Structured Code Coverage eXtension

der einfache Weg zum automatisierten Test
Testabdeckung in IEC61131-3 Structured Text/ ST / SCL

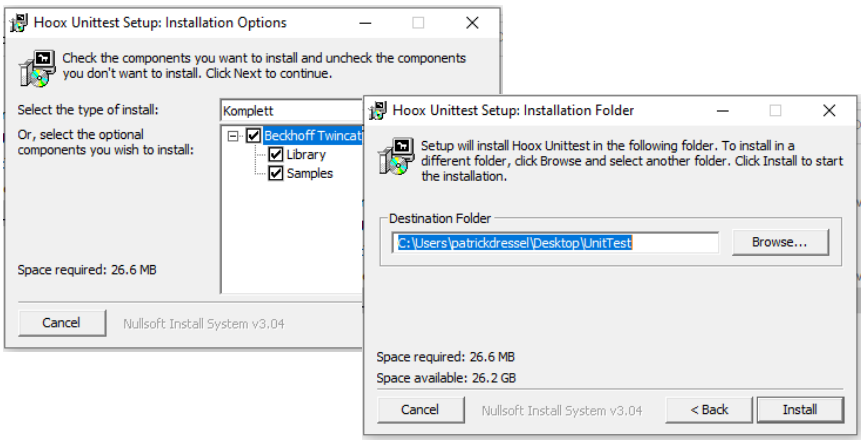
Hinweis

Lizensierung

Die Nutzung der Bibliothek ist kostenlos. Die Schnittstelle für die eigene Auswertung der Tests Tests ist dokumentiert und kann sofort verwendet werden.

Installation

Der Installer ist unter https://www.hoox.software/downloads abgelegt und kann dort frei heruntergeladen werden. Während der Installation können die optionalen Teile eingestellt werden.



Nach der Installation befinden sich alle notwendige Dateien in angegebenen Verzeichnis.

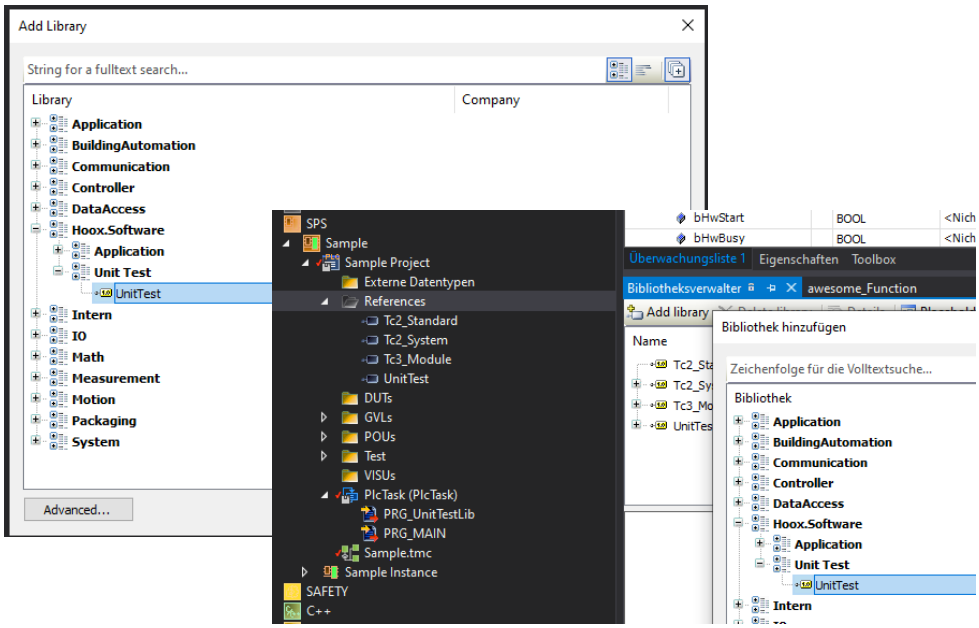
Pfad	Beschreibung
lib/UnitTest.library	Bibliothek
Sample/UnitTestSample	Twincat 3 Beispiel

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Bibliothek verwenden

Bibliothek importieren

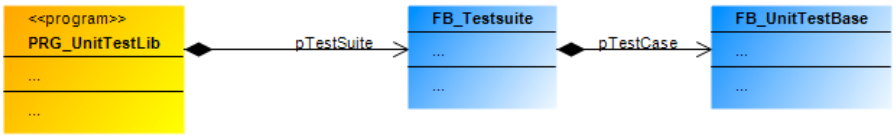
Die Verwaltung des Bibliotheksrepository öffnen und die neue Bibliothek hinzufügen. Danach in der Applikation die Bibliothek einbinden.



