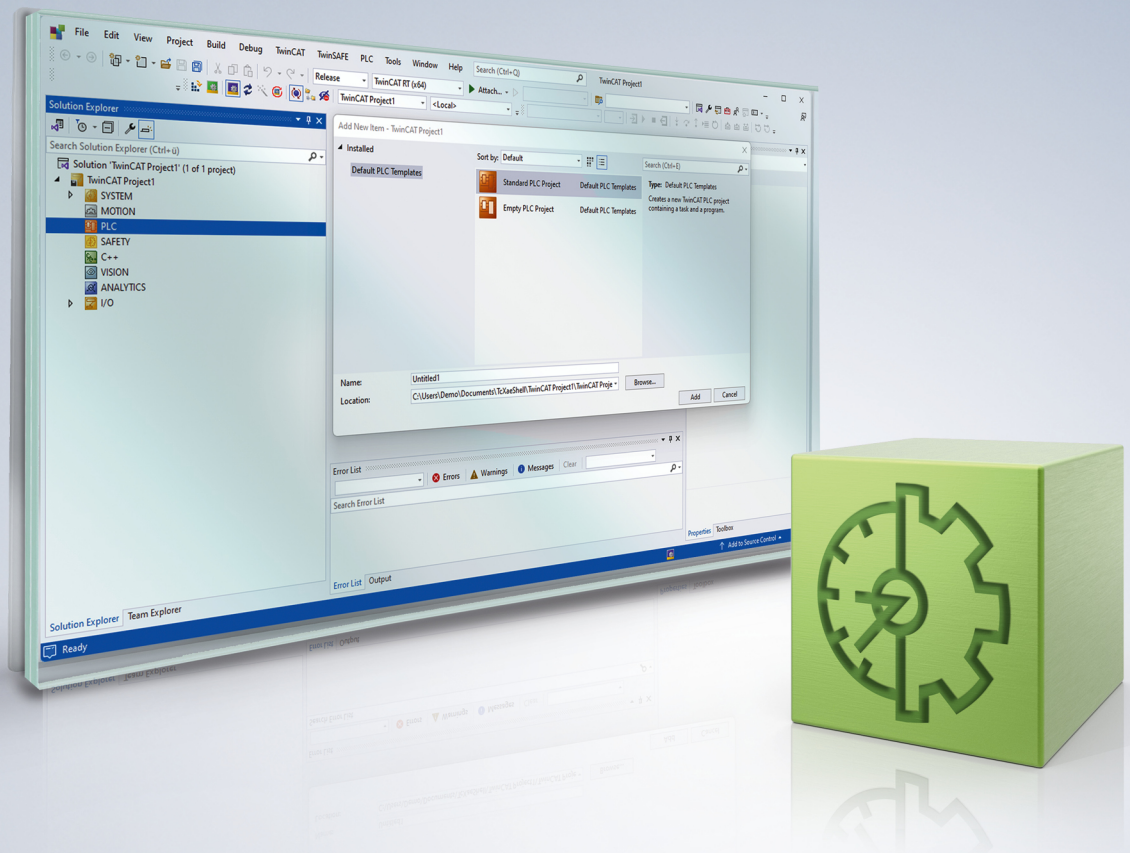
BECKHOFF New Automation Technology

Handbuch | DE

TF1140

TwinCAT 3 | TestFramework



14.07.2026 | Version: 0.1.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® und XTS® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Kennzeichnungen führen.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie, lizenziert durch die Beckhoff Automation GmbH, Deutschland.

Copyright

Fremdmarken

In dieser Dokumentation können Marken Dritter verwendet werden. Die zugehörigen Markenvermerke finden Sie unter: https://www.beckhoff.com/trademarks.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Installation

Systemvoraussetzungen

Technische Daten	Beschreibung
Betriebssystem	Windows 10
Zielplattform	Windows
Minimale TwinCAT-Version	TwinCAT 3.1.4026
TwinCAT-Lizenzen	TF1140 TwinCAT 3 TestFramework

TwinCAT Package Manager: Installation (TwinCAT 3.1 Build 4026)

Eine ausführliche Anleitung zur Installation von Produkten finden Sie im Kapitel TwinCAT-Software verwalten in der Installationsanleitung TwinCAT 3.1 Build 4026.

