1. Wie funktioniert SASUnit 2.0

Im Nachfolgenden werden beide Arbeitsweisen (batch und interaktiv) mit SASUnit vorgestellt.

1.1 SASUnit – Batch (wie bisher)

Die Stärke von SASUnit als Batchaufruf ist die Dokumentation der gesamten Testsuite und das automatisierte Wiederholen von Tests. Im Batchaufruf ist die Laufzeit weniger wichtig, denn hier kommt es auf die Dokumentation der Testergebnisse an und man erwartet auch eine längere Laufzeit.
In manchen Bereichen wird für die Erstellung der SAS® Programme der SAS® Display Manager eingesetzt. Somit befindet sich der Entwickler direkt auf der Umgebung, auf der auch SASUnit läuft und die Dokumentation erzeugt. Hier kann SASUnit seine ganze Stärke ausspielen.
Das war auch bisher die Zielrichtung in der Weiterentwicklung von SASUnit.

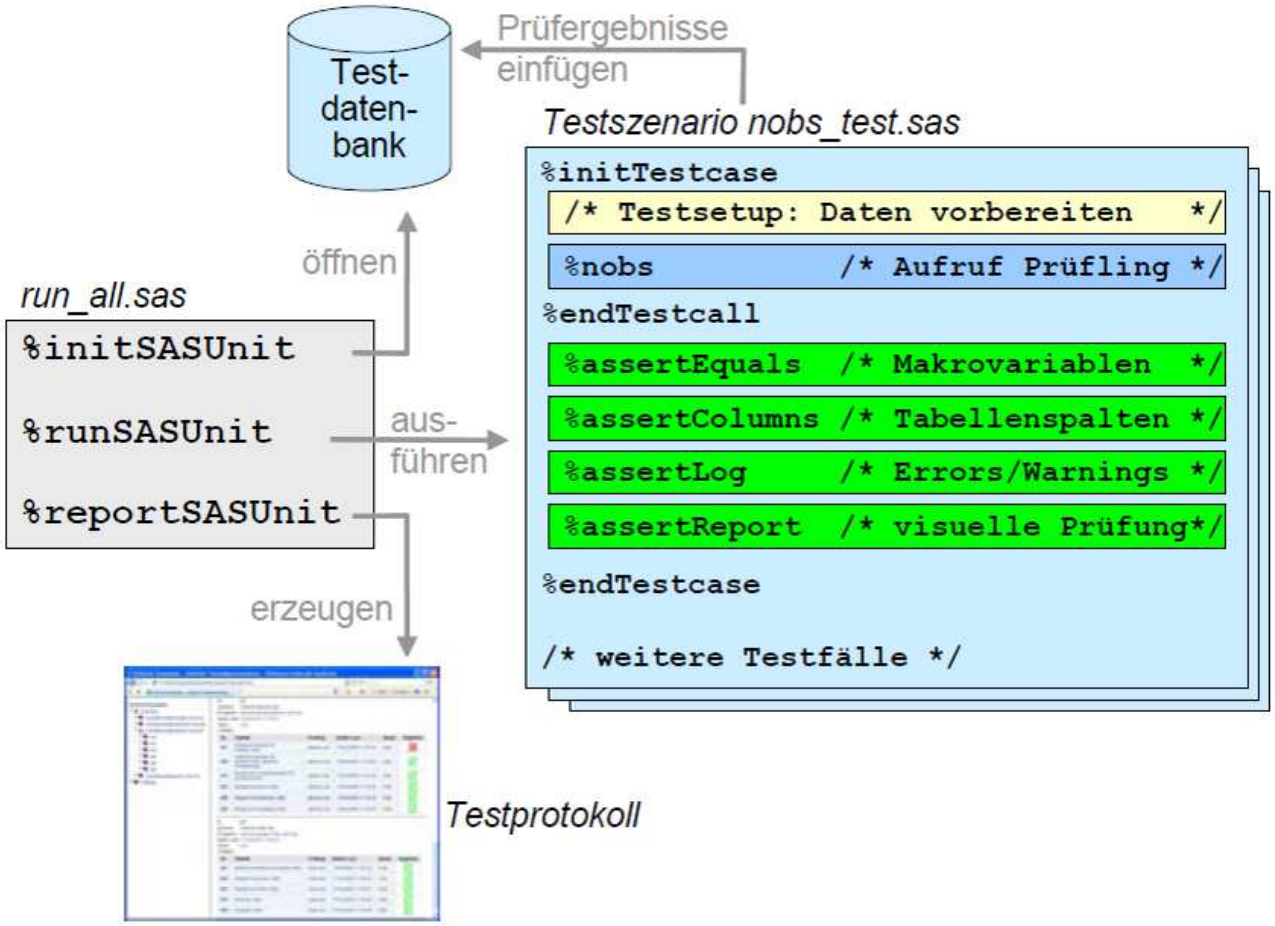


Abbildung 1: SASUnit Architektur (Batch)

1.2 SASUnit – interaktiv

Einige SAS-Umgebungen haben jedoch keinen Zugriff mehr auf den Display Manager und nutzen eine remote SAS Session. Bisher gab es für dieses Einsatzgebiet keine adäquate Möglichkeit, SASUnit zu verwenden.
Das Starten einer Batch-SAS® Sitzung auf einem remote Rechner ist mit SAS®/BASE Mitteln etwas komplizierter. Man kann das bspw. durch einen Stored Process erreichen, der auf dem remote Rechner eine Command-Datei ausführt.
Auch die Ergebnisse müssen erst von der remote Sitzung in den lokalen Client kopiert werden.
Warum sollte man das denn nicht dem SAS® Enterprise Guide überlassen. Der kann das doch schon.