Anwendung

Allgemein

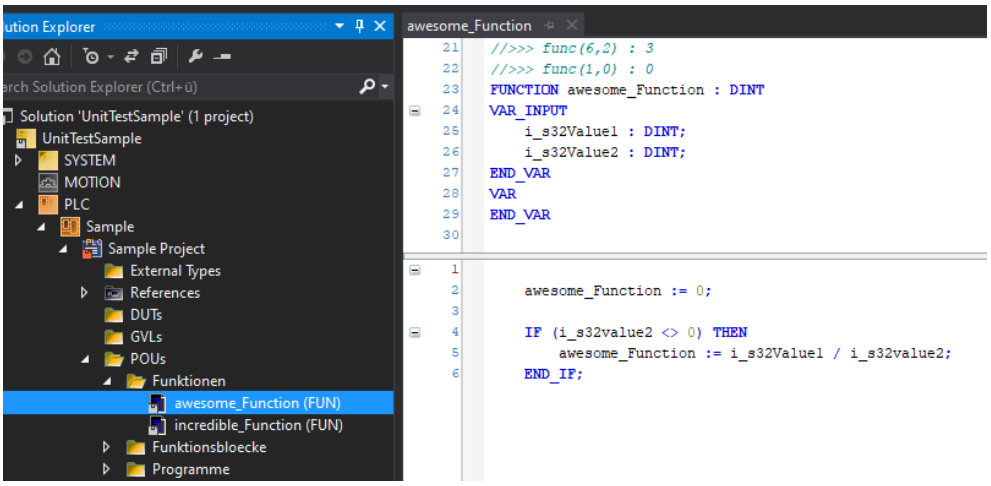
Die UnitTestlib besteht im wesentlichen aus drei Teilen



Baustein	Beschreibung
PRG_UnitTestLib	Verwaltet alle konfigurierten TestSuites
FB_TestSuite	Verwaltet die zugeordneten Testfälle
FB_UnitTestBase	Basisklasse für einen Testfall
FB_TextReport	Beispiel für einen Export des Testlaufs
FB_ReportBase	Basisklasse für einen Report der nach dem Testlauf aufgerufen wird.

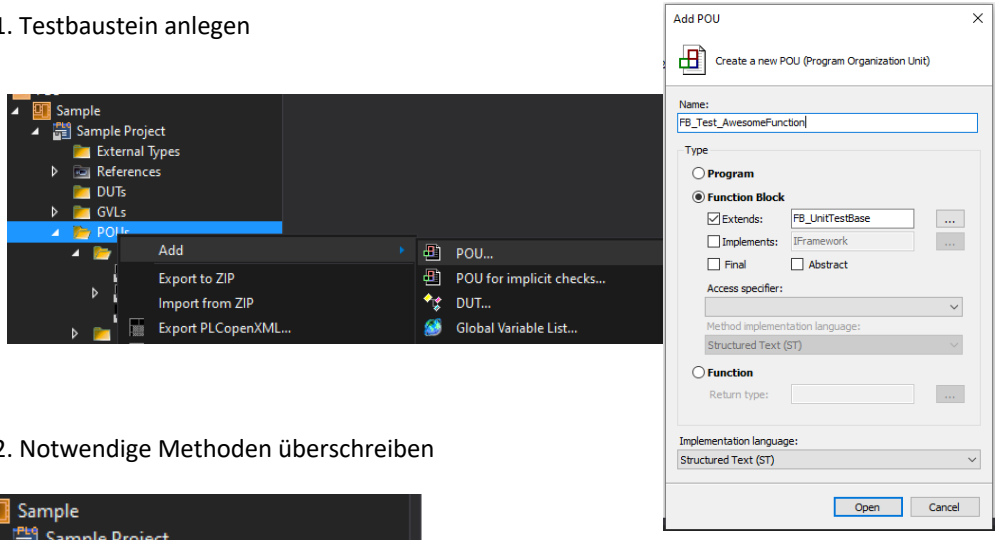
Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Beispiel : Es soll die folgende Funktion `awesome_Function` getestet werden.