Installieren Sie den folgenden Workload, um das Produkt nutzen zu können:

- TF1140 | TwinCAT 3 TestFramework

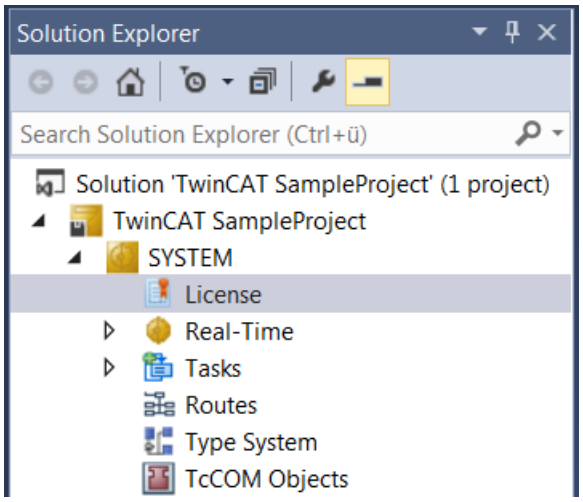
Lizenzierung

Das TF1140 TwinCAT 3 TestFramework bietet Ihnen eine 7-Tage-Testlizenz, mit der Sie auf Ihrem lokalen Engineering-Gerät Ihre Tests erstellen und lokal laufen lassen können. Um die Funktion für automatische Tests, zum Beispiel Tests in einer Pipeline, nutzen zu können, benötigen Sie eine Lizenz für die Vollversion des Produkts. Im Folgenden finden Sie die Erklärungen zu beiden Lizenzierungsarten.

Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

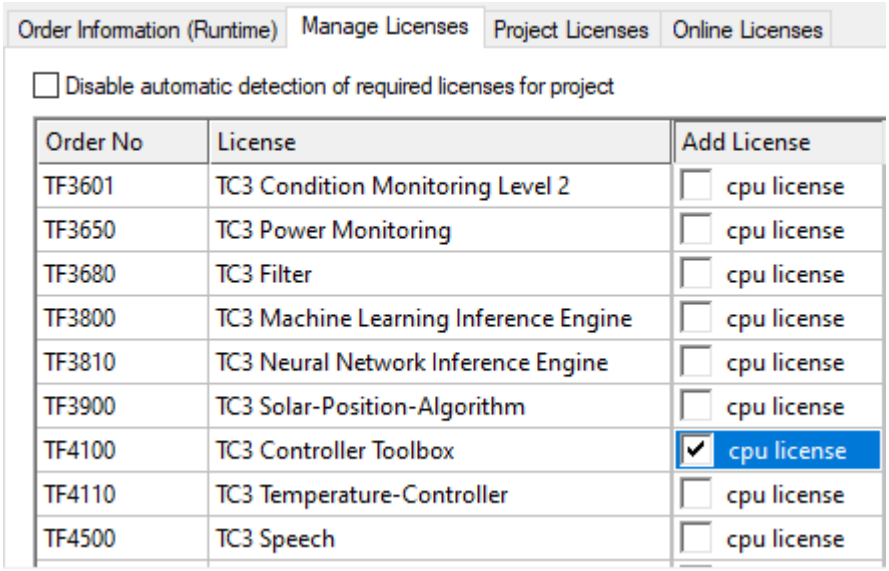
Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.
4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

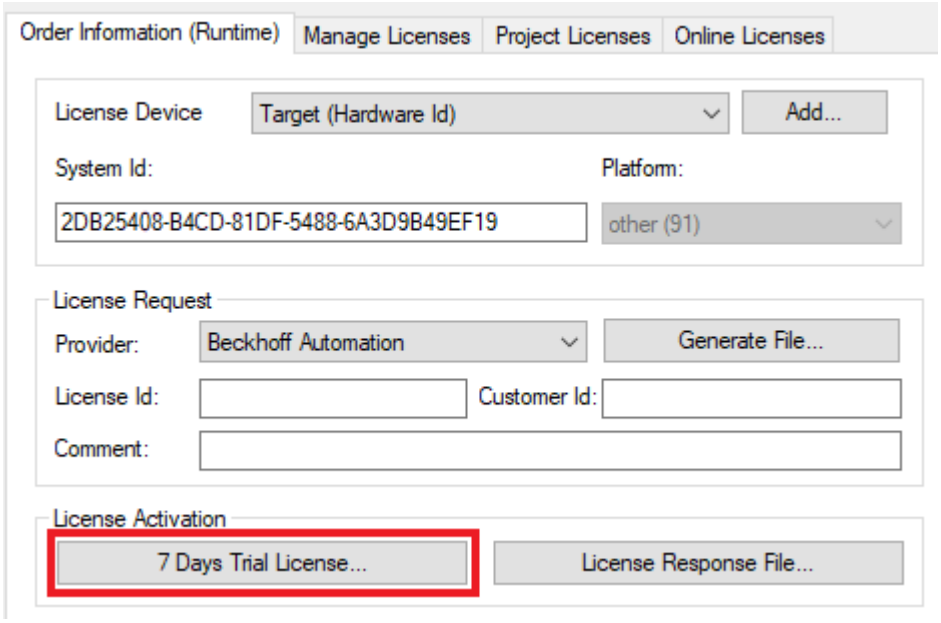
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