Beim Erstellen eines Testszenarios ist es wichtiger, schnell die Ergebnisse des Ablaufs zu erhalten. Eine Testabdeckung oder eine Programmdokumentation sind hierbei weniger wichtig. Diese kann man auch in einem Batchraufruf erzeugen lassen, wenn das Szenario vollständig implementiert ist.
Mit einem schnellen, interaktiven Modus lässt sich SASUnit auch für testgetriebenes Entwickeln / Design sinnvoll einsetzen.
Das war der Startschuss für den interaktiven Modus von SASUnit.
Die vorliegende Lösung ist ein erster Schritt in eine neue Richtung. Hier ist sicherlich noch einiges zu erweitern und/oder zu verbessern. Hierzu wollen wir die Rückmeldung der Community verwenden, um nicht am Bedarf vorbei zu entwickeln.

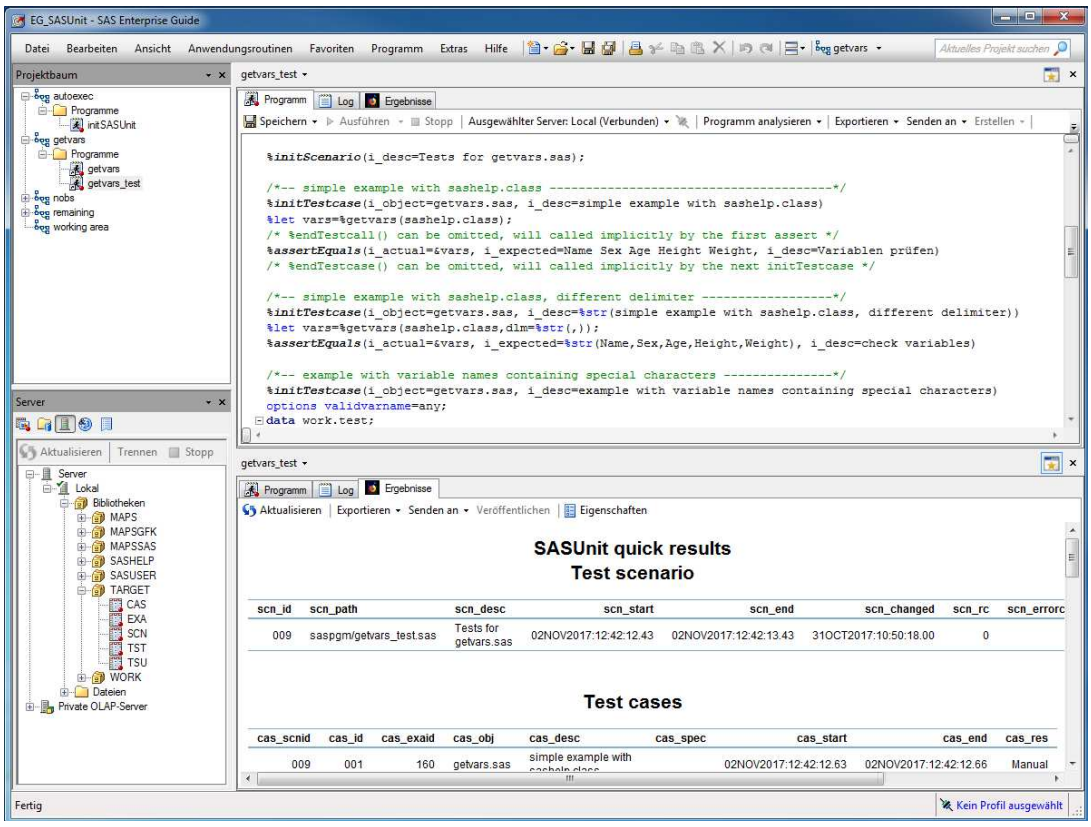


Abbildung 2: SASUnit Architektur (Interaktiv)

2 SASUnit 2.0 - Neuer interaktiver Modus

In diesem Abschnitt wird der Ansatz für den interaktiven Modus von SASUnit beleuchtet.

2.1 Ziel

Es ist das Ziel, im Enterprise Guide ein Testszenario schreiben **und ausführen** zu können. Ebenso soll ein kompakter Bericht über Erfolg bzw. Misserfolg der Testfälle verfügbar sein. Somit ist es möglich, SASUnit im Umfeld von TDD (Test-driven Development / Design) einsetzen zu können.

2.2 Architektur / Vorgehen

An der grundlegenden Architektur hat sich nicht viel geändert. Die Testdatenbank ist die gleiche und es wird auch immer noch ein Aufruf von ***initSASUnit*** benötigt. Bei diesem Aufruf wird jedoch nur noch ein Teil der Funktionalität ausgeführt. Die Prüfungen auf die Existenz der Autocall-Pfade werden weiterhin durchgeführt. Da jedoch das Log und der Bericht direkt in den Client zurückgeführt wird, werden bspw. die Verzeichnisse für Logs und Berichte nicht gelöscht.
Zurzeit wird auch beim interaktiven Aufruf die gleiche SASUnit Testdatenbank verwendet wie im Batchaufruf. Somit spiegelt die Testdatenbank immer den aktuellen Stand der Entwicklung wider.