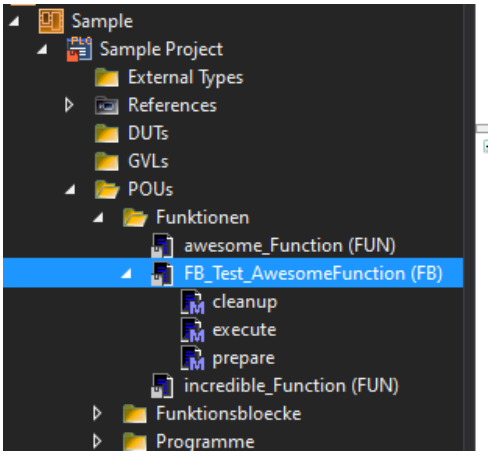


Vorgehensweise

1. Testbaustein anlegen



2. Notwendige Methoden überschreiben



Um den Testbaustein zu verwenden müssen die Methoden
`prepare` *(z.B für Abfrage Testeingangsbedingung)*
`execute` *(TestAusführung)*
`cleanup` (Testausgangsbedingung)
überschrieben werden.

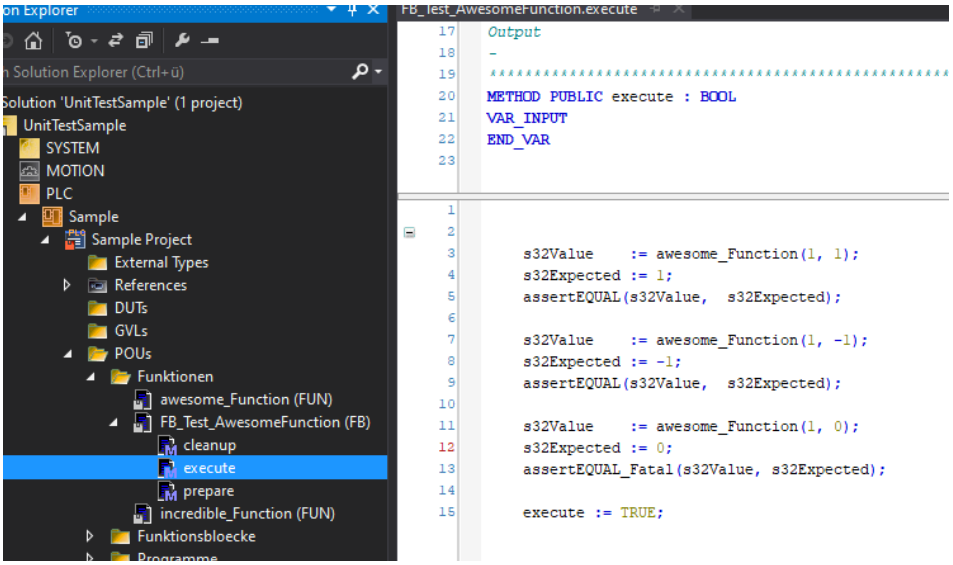
Im ersten Schritt reicht es aus wenn `prepare` und `cleanup` jeweils `true` zurückgeben.
In diesen beiden Methoden kann Code eingetragen werden der vor (`prepare`) oder nach (`cleanup`) dem Testfall ausgeführt werden soll.

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

3. Test implementieren

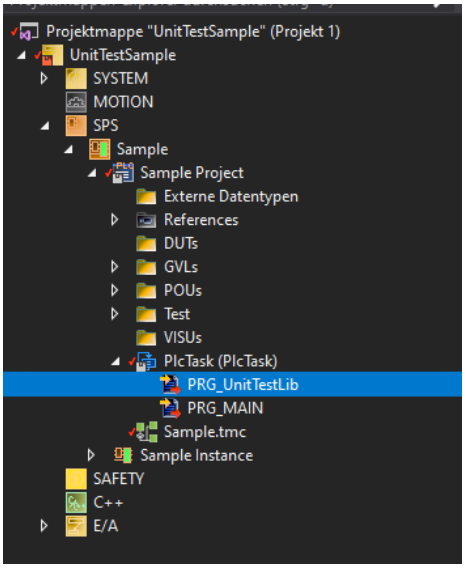
Für die Einstufung des Testfalls ("passed" oder "failed") müssen sogenannte Asserts aufgerufen werden. Diese bestehen immer aus einem aktuellen und einem erwarteten Wert. Ein Assert ist eine Behauptung das der aktuelle dem erwarteten Wert entpricht --> `AssertTrue` oder **nicht** entpricht --> `AssertFalse`
Dieser Wert **muss** als Variable übergeben werden.
- Es kann jeder Datentyp verwendet werden,
- Es müssen für beide Parameter die gleichen Datentypen verwendet werden.

Auf das Beispiel bezogen:
Die Funktion erwartet zwei DINT Werte als Eingangsparameter und gibt einen DINT Wert zurück.
In den lokalen Variablen des Testbaustein benötigen wir folgende Variablen
`s32Value` für das Ergebnis der Funktion
`s32Expected` für das erwartete Ergebnis.
Im Beispiel wird der Test mit drei Aufrufen durchgeführt.



