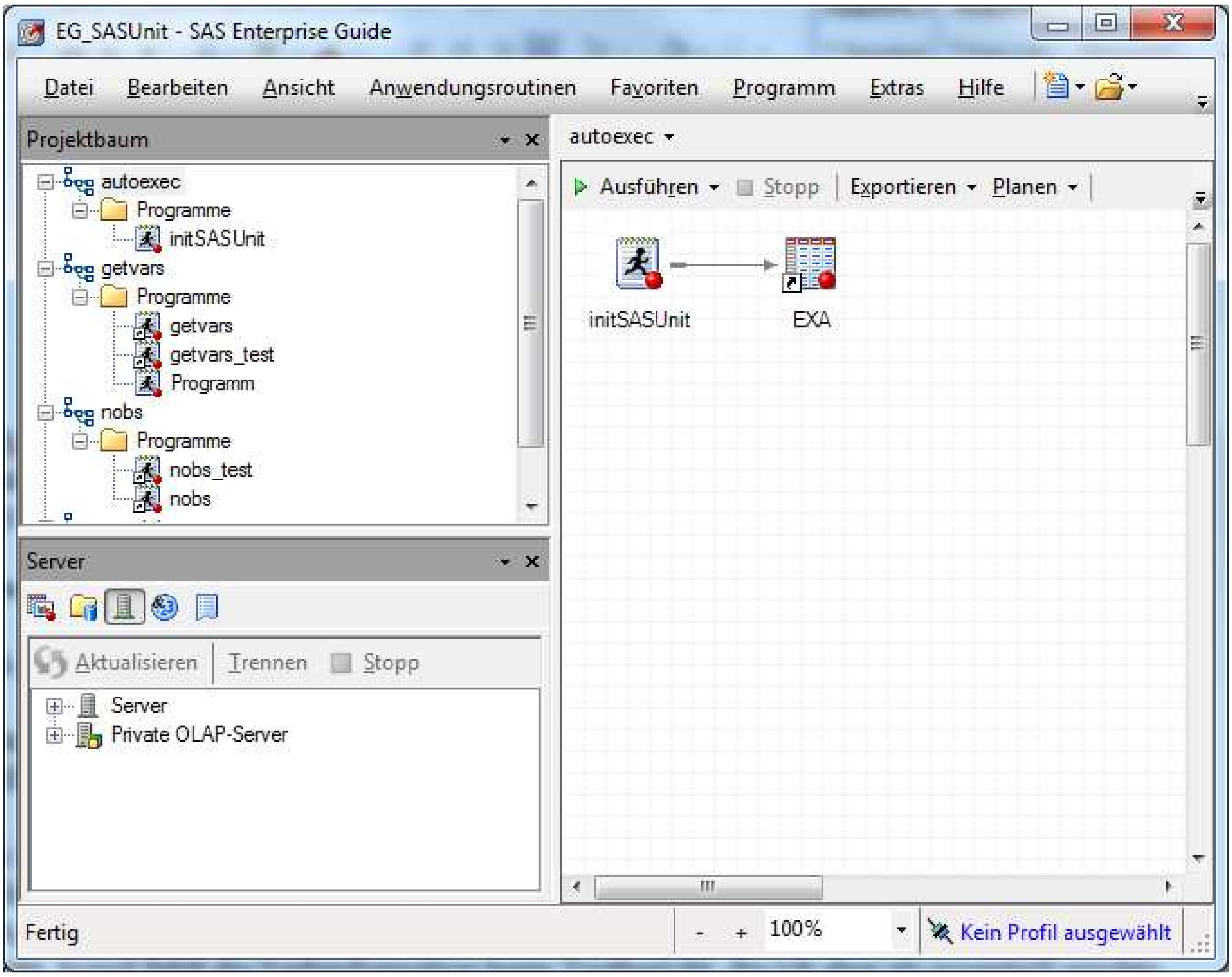


Figure 4: Mode of operation in the Enterprise Guide process flow autoexec

In the Autoexec process flow, initSASUnit is called automatically.

The remaining process flows serve to structure the project and are arbitrarily selected here.