Da jedes Testszenario nun selbst seine Szenario Id aus der Testdatenbank liest, muss es für den interaktiven Aufruf angepasst werden. (näheres siehe 3.3)
Am Anfang muss ein Aufruf von ***initScenario*** eingefügt werden.

```
/**
   \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()
```

Am Ende soll ein kurzer Bericht erzeugt werden. Diese Aufgabe übernimmt das Makro ***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 */
```

Der Fokus des interaktiven Aufrufs liegt in der Unterstützung von testgetriebenem Entwickeln / Design. Die Ausführung soll schnell sein und beschränkt sich deswegen auf die minimale Funktionalität, die während des Implementierens benötigt wird.

Für die Aufbereitung der gesamten Testdokumentation, der Testabdeckung, der Programmdokumentation und dem vollständigen Prüfen aller Prüflinge ist der Batchaufruf weiterhin unerlässlich!

Mit der aktuellen Architektur sollte der interaktive Aufruf auch in Umgebungen ohne XCMD lauffähig sein. Mit Ausnahme der Prüfungen, die Betriebssystemkommandos verwenden (***assertExternal***, ***assertImage*** und ***assertText***).

2.3 Rahmenbedingungen und Implikationen

Da es kein Rahmenprogramm mehr gibt, das den Aufruf steuert und Informationen übergibt, muss jedes Szenario wissen wie es heißt (Name des gerade ausgeführten SAS-

Programms), um die Testdatenbank richtig zu aktualisieren. Diese Informationen wurden bisher aus der aufrufenden Batch Session übergeben und wurden nun in das laufende Testszenario verlagert.

Dazu gehören:

- der Name des aufgerufenen Test Szenario
- die Szenario ID aus der Testdatenbank

Je nachdem wie das Testszenario aufgerufen wird (Batch oder Interaktiv) stehen diese Informationen an anderen Stellen in der SAS Session. Das Wissen darum ist in einem neuen Makro ***_readEnvMetadata*** gekapselt.

Aufgrund der architektonischen Änderungen und der Art des Aufrufs im Enterprise Guide gilt es Folgendes beachten:

Beim interaktiven Aufruf gibt es nur eine(!) SAS Session.

- Alle Szenarien teilen sich die Work Bibliothek.
  - Da am Ende des Szenarios die temporären Tabellen stehen bleiben, kann es zu unerwünschten Seiteneffekten kommen. Deshalb ist es nötig, am Ende des Szenarios aufzuräumen.
- Alle Szenarien teilen sich die Laufzeitumgebung und somit auch die Makrovariablen.
  - Auch hier kann es zwischen den Szenarien Seiteneffekte geben.
  - Wichtiger ist jedoch, dass es auch beim wiederholten Aufruf eines Szenarios Seiteneffekte geben kann. Es ist erforderlich, alle verwendeten Makrovariablen in den Testsetup mit einzubeziehen.

Nach dem Aufruf eines Szenarios im Enterprise Guide soll das komplette Log dort verfügbar sein.

- Eine Aufteilung des Logs in einzelne Abschnitte per PROC PRINTTO ist nicht mehr angezeigt.
- Das Log wird in den Enterprise Guide zurück gestreamt. In der aktuellen Version wird dieses Log nicht ausgewertet. Somit können die beiden Makros ***assertLog*** und ***assertLogMsg*** nicht wie gewohnt arbeiten. Beide Makros tragen diesem Umstand Rechnung und setzen das Testergebnis auf manuell und schreiben eine passenden Meldung in die Testdatenbank:

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.

Abbildung 3: Meldung (interaktiv) von assertLog und asertMsg

Aufgrund der Einschränkung bei einigen SAS-Umgebungen ist auch die Unterstützung für NOXCMD Umgebungen implementiert.
Dies hat jedoch wie schon erwähnt zur Folge, dass die externen Prüfungen nicht funktionieren können.

Das gilt für die folgenden Makros:
- assertExternal
- assertImage
- assertText

Falls eines dieser Makros in einer NOXCMD Umgebung eingesetzt wird, dann setzt es das Prüfungsergebnis auf manuell und schreibt eine entsprechende Meldung in die Testdatenbank.

```
%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 Unterstütze Einsatzszenarien

Der interaktive Modus ist dazu gedacht, die Testszenarien schneller und nach TDD Methodik zu entwickeln.
Nachfolgend die von uns angedachten Einsatzszenarien.

2.4.1 Aufrufszenarien

Der Einsatz mit einer lokalen SAS Session im Enterprise Guide funktioniert am besten. Hier hat man die Rechte, Betriebssystembefehle absetzen zu dürfen und SASUnit tut sich hier am leichtesten.

Aber auch mit einer Remote Session lässt sich arbeiten.
Hier kann man mehrere Varianten je nach Lage der Testszenarien und der SASUnit Makros unterscheiden.

Testszenarien	SASUnit Makros
Lokal	Lokal
Lokal	Remote
Remote	Remote

Tabelle 1: SASUnit interaktiv Aufrufszenarien

Alle Szenarien sind mit SASUnit 2.0 möglich.

2.4.2 Clients für SASUnit 2.0

Der bevorzugte Client für SASUnit 2.0 ist klar der SAS® Enterprise Guide. Mit diesem wird bei HMS entwickelt und getestet.

SASUnit 2.0 ist auch unter SAS® Studio und Jupyter Notebooks lauffähig.

Für Jupyter Notebooks musste eine Anpassung vorgenommen werden.
Durch die Art des SAS®-Aufrufs durch Jupyter Notebooks ist in der SAS® Session nicht erkennbar, welches Programm gerade ausgeführt wird. Somit fehlt die Information für das Update der Testdatenbank. Um dieses Manko wett zu machen, gibt es einen zusätzlichen Parameter ***i_object*** für das Makro ***initScenario***.

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

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

Somit ist es auch mit Jupyter Notebooks möglich, SASUnit 2.0 zu verwenden.

2.5 Fehlende Abwärtskompatibilität

SASUnit 2.0 trägt diese Versionsnummer mit Bedacht. Die Vorgängerversion ist 1.7. Um SASUnit nicht unnötig kompliziert zu machen, verwenden beide Aufrufvarianten (Batch und interaktiv) die gleiche Logik.