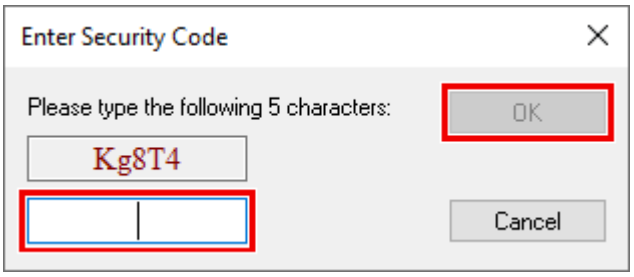
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

   ⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt**.**

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



   ⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.

⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.

10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT-3-Lizenzierung“.

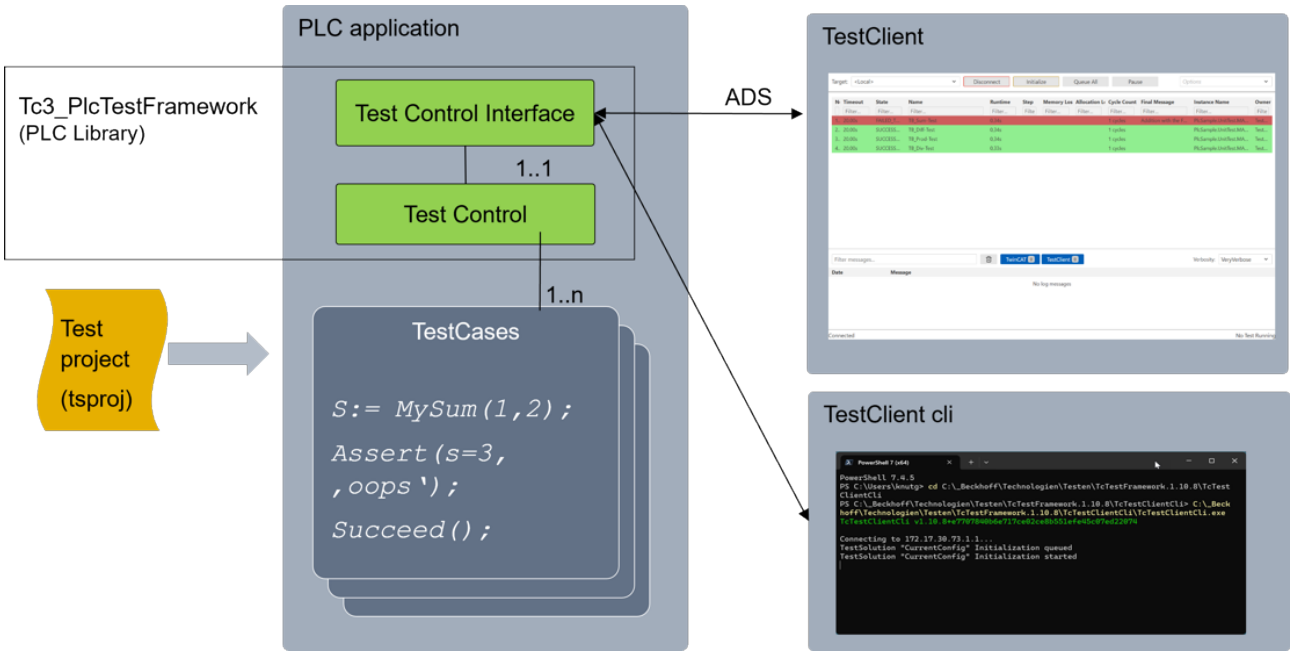
3 Überblick

Testgetriebene Entwicklung (engl. Test Driven Development, TDD) ist eine Entwicklungsmethode von Steuerungscode, bei der vor der eigentlichen Implementierung der Funktionen eines Steuerungscodes Tests geschrieben werden, um diese Funktionen zu testen. Somit beeinflusst die gewählte Implementierung die Erstellung der Testfälle nicht. Die Testfälle werden nach der Implementierung des Steuerungscodes so lange iterativ immer wieder ausgeführt, bis