4. Test ausführen

Um die Testfälle ausführen zu können muss das Programm `PRG_UnitTestLib` aus der Bibliothek an den PlcTask angehängt werden.



Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Testbausteine Instanzieren

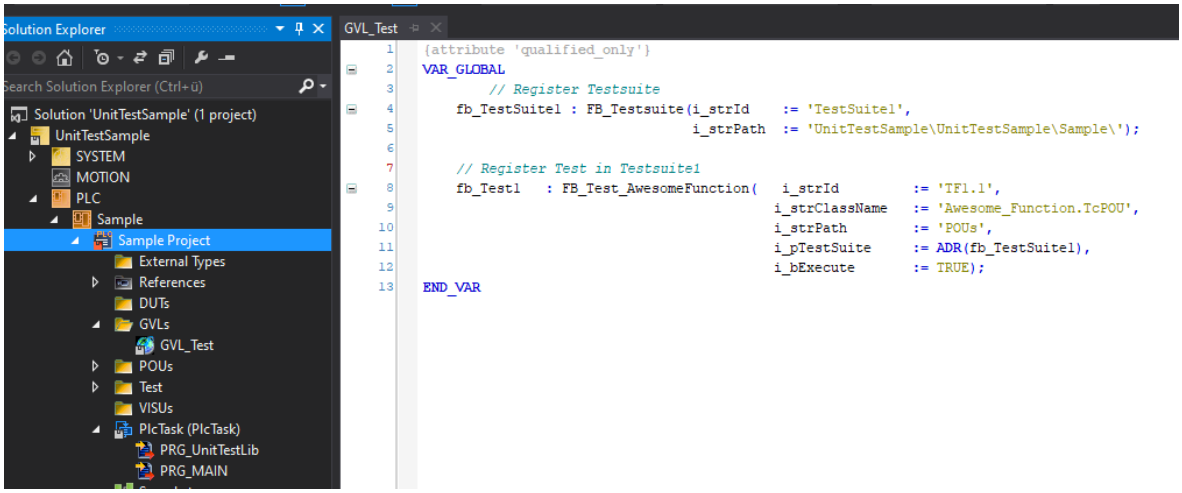
Die Testbausteine müssen innerhalb einer globalen Variablenliste instanziert werden. Für die Ausführung wird zur besseren Übersicht eine Testsuite benötigt die mehrere Tests verwaltet. Für die spätere Auswertung müssen noch weitere Informationen der Instanz übergeben werden:

Testsuite

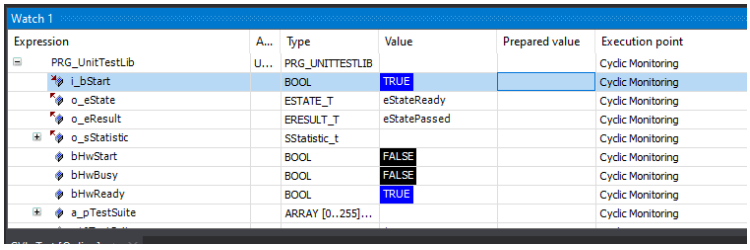
Parameter	Beschreibung
`i_strId`	eindeutige Id der Testsuite
`i_strPath`	Pfad zum Quellcode der enthaltenen Testfälle

Testfall

Parameter	Beschreibung
`i_strId`	eindeutige Id der Testsuite
`i_strClassName`	Dateiname des getesteten Bausteins
`i_strPath`	Pfad zum Quellcode der enthaltenen Testfälle
`i_pTestSuite`	Zeiger auf Testsuite die den Testfall verwaltet
`i_bExecute`	Testfall ausführen ja/nein. Bei nein wird dieser mit dem Status `skipped` übersprungen.

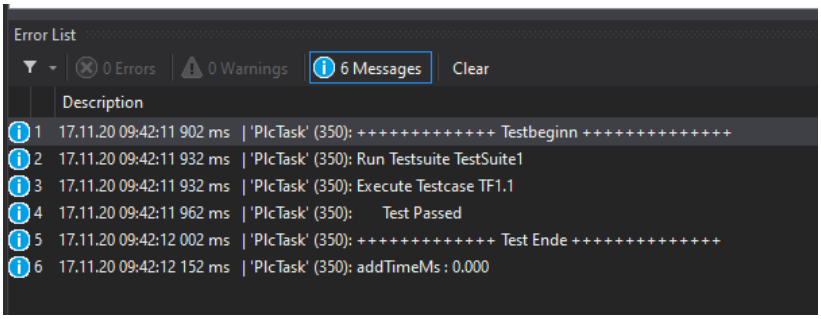


Anschließend kann das Projekt wie gewohnt auf die Steuerung geladen und gestartet werden. Um den Testlauf zu starten muss die Variable `PRG_UnitTestLib.i_bStart` auf den Wert `true` gesetzt werden.