Ab Version 2.0 wird auch im Batchmodus die Szenario Id über ***initScenario*** ermittelt.
In einzelnen Fällen kann es nötig sein, ein Testszenario anzupassen, damit es im Batch weiterhin funktioniert.

Beim Aufruf von ***initTestcase*** wird geprüft ob ***initSenario*** schon gelaufen ist. Falls nicht, wird es aufgerufen. Es besteht also kein Zwang alle Szenarien anzupassen.

Wenn in einem Szenario beim Testsetup auf SASUnit Makrovariablen zugegriffen wird, so wie das bei einigen unserer Testszenarien der Fall ist, dann muss man explizit den Aufruf von ***initScenario*** **davor(!)** einfügen.

```
/**
   \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
)
```

Mehr muss für den Batchaufruf von SASUnit 2.0 nicht an den Szenarien angepasst werden.

3 Angedachte Arbeitsweise im interaktiven Modus

Der Enterprise Guide ist der bevorzugte Client für SASUnit 2.0, weshalb sich die von HMS angedachte Arbeitsweise auch daran orientiert.
SASUnit 2.0 liefert für das Beispiel Projekt ein funktionsfähiges Enterprise Guide Projekt (mit lokaler und remote SAS Session) mit. Es zeigt wie man mit dem Enterprise Guide und SASUnit 2.0 umgehen kann.

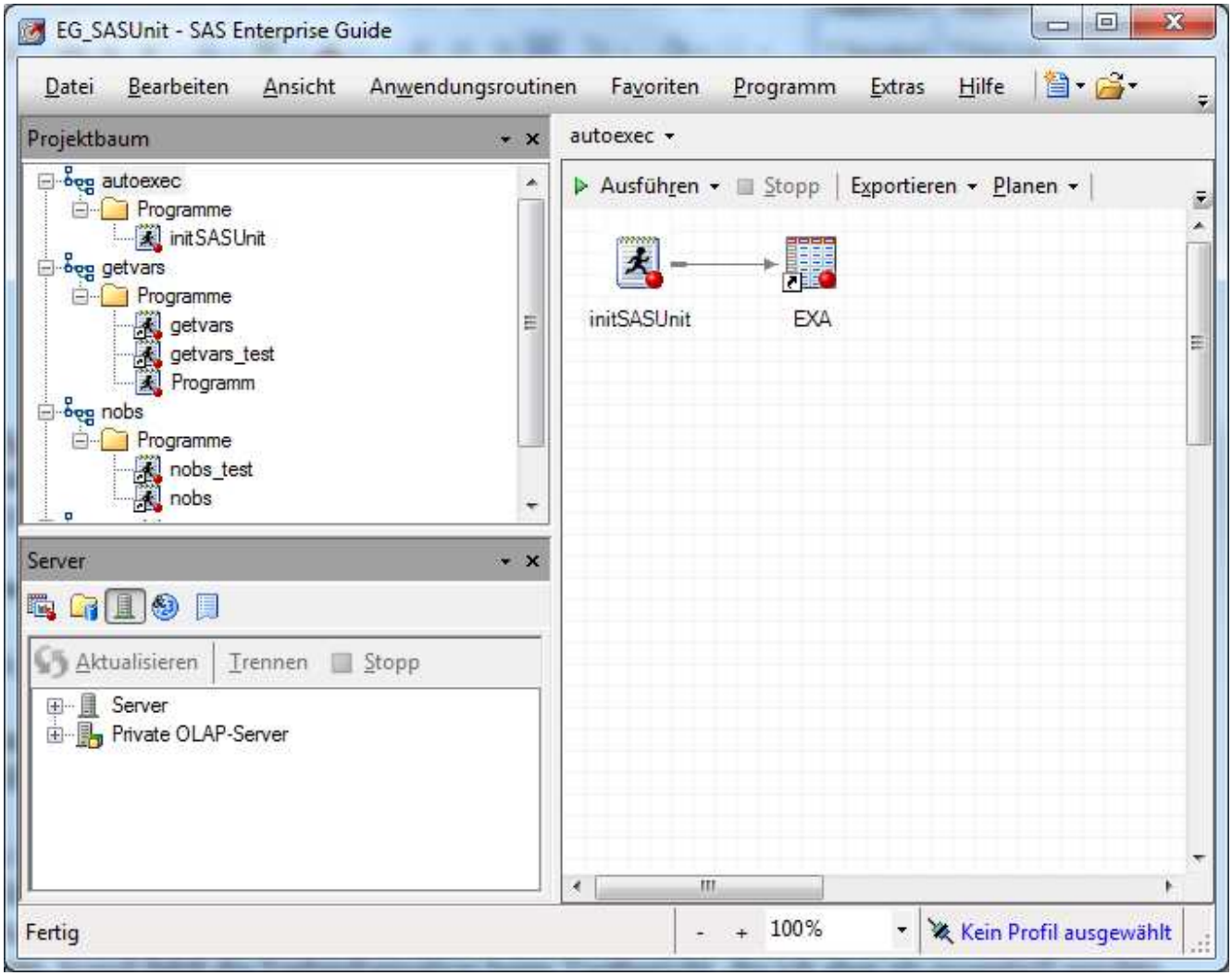


Abbildung 4: Arbeitsweise im Enterprise Guide Prozessfluss autoexec

Im Autoexec-Prozessfluss wird ***initSASUnit*** automatisch aufgerufen.

Die restlichen Prozessflüsse dienen der Strukturierung des Projekts und sind hier willkürlich gewählt.

Es kann sein, das Standard Reporting Format des Enterprise Guide (SAS Report) keine Inline-Formatierung rendert. Das liegt an der Kombination SAS Enterprise Guide und SAS Version im Hintergrund. Bspw. bei SAS 9.4 Maintenance Release 2 funktioniert es nicht. Somit fehlt die Farbinformation beim Testbericht, die ich aber als essentiell erachte.