It is possible that the standard reporting format of the Enterprise Guide (SAS Report) does not render inline formatting. This is due to the combination of SAS Enterprise Guide and SAS version in the background. For example, SAS 9.4 Maintenance Release 2 does not work. Thus the color information is missing in the test report, which I consider essential. To get colored reports you have to change the result of the scenario to HTML once - then it works with the test report.

But with SAS 9.4 and Maintenance Release 5 it works perfectly.

3.1 Working with the process flow working area

The process flow working area plays a special role. It is used to contain the macro currently being worked on, including the test scenario. Thus, the test call can easily be triggered on the process flow with F3. Alternatively, you can also use the context menu with the right mouse button on the macro "Execute branch from...".

After the implementation is finished, you can move the entire branch to another process flow and then have the working area free for the implementation of the next feature.

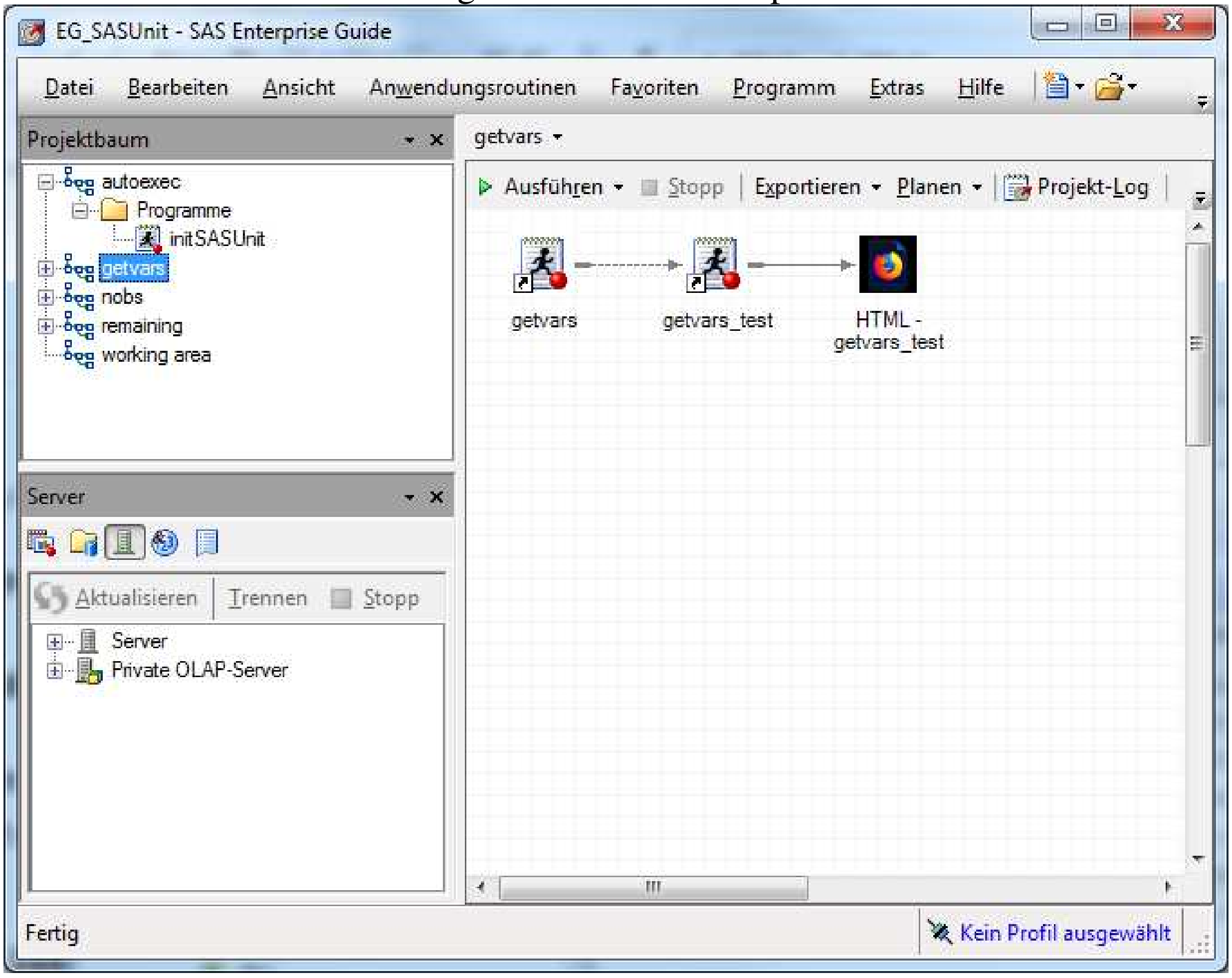
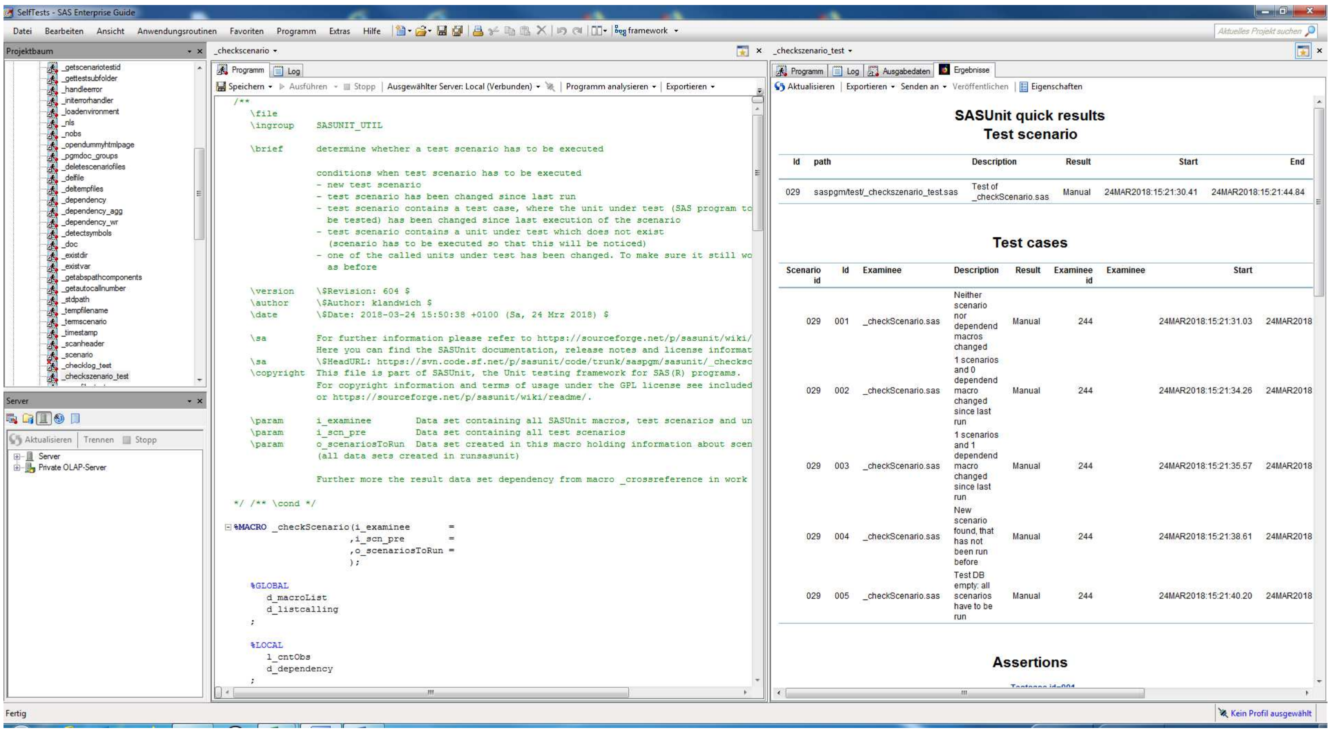