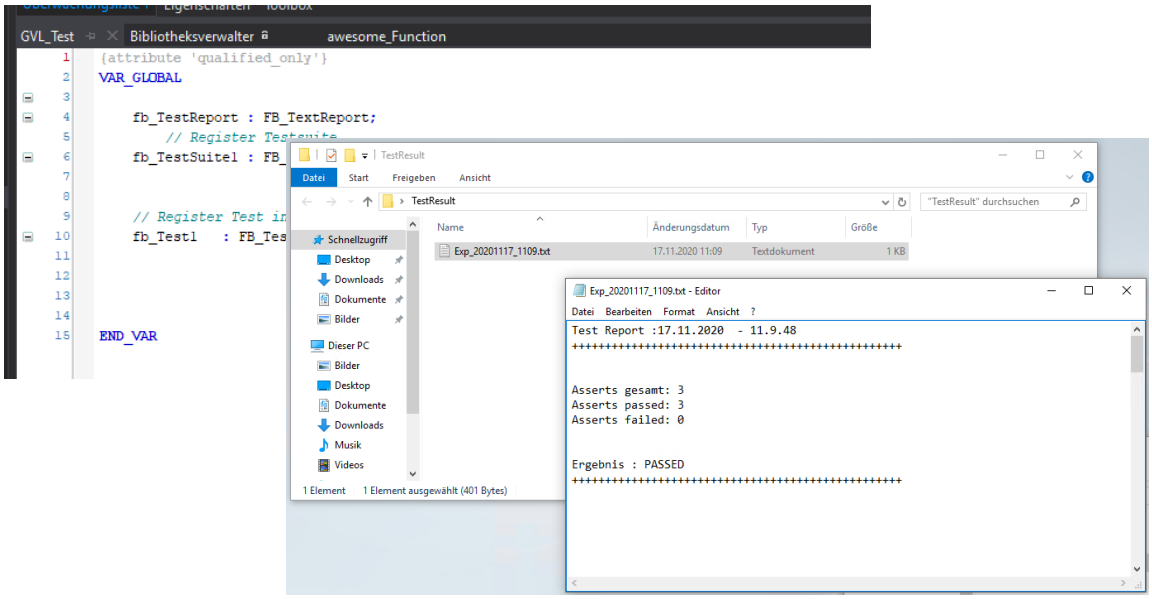


Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

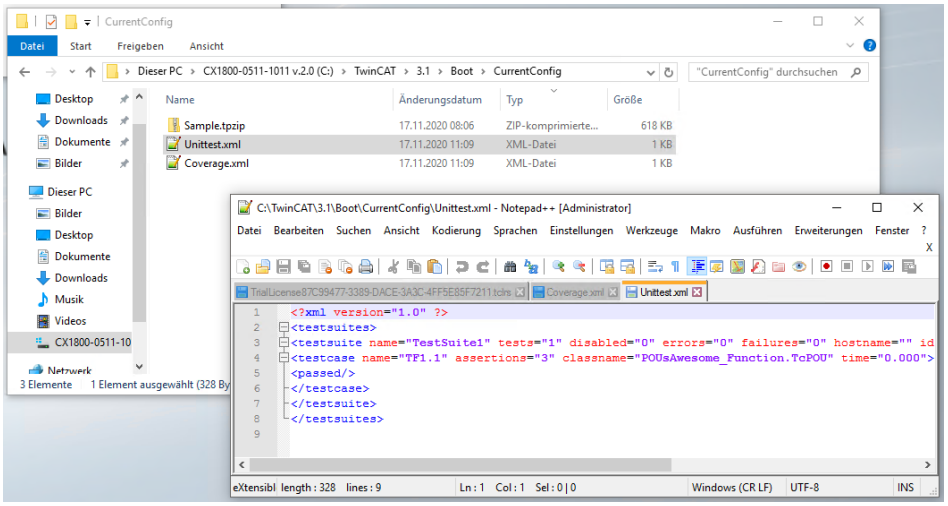
Das Ergebnis wird während dem Testlauf im Statusfenster ausgegeben.