diese nicht mehr fehlschlagen und somit die gewünschte Funktionsweise gegeben ist. Das Ziel dieses Vorgehens ist eine höhere Codequalität.

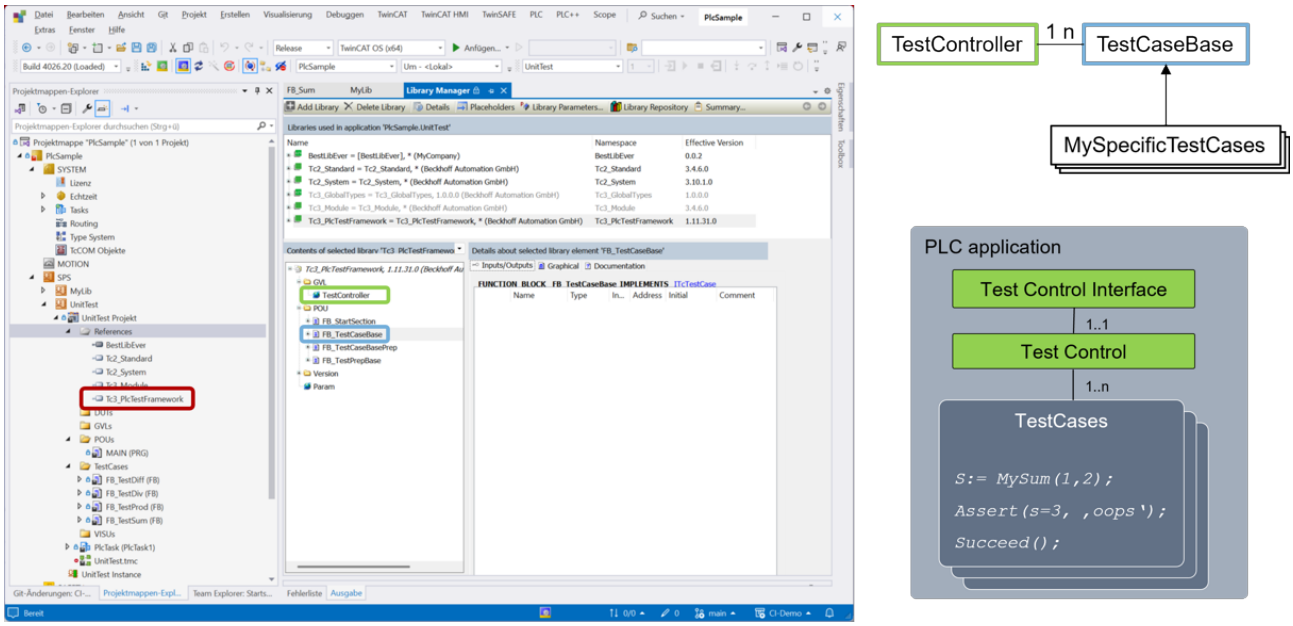
TDD wird häufig mit anderen Methoden der agilen Softwareentwicklung, wie zum Beispiel Continuous Integration, eingesetzt, um neben der Qualität auch die Veränderungsgeschwindigkeit von Code zu erhöhen und die Zeit bis zur Auslieferung neuer Funktionen an den Endkunden zu verringern.

4 Technische Einführung

Das UnitTestFramework besteht aus einer SPS-Bibliothek, welche in das Testprojekt eingebunden werden muss, einem Test-Client mit Benutzeroberfläche und einem Test-Client ohne Benutzeroberfläche. Letzterer wird für Aufrufe per Kommandozeile verwendet und kann verwendet werden, um Tests im Rahmen von CI/CD automatisch in einer Pipline auszuführen. Im folgenden Bild ist der Aufbau schematisch dargestellt.