Um farbige Reports zu erhalten muss man beim Szenario einmalig das Ergebnis auf HTML umstellen - dann klappt‘s auch mit dem Testbericht.
Mit SAS 9.4 und Maintenance Release 5 klappt es aber.

3.1 Arbeiten mit Prozessfluss working area

Eine Sonderstellung kommt dem Prozessfluss ***working area*** zu. Er dient dazu, das gerade in Arbeit befindliche Makro samt Testszenario zu enthalten. Somit lässt sich der Testaufruf einfach mit F3 auf dem Prozessfluss auslösen. Alternativ kann man auch mit dem Kontextmenü der rechten Maustaste auf dem Makro „Zweig ausführen von…“ wählen.

Nach Beendigung der Implementierung kann man den kompletten Zweig in einen anderen Prozessfluss verschieben und hat dann die ***working area*** wieder frei für die Implementierung des nächsten Features.

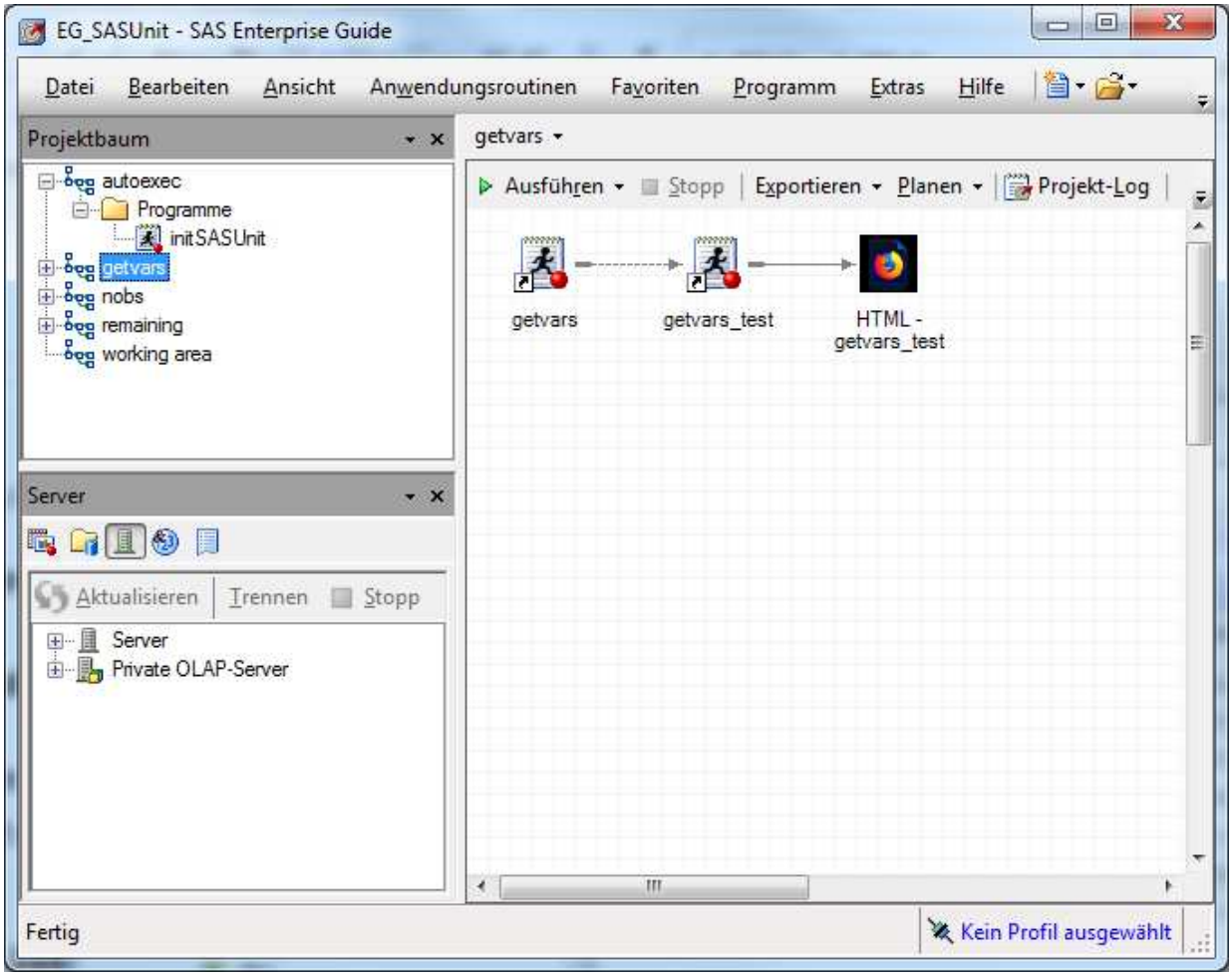


Abbildung 5: Arbeitsweise im Enterprise Guide mit working area

3.2 Arbeiten mit parallelen Fenster

Eine weitere Möglichkeit mit dem Enterprise Guide zu arbeiten besteht darin, den Prüfling und das Szenario parallel in zwei Fenstern zu öffnen. Somit kann man den Prüfling bearbeiten und die Ergebnisse des Szenarios aktualisieren.
Dies ermöglicht ein flüssigeres Arbeiten.