EG_SASUnit - SAS Enterprise Guide
Datei Bearbeiten Ansicht Anwendungsroutinen Favoriten Programm Extras Hilfe
Projektbaum
autoexec
Programme
initSASUnit
getvars
nobs
remaining
working area
Server
Aktualisieren Trennen Stopp
Server
Private OLAP-Server
getvars
Ausführen Stopp Exportieren Planen Projekt-Log
getvars
getvars_test
HTML - getvars_test
Fertig
Kein Profil ausgewählt

Figure 5: Mode of operation in the Enterprise Guide process flow working area

3.2 Working with parallel windows

Another way to work with the Enterprise Guide is to open the candidate and the scenario in two parallel windows. This allows you to edit the examinee and simply update the results of the scenario.
This allows a more fluent work.



```
/**
   \file
   \ingroup    SASUNIT_UTIL

   \brief      determine whether a test scenario has to be executed

               conditions when test scenario has to be executed
               - new test scenario
               - test scenario has been changed since last run
               - test scenario contains a test case, where the unit under test (SAS program to
                 be tested) has been changed since last execution of the scenario
               - test scenario contains a unit under test which does not exist
                 (scenario has to be executed so that this will be noticed)
               - one of the called units under test has been changed. To make sure it still wo
                 as before

   \version    \$Revision: 604 $
   \author     \$Author: klandwich $
   \date       \$Date: 2018-03-24 15:50:38 +0100 (Sa, 24 Mrz 2018) $

   \sa         For further information please refer to https://sourceforge.net/p/sasunit/wiki/
               Here you can find the SASUnit documentation, release notes and license informat
   \sa         \$HeadURL: https://svn.code.sf.net/p/sasunit/code/trunk/saspgm/sasunit/_checksc
   \copyright  This file is part of SASUnit, the Unit testing framework for SAS(R) programs.
               For copyright information and terms of usage under the GPL license see included
               or https://sourceforge.net/p/sasunit/wiki/readme/.

   \param      i_examinee        Data set containing all SASUnit macros, test scenarios and un
   \param      i_scn_pre         Data set containing all test scenarios
   \param      o_scenariosToRun  Data set created in this macro holding information about scen
               (all data sets created in runsasunit)

               Further more the result data set dependency from macro _crossreference in work

*/ /** \cond */

%MACRO _checkScenario(i_examinee       =
                     ,i_scn_pre        =
                     ,o_scenariosToRun =
                     );

   %GLOBAL
      d_macroList
      d_listcalling
   ;

   %LOCAL
      l_cntObs
      d_dependency
   ;
```

SASUnit quick results
Test scenario

Id	path	Description	Result	Start	End
029	saspgm/test/_checkszenario_test.sas	Test of _checkScenario.sas	Manual	24MAR2018:15:21:30.41	24MAR2018:15:21:44.84

Test cases

Scenario id	Id	Examinee	Description	Result	Examinee id	Examinee	Start	
029	001	_checkScenario.sas	Neither scenario nor dependend macros changed	Manual	244		24MAR2018:15:21:31.03	24MAR2018
029	002	_checkScenario.sas	1 scenarios and 0 dependend macro changed since last run	Manual	244		24MAR2018:15:21:34.26	24MAR2018
029	003	_checkScenario.sas	1 scenarios and 1 dependend macro changed since last run	Manual	244		24MAR2018:15:21:35.57	24MAR2018
029	004	_checkScenario.sas	New scenario found, that has not been run before	Manual	244		24MAR2018:15:21:38.61	24MAR2018
029	005	_checkScenario.sas	Test DB empty: all scenarios have to be run	Manual	244		24MAR2018:15:21:40.20	24MAR2018