In der Bibliothek befindet sich der Baustein FB_TextReport der beispielhaft einen Auswertung auf dem Zielsystem erstellt sobald dieser instanziert wird. Dafür muss der Ordner *C:\Users\Administrator\Desktop\TestResult* angelegt werden und eine Instanz in der Variablenliste hinzugefügt werden.



zustäzlich wird unter *C:\TwinCAT\3.1\Boot\CurrentConfig* eine Auswertungsdatei im JUnit Format erstellt.



Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Konfiguration des Testlaufs über XML

Diese Vorgehensweise ist natürlich relativ aufwändig und kann durch Verwendung eines Skripts zunächst lokal oder über einen Buildserver sehr einfach automatisiert werden.

Die Konfiguration eines Testobjekts erfolgt durch das einfache Erstellen einer XML Datei die den auszuführenden Testlauf beschreibt. Die Datei ist bewusst einfach gehalten sodass diese ohne Programmierkenntnisse erstellt werden kann.

Beispiel eine XML Konfiguration

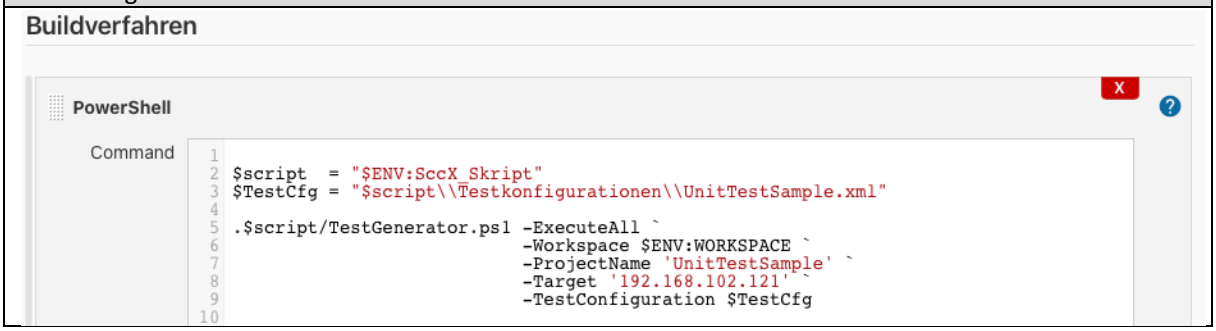
```
<UnitTestLib    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns="http://hoox.software"
                xsi:schemaLocation="http://hoox.software ../Schema/UnitTestSample.xsd">

    <UnitTest>
        <TestSuite Name="TestSuite1" Path="UnitTestSample/UnitTestSample/Sample">
            <TestCase Typ="FB_Test_Awesome_Function" Name="TF1.1" ClassName="Awesome_Function" Path="POUs" Disabled="false"/>
            <TestCase Typ="FB_Test_FB_GreatFunctionblock" Name="TF1.3" ClassName="FB_GreatFunctionblock" Path="POUs" Disabled="fa
            <TestCase Typ="FB_Test_PRG_Main" Name="TF1.2" ClassName="PRG_Main" Path="POUs" Disabled="false"/>
            <TestCase Typ="FB_Test_PRG_Main_FAIL" Name="TF1.4" ClassName="PRG_Main" Path="POUs" Disabled="false"/>
        </TestSuite>
        <TestReport Typ="FB_TextReport"/>
        <!--TestReport Typ="FB_XmlReport"/>-->
    </UnitTest>

</UnitTestLib>
```

Konfiguration im Buildserver

Die Konfiguration im Buildserver beschränkt sich lediglich darauf das Powershell CmdLet mit den entsprechenden Umgebungsvariablen zu "füttern" und damit den gesamten Testlauf anzutriggern. Über verschieden Konfigurationsschalter kann die Anzahl der Schritte eingeschränkt werden oder ein kompletter Testlauf gestartet werden.

Kompletter Testlauf in einzelnen Schritten
- Laden des zu testenden Projekts
- Versuch das Original Projekt zu kompilieren
- Erstellen des Testobjekts : Instrumentieren des Originalprojekts
- Versuch das Testobjekts zu kompilieren
- Ausführen des Testobjekts auf einem Zielsystem
- Starten des Unittests, und Abholen des Ergebnisses