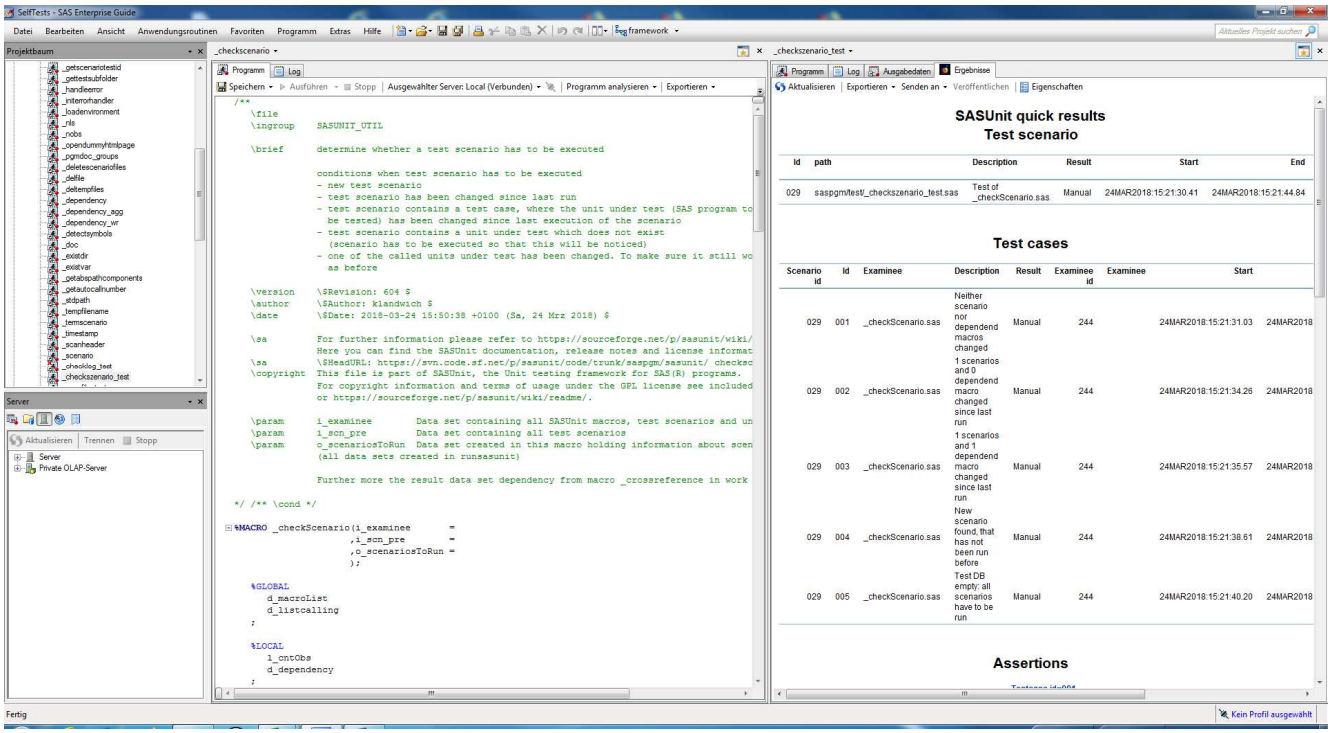


Abbildung 6: Arbeitsweise im Enterprise Guide mit parallelen Fenstern

3.2 Mehr Übersicht im Szenario

Bei Fehlern im Szenario muss man den entsprechenden Testfall im Quellcode suchen. Um diese zu erleichtern gibt es Tastaturmakros, die einen Wrapper für den Testfall erstellen. Wenn alle Wrapper (Makros) zusammengeklappt sind, dann hat man eine sehr gute Übersicht der einzelnen Testfälle und kann schnell navigieren.

```
%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 */
```

Abbildung 7: Übersicht der Testfälle mit Makros

Zur Vereinfachung liefern wir ein Makro mit, das einen Rahmen für den Testfall bildet.