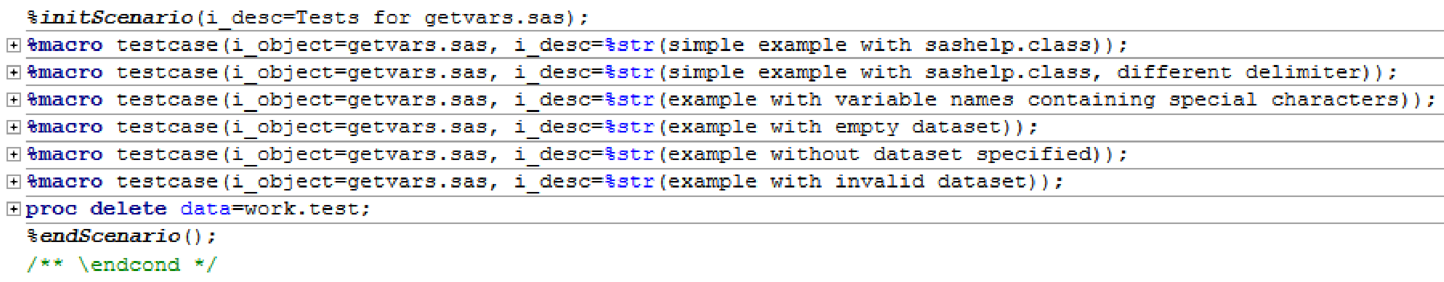
Assertions

SelfTests - SAS Enterprise Guide
Datei Bearbeiten Ansicht Anwendungsroutinen Favoriten Programm Extras Hilfe
framework
Aktuelles Projekt suchen
Projektbaum
_checkscenario
Programm Log
Speichern Ausführen Stopp Ausgewählter Server: Local (Verbunden) Programm analysieren Exportieren
_checkszenario_test
Programm Log Ausgabedaten Ergebnisse
Aktualisieren Exportieren Senden an Veröffentlichen Eigenschaften
Server
Aktualisieren Trennen Stopp
Server
Private OLAP-Server
Fertig
Kein Profil ausgewählt

Figure 6: Mode of operation in the Enterprise Guide with parallel windows

3.2 Better overview in the scenario

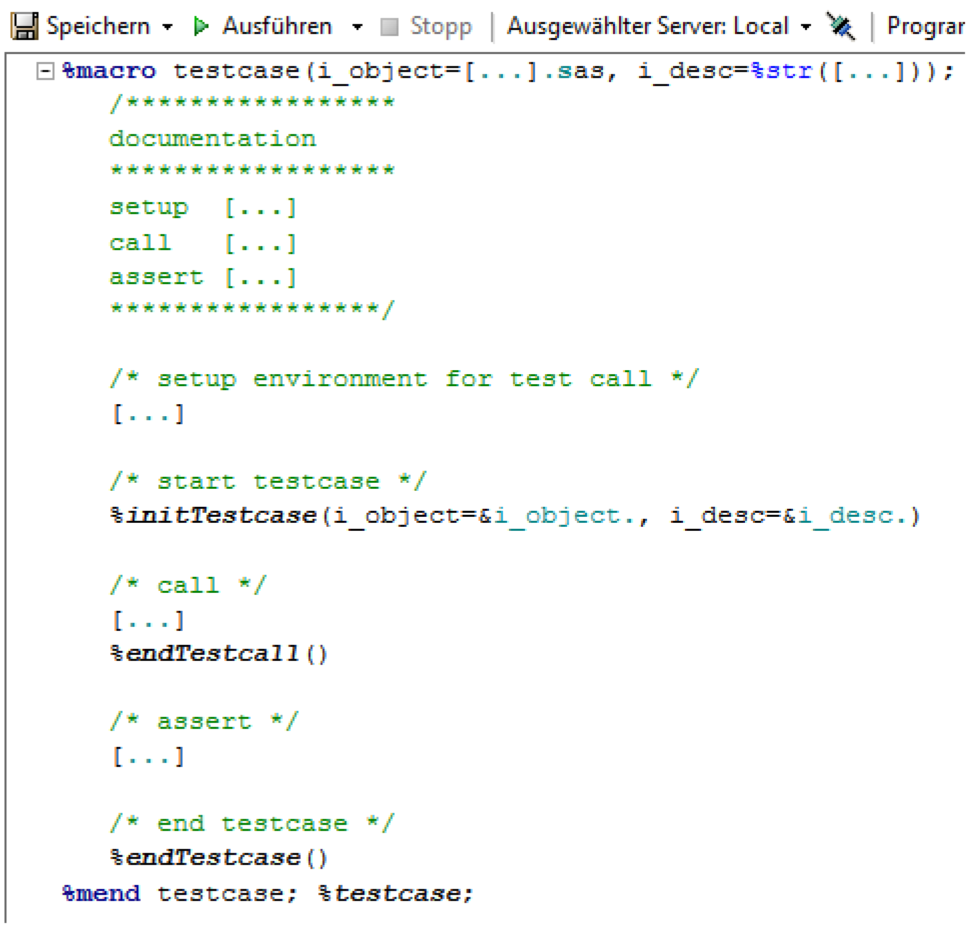
If errors occur in the scenario, you need to search for the corresponding test case in the source code. To facilitate this, there are keyboard macros that create a wrapper for the test case. When all wrappers (macros) are collapsed, you have a very good overview of the individual test cases and can navigate quickly.



```
 %initScenario(i_desc=Tests for getvars.sas);
⊞%macro testcase(i_object=getvars.sas, i_desc=%str(simple example with sashelp.class));
⊞%macro testcase(i_object=getvars.sas, i_desc=%str(simple example with sashelp.class, different delimiter));
⊞%macro testcase(i_object=getvars.sas, i_desc=%str(example with variable names containing special characters));
⊞%macro testcase(i_object=getvars.sas, i_desc=%str(example with empty dataset));
⊞%macro testcase(i_object=getvars.sas, i_desc=%str(example without dataset specified));
⊞%macro testcase(i_object=getvars.sas, i_desc=%str(example with invalid dataset));
⊞proc delete data=work.test;
 %endScenario();
 /** \endcond */
```