Einstellung Buildserver



Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

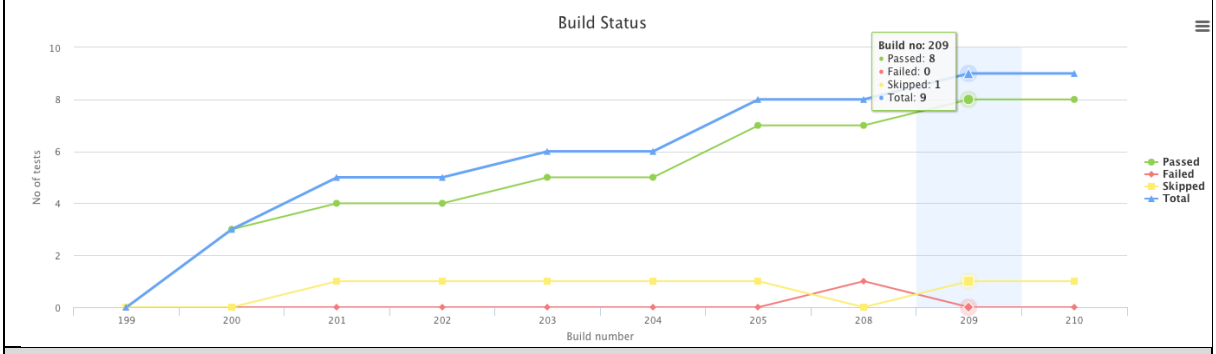
Export als Junit XML

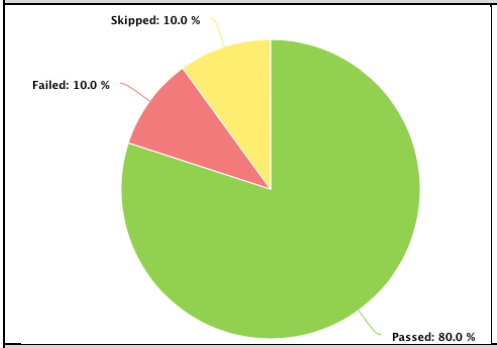
JUnit ist ein allgemein verfügbares Plugin im Jenkins Buildserver das verwendet werden kann um die Ergebnisse der ausgeführten Unittests grafisch darzustellen. Die Unittest Bibliothek erzeugt eine Ausgabedatei die von diesem Plugin gelesen werden kann.

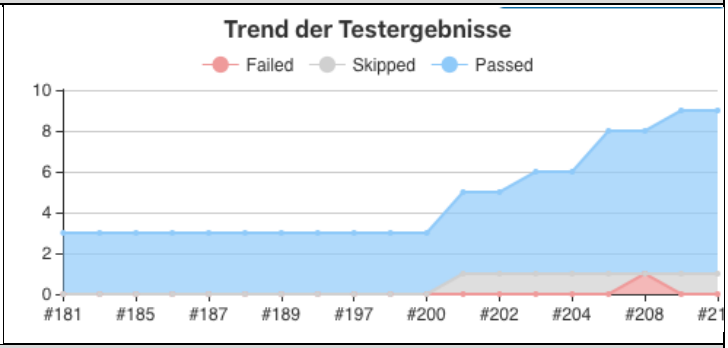
JUnit-Plugin : Tabellarische Darstellung

Chart	Package/Class/Testmethod	Passed	Transitions	210	209	208	205	204	203	202	201	200	199
☐	⊖ POUs	89% (98%)	2	0.700	0.700	0.560	0.560	0.280	0.280	0.140	0.140	0.000	N/A
☐	⊖ Awesome_Function	100% (100%)	0	0.420	0.420	0.420	0.420	0.140	0.140	0.000	0.000	0.000	N/A
☐	TF1.1	100% (100%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF2.2	100% (100%)	0	0.140	0.140	0.140	0.140	0.140	0.140	N/A	N/A	N/A	N/A
☐	TF2.3	100% (100%)	0	0.140	0.140	0.140	0.140	N/A	N/A	N/A	N/A	N/A	N/A
☐	TF2.4	100% (100%)	0	0.140	0.140	0.140	0.140	N/A	N/A	N/A	N/A	N/A	N/A
☐	⊖ FB_GreatFunctionblock	100% (100%)	0	0.140	0.140	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF1.3	100% (100%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF2.5	100% (100%)	0	0.140	0.140	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
☐	⊖ PRG_Main	89% (94%)	2	0.140	0.140	0.140	0.140	0.140	0.140	0.140	0.140	0.000	N/A
☐	TF1.2	100% (100%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF1.4	0% (0%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A	N/A
☐	TF2.1	100% (100%)	0	0.140	0.140	0.140	0.140	0.140	0.140	0.140	0.140	N/A	N/A