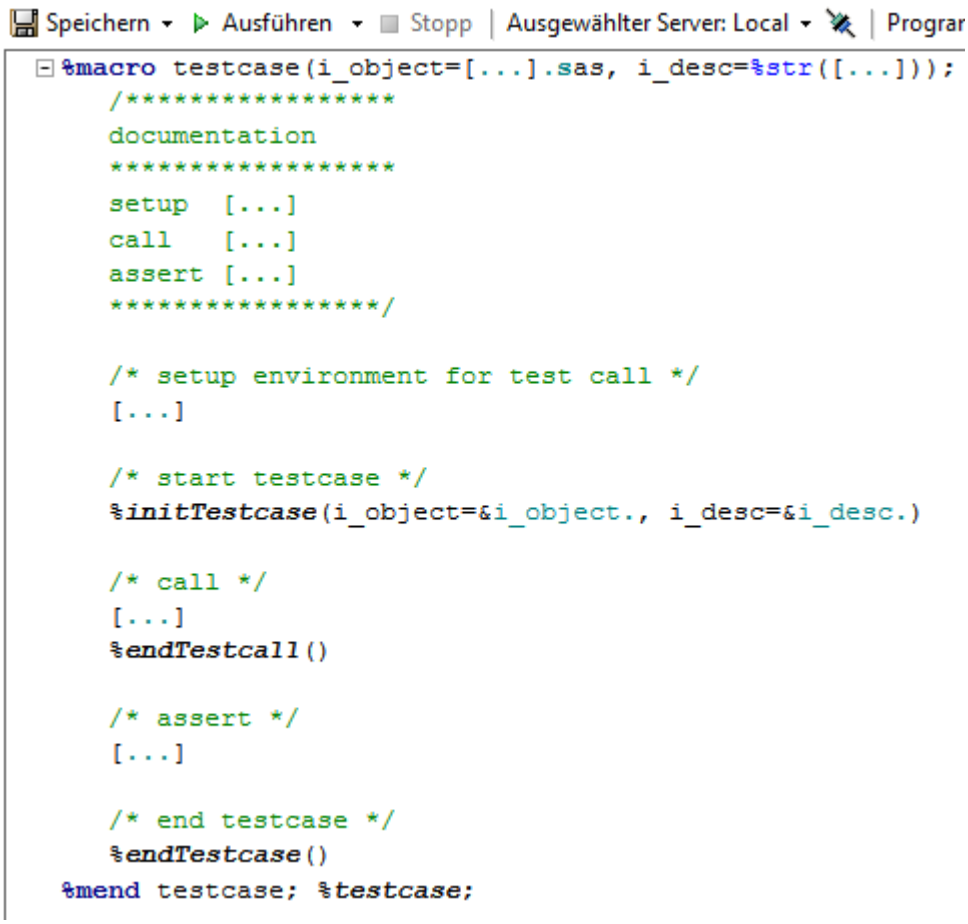


Abbildung 8: Enterprise Guide Makro für Testfälle

Auch für den Kommentarkopf liefern wir ein Makro mit:

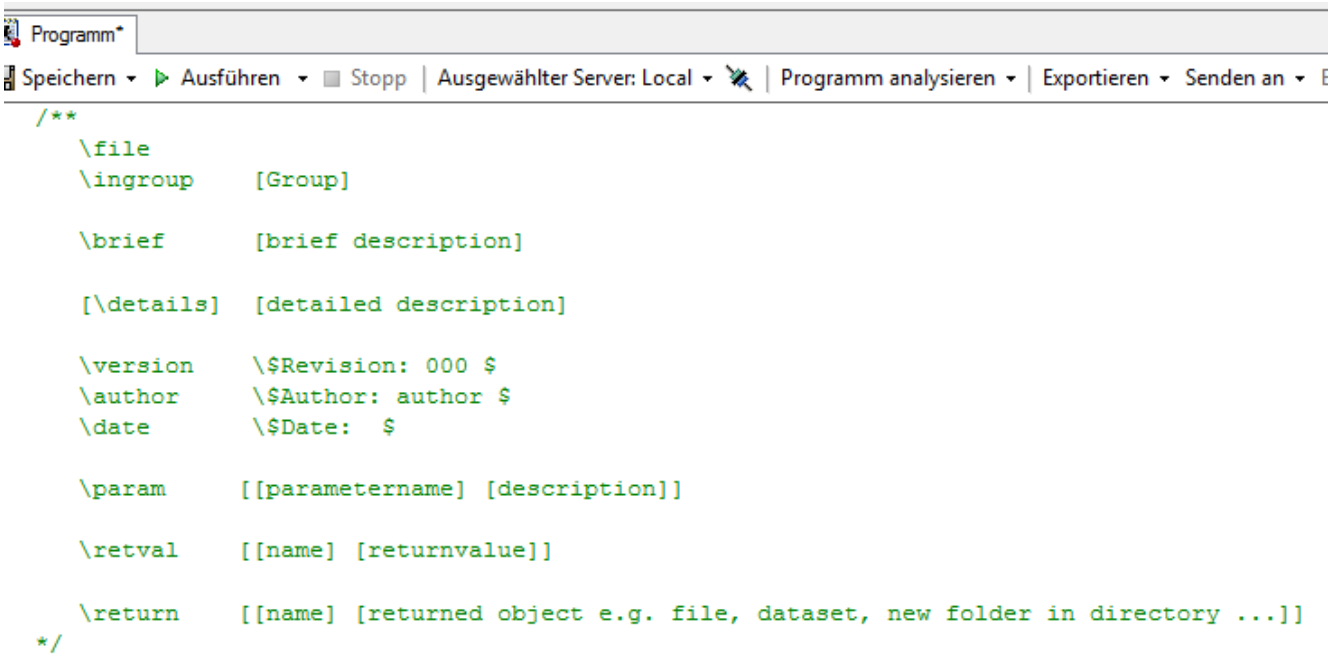


Abbildung 9: Enterprise Guide Makro für Kommentarkopf

Hier die Tastenkombinationen für die einzelnen Funktionen:

Funktion	Tastenkombination	Provider
Kommentarkopf einfügen	Strg + Alt + c	HMS
Testfall Makro einfügen	Strg + Alt + t	HMS
Alles aufklappen	Strg + Alt + + (Ziffernblock)	SAS
Alles zusammenklappen	Strg + Alt + - (Ziffernblock)	SAS

Tabelle 2: SASUnit Enterprise Guide Tastenkombinationen

4 Sonstige Neuerungen

Hier folgt eine Auflistung der sonstigen Neuerungen seit Version 1.5

4.1 Die Programmdokumentation ist jetzt ein Teil von SASUnit

SASUnit hat für die Programmdokumentation bisher DoxyGen unterstützt. Vom Funktionsumfang wurde nur ein kleiner Teil von DoxyGen benötigt. Viele Funktionen von DoxyGen sind mit SAS Programmen auch einfach nicht möglich.
Neuere Versionen von DoxyGen arbeiten nicht mehr mit der von uns ausgelieferten Konfigurationsdatei zusammen.

Deshalb fiel die Entscheidung die Programmdokumentation in SASUnit zu integrieren. Ein Großteil der genutzten Funktionalität aus DoxyGen wurde in SAS nach-programmiert.

4.2 Neuer Style

Um ein einheitliches Erscheinungsbild zwischen der DoxyGen-Programmdokumentation und dem SASUnit Testbericht zu erreichen, war das Layout des Testberichts an DoxyGen angelehnt.
Mit der Übernahme der Programmdokumentation in SASUnit war der Weg frei für ein neues, moderneres Layout. Über einen Schalter im Makro ***reportSASUnit*** kann man zwischen den beiden mitgelieferten Layouts auswählen.

Durch das Facelift und der damit verbunden Programmänderungen wurde auch die Möglichkeit geschaffen eigene Styles in SASUnit zu verwenden.