Figure 7: Overview of test cases through macros

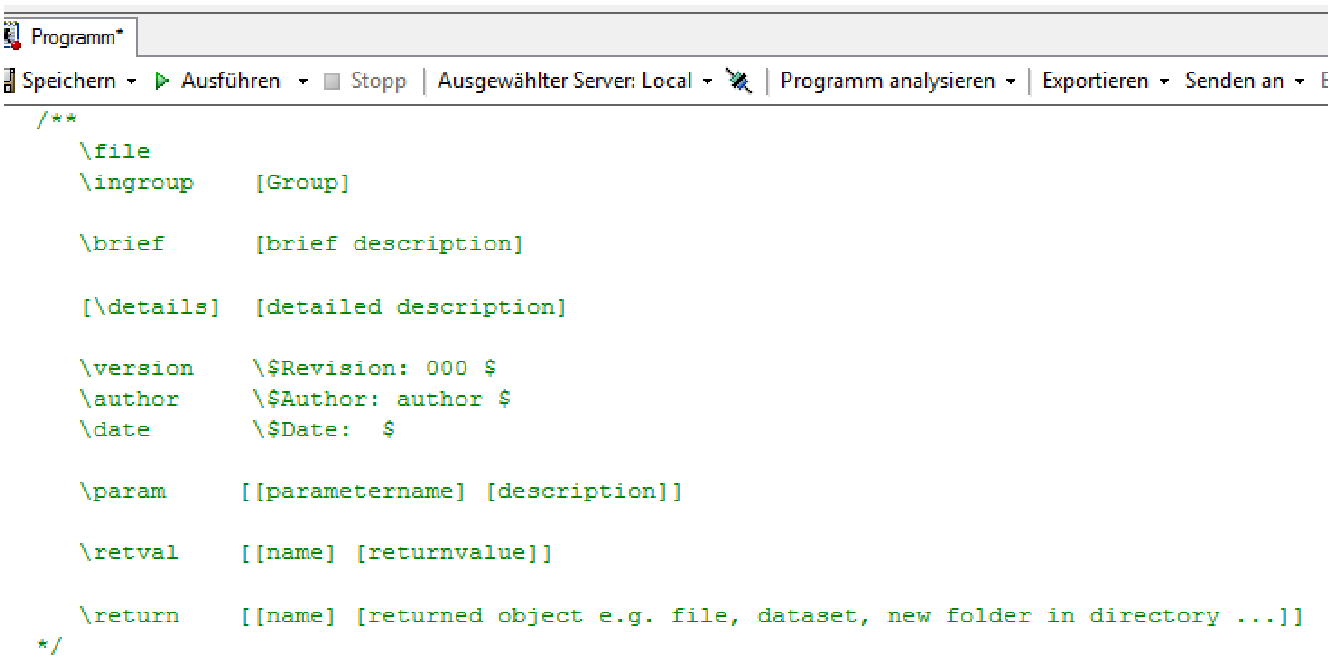
For simplification, we provide a macro that creates a framework for the test case.



```
%macro testcase(i_object=[...].sas, i_desc=%str([...]));
   /*****************
   documentation
   ******************
   setup  [...]
   call   [...]
   assert [...]
   *****************/

   /* setup environment for test call */
   [...]

   /* start testcase */
   %initTestcase(i_object=&i_object., i_desc=&i_desc.)

   /* call */
   [...]
   %endTestcall()

   /* assert */
   [...]

   /* end testcase */
   %endTestcase()
%mend testcase; %testcase;
```