JUnit Plugin : Grafische Darstellung





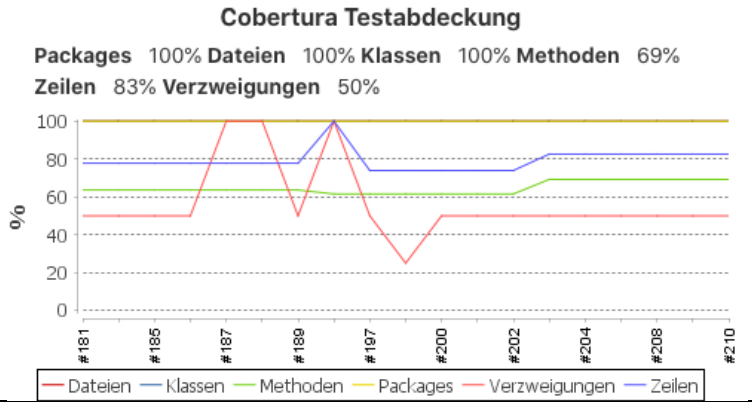


Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Export der Testabdeckung als Cobertura XML

Cobertura ist ein allgemein verfügbares Plugin im Jenkins Buildserver das verwendet werden kann um die Testabdeckung zu berechnen und, noch interessanter, den Quellcode dazu anzuzeigen.
Die Unittest Bibliothek erzeugt eine Ausgabedatei die von diesem Plugin gelesen werden kann.

Cobertura Plugin : Zusammenfassung



Cobertura Plugin : Zusammenfassung

Zusammenfassung der Testabdeckung nach Package

Name	Dateien	Klassen	Methoden	Zeilen	Verzweigungen
POUs	100% 3/3	100% 3/3	64% 7/11	78% 14/18	50% 7/14

Aufschlüsselung der Testabdeckung nach Datei

Name	Klassen	Methoden	Zeilen	Verzweigungen
UnitTestSample/Sample/POUs/FB_GreatFunctionblock.TcPOU	100% 1/1	50% 3/6	73% 8/11	50% 5/10
UnitTestSample/Sample/POUs/PRG_MAIN.TcPOU	100% 1/1	75% 3/4	80% 4/5	50% 1/2
UnitTestSample/Sample/POUs/awesome_Function.TcPOU	100% 1/1	100% 1/1	100% 2/2	50% 1/2

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Die Testabdeckung kann dann bis in auf die Codeebene verfolgt werden.

Cobertura Plugin : Anzeige des Quellcodes

Aufschlüsselung der Testabdeckung nach Klasse

Name	Methoden	Zeilen	Verzweigungen
PRG_MAIN	75% 3/4	80% 4/5	50% 1/2

Quelldatei

UnitTestSample/Sample/POUs/PRG_MAIN.TcPOU

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TcPlcObject Version="1.1.0.1" ProductVersion="3.1.4024.3">
3    <POU Name="PRG_MAIN" Id="{fc5f729b-67e4-44ab-8a6b-4b77bbb61c88}" SpecialFunc="None">
4      <Declaration><![CDATA[(*********************************************************************************
5  Baustein: PRG_MAIN.execute
6  Spezifikation: doc/Spezifikation.pdf, Kap. x.y.z
```

Wurden Codeteile nicht durchlaufen ist dies einfach ersichtlich (rot markierte Zeile).

Zeilen die vom Testlauf erfasst wurden

X_Tc3_Sample › #243

```
26   FUNCTION incredible_Function : DINT
27   VAR_INPUT
28           i_s32Value1 : DINT;
29           i_s32Value2 : DINT;
30           i_s32Value3 : DINT;
31   END_VAR
32   VAR
33   END_VAR]]></Declaration>
34       <Implementation>
35         <ST><![CDATA[
36 4       CASE i_s32Value1 OF
37
38           0..100:
39 1
40                   IF (i_s32value3 <> 0) THEN
41 1
42                           incredible_Function := (i_s32Value1 * i_s32Value2) /
43                                                                   i_s32value3;
44
45                   ELSIF (i_s32value3 > 50) THEN
46 0
47                           incredible_Function := (i_s32Value1 * i_s32Value2) /
48                                                       i_s32value3 * i_s32value3;
49
50                   END_IF;
51
52           101..200:
53 1
54                   IF (i_s32value1 <> 0) THEN
55 1
56                           incredible_Function := (i_s32Value2 * i_s32Value3) /
57                                                                   i_s32value1;
58                   END_IF;
59
60           201..999999:
61 1
62                   IF (i_s32value3 <> 0 and i_s32value1 <> 0) THEN
63 1
64                           incredible_Function :=                 i_s32Value2 /
65                                                     (i_s32value1 * i_s32Value3);
66                   END_IF;
67
68           ELSE
```

Dadurch ist es sehr einfach möglich fehlende Tests zu ermitteln und nachträglich zu implementieren.

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software