Eine ausführliche Dokumentation hierzu findet sich auf der sourceforge Plattform:
https://sourceforge.net/p/sasunit/wiki/How%20to%20create%20your%20own%20SASUnit%20style/

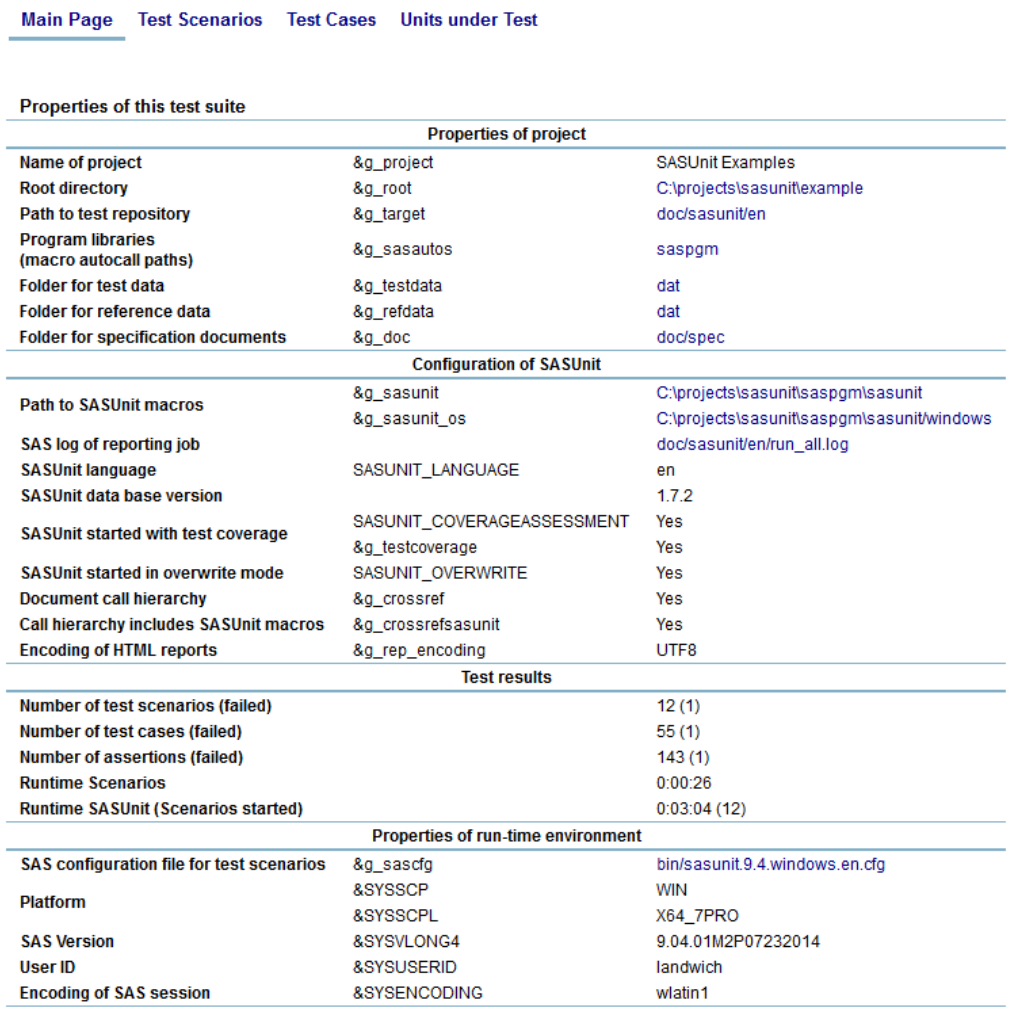
4.3 Erweiterung der Dokumentation auf sourceforge

Die Dokumentation auf sourceforge wurde um einen „How To and Best Practices“-Abschnitt erweitert.
https://sourceforge.net/p/sasunit/wiki/How%20to%20and%20Best%20Practices/

Hier liegen Anleitungen sowie Tipps und Tricks für den Umgang mit SASUnit. Auch Ergebnisse von und Antworten auf Posts sind dort dokumentiert, falls wir finden, dass sie von allgemeinem Interesse sind.

4.4 Neue Inhalte auf der Home HTML Seite

Die Home-Seite hat ein leichtes Rework erfahren. Es werden nun mehr Informationen angezeigt. Zusätzlich werden jetzt bis zu 30 Autocall-Pfade unterstützt und nicht wie bisher maximal zehn.



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

Abbildung 10: Neue Inhalte auf der Home-Seite

5 Ausblick / Aufruf an die Community

Es ist uns bewusst, dass SASUnit 2.0 nur ein erster Schritt in Richtung Client-Server Integration ist.

Wo die Reise für SASUnit hinführt ist auch für uns im Moment nicht klar.

Gehen wir diesen Weg weiter und unterstützen das Entwickeln der Szenarien besser.

Oder soll es doch ein Art SASUnit Plug-In für den Enterprise Guide geben, bei dem man Szenarien für den Test auswählen kann. Das Plug-In böte dann auch die Möglichkeit einen Batchaufruf auszuführen. Zusätzlich ließe sich die Abarbeitung auch in eine Workspace-Server Session verlagern. Auch die Möglichkeit gezielt einzelne Szenarien für den Test auszuwählen wäre dann möglich.

Die uns bekannten Einsatzgebiete sind vom Display Manager geprägt. Deshalb war der Schritt in Richtung Client-Server Einsatz erst mal ein vorsichtiger.

Wenn es jedoch konkret Bedarf für eine Erweiterung der TDD Unterstützung und der Arbeit im Enterprise Guide gibt, dann kommen wir dem gerne nach.