Figure 8: Enterprise Guide macro for test cases

We also provide a macro for the comment header:



```
/**
   \file
   \ingroup    [Group]

   \brief      [brief description]

   [\details]  [detailed description]

   \version    \$Revision: 000 $
   \author     \$Author: author $
   \date       \$Date:   $

   \param     [[parametername] [description]]

   \retval    [[name] [returnvalue]]

   \return    [[name] [returned object e.g. file, dataset, new folder in directory ...]]
*/
```

Figure 9: Enterprise Guide macro for comment header

Here are the key sequences for the individual functions:

Function	**Key Sequence**	**Provider**
Insert comment header	Ctrl + Alt + c	HMS
Insert test case	Ctrl + Alt + t	HMS
Expand all	Ctrl + Alt + + (Numeric keypad)	SAS
Collapse all	Ctrl + Alt + - (Numeric keypad)	SAS

Table 2: SASUnit Enterprise Guide key sequences

4 Other innovations

Here is a list of the other new features since version 1.5

4.1 The program documentation is now part of SASUnit

SASUnit has previously supported DoxyGen for program documentation. Only a small part of the functionality of DoxyGen was needed. Many functions of DoxyGen are simply not possible with SAS programs.
Newer versions of DoxyGen no longer work with the configuration file we shipped.

Therefore the decision was made to integrate the program documentation into SASUnit. Much of the functionality used in DoxyGen has been re-programmed in SAS.

4.2 New Style

To achieve a consistent appearance between the DoxyGen program documentation and the SASUnit test report, the layout of the test report was based on DoxyGen.
With the adoption of the program documentation in SASUnit, the way was opened for a new, more modern layout. With a switch in the macro reportSASUnit you can choose between the two supplied layouts.

The facelift and the associated program changes also made it possible to use your own styles in SASUnit.

A detailed documentation can be found on the sourceforge platform:
https://sourceforge.net/p/sasunit/wiki/How%20to%20create%20your%20own%20SASUnit%20style/

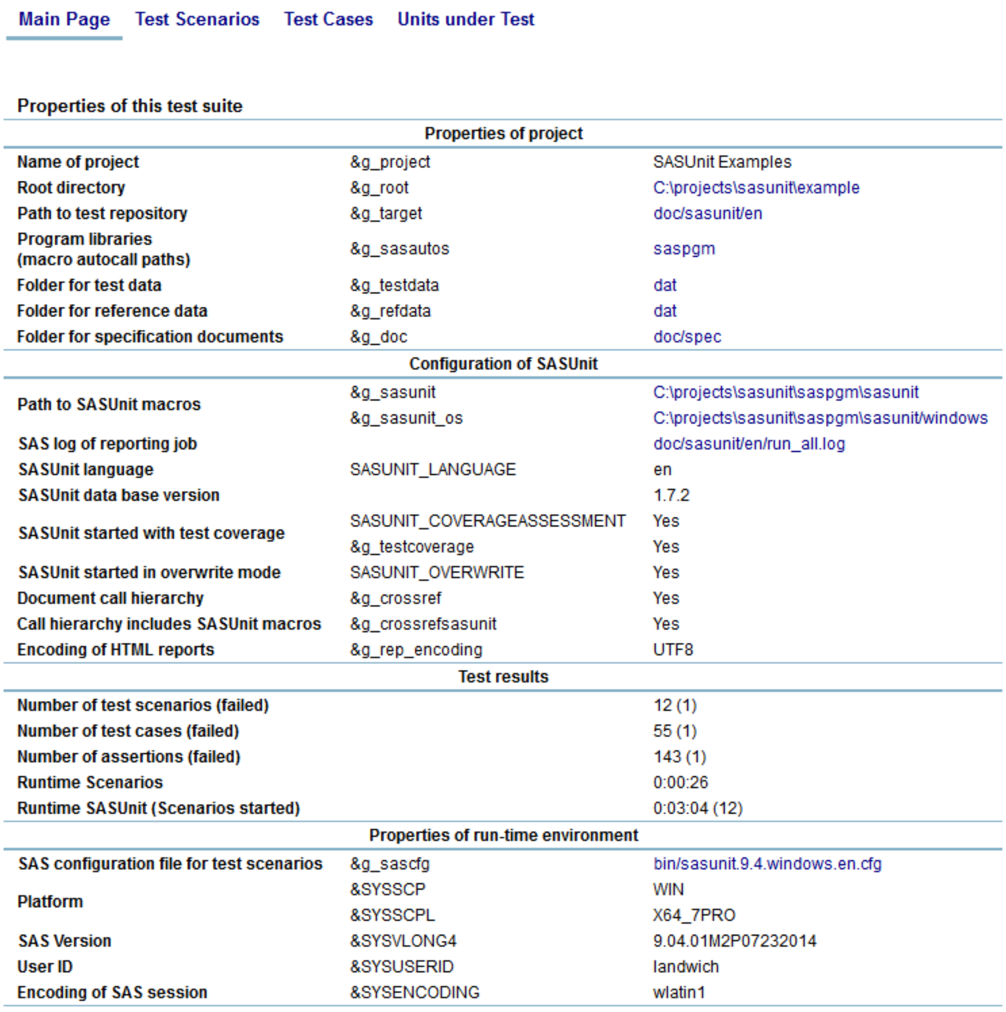
4.3 Extending the documentation to sourceforge

A "How To and Best Practices" section has been added to the documentation on sourceforge.
https://sourceforge.net/p/sasunit/wiki/How%20to%20and%20Best%20Practices/

Here are instructions and tips and tricks for using SASUnit. Results from and answers to posts are also documented if we find that they are of general interest.

4.4 New content on the Home HTML page

The home page has undergone a slight rework. More information is now dispalyed. In addition, up to 30 autocall paths are supported and not only ten as before.



Main Page Test Scenarios Test Cases Units under Test

Properties of this test suite

Properties of project		
Name of project	&g_project	SASUnit Examples
Root directory	&g_root	C:\projects\sasunit\example
Path to test repository	&g_target	doc/sasunit/en
Program libraries (macro autocall paths)	&g_sasautos	saspgm
Folder for test data	&g_testdata	dat
Folder for reference data	&g_refdata	dat
Folder for specification documents	&g_doc	doc/spec
Configuration of SASUnit		
Path to SASUnit macros	&g_sasunit	C:\projects\sasunit\saspgm\sasunit
	&g_sasunit_os	C:\projects\sasunit\saspgm\sasunit\windows
SAS log of reporting job		doc/sasunit/en/run_all.log
SASUnit language	SASUNIT_LANGUAGE	en
SASUnit data base version		1.7.2
SASUnit started with test coverage	SASUNIT_COVERAGEASSESSMENT	Yes
	&g_testcoverage	Yes
SASUnit started in overwrite mode	SASUNIT_OVERWRITE	Yes
Document call hierarchy	&g_crossref	Yes
Call hierarchy includes SASUnit macros	&g_crossrefsasunit	Yes
Encoding of HTML reports	&g_rep_encoding	UTF8
Test results		
Number of test scenarios (failed)		12 (1)
Number of test cases (failed)		55 (1)
Number of assertions (failed)		143 (1)
Runtime Scenarios		0:00:26
Runtime SASUnit (Scenarios started)		0:03:04 (12)
Properties of run-time environment		
SAS configuration file for test scenarios	&g_sascfg	bin/sasunit.9.4.windows.en.cfg
Platform	&SYSSCP	WIN
	&SYSSCPL	X64_7PRO
SAS Version	&SYSVLONG4	9.04.01M2P07232014
User ID	&SYSUSERID	landwich
Encoding of SAS session	&SYSENCODING	wlatin1

Figure 10: New content on the home page

5 Perspectives / Appeal to the community

We are aware that SASUnit 2.0 is only a first step towards client-server integration.

Where the journey for SASUnit will lead is not clear to us at the moment either.

Let us continue along this path and support the development of scenarios better.

Or should there be a kind of SASUnit plug-in for the Enterprise Guide, where you can select scenarios for the test? The plug-in would then also offer the option of executing a batch call. In addition, processing could be moved to a workspace server session. It would also be possible to select individual scenarios for testing.

The areas of application known to us are characterised by the Display Manager. Therefore, the step towards client-server use was a cautious one.

However, if there is a specific need for an extension of TDD support and work in the Enterprise Guide, we will be happy to do so.