Liegt der zu testende Steuerungscode in einer SPS-Bibliothek, wird diese ebenfalls in der zu testenden Version in das Testprojekt eingebunden. Dies kann zum Beispiel mittels Automation Interface automatisiert geschehen. Das Testprojekt enthält die Test-Cases für die verschiedenen Bausteine in der Bibliothek. Jeder Test-Case ist dabei ein eigener Funktionsbaustein. In der Basisklasse der Test-Cases ist bereits implementiert, dass sich die Test-Cases bei einem Test-Controller anmelden. Dieser befindet sich in der globalen Variablen-Liste der Test-Framework-Bibliothek (siehe folgende Abbildung).



Sobald sich ein Test-Client mit dem Zielsystem verbindet, auf dem ein Test-Projekt aktiviert wurde, werden automatisch die Testfälle vom Test-Controller ausgelesen und im Test-Client mit GUI als Liste dargestellt. Von hier aus kann nun der Aufruf der Test-Cases als Ganzes oder einzeln erfolgen. Der Test-Client ohne GUI stellt die Test-Cases nicht dar, sondern beginnt nach dem Aufruf direkt mit der Abarbeitung der Test-Fälle. Die Ergebnisse der Tests werden als Datei im JUnit oder HTML-Format hinterlegt. Aus dem Test-Client mit GUI können diese ebenfalls in diesen beiden Formaten exportiert werden.

5 Quickstart

Das folgende Kapitel soll einen leichten Einstieg in die Verwendung des TwinCAT 3 TestFrameworks ermöglichen. Prinzipiell kann das TestFramework auf zwei verschiedene Arten verwendet werden.

- Die Testfälle befinden sich in einem TwinCAT-SPS-Projekt und der zu testende Steuerungs-Code wird als SPS-Bibliothek eingebunden. Diese Verwendung ist für den Test von SPS-Bibliotheken gedacht.
- Die Testfälle und der Steuerungscode, der getestet werden soll, befinden sich im selben SPS-Projekt, wobei z. B. per Compiler-Schalter unterschieden wird, ob die Test-Fälle auch übersetzt werden (siehe Kapitel bedingte Kompilierung/ bedingt Pragmas).

In diesem Quickstart wird auf den ersten Anwendungsfall eingegangen, da dieser vermutlich das häufigere Anwendungsscenario darstellt.

1. Erstellen Sie ein neues TwinCAT-3-Projekt.
2. Erzeugen Sie ein neues SPS-Projekt.
3. Fügen Sie die SPS-Bibliothek Tc3_PlcTestFramework dem SPS-Projekt hinzu.
4. Fügen Sie die SPS-Bibliothek hinzu, die getestet werden soll, sofern die Testfälle nicht direkt im Steuerungscode der Applikation enthalten sind.

Implementierung der Test-Fälle:

Führen Sie die folgenden Schritte für alle Testfälle durch, die Sie implementieren möchten:

5. Erzeugen Sie einen neuen Funktionsbaustein, welcher sich vom Baustein FB_TestCaseBase aus der SPS-Bibliothek Tc3_PlcTestFramework ableitet.
6. Fügen Sie über dem Namen des Funktionsbausteins den Namen des Testfalls als Attribut ein, so wie er im Test-Log bzw. den Test-Clients angezeigt werden soll. Möchten Sie keinen Namen explizit vergeben, wird automatisch der Name der POU als Name des Testfalls verwendet.
7. Fügen Sie optional per Attribut ein Timeout und den Testfall-Owner ein.

   ⇨ Der Deklarationsteil des Funktionsbausteins könnte danach wie folgt aussehen:

```
{ attribute 'Name':='TB_Sum-Test' }
{ attribute 'Timeout':='t#20s' }
{ attribute 'Owner':='TestOwner' }
FUNCTION_BLOCK FB_TestSum EXTENDS FB_TestCaseBase
VAR_INPUT
END_VAR
…
```

8. Deklarieren Sie in der Variablen-Deklaration des Funktionsbausteins den Baustein, den Sie testen wollen
9. Deklarieren Sie zusätzlich Variablen, welche das erwartete Ergebnis des Testfalls und das Ergebnis enthalten werden. Diese explizite Definition ist erforderlich, da die Methode, welche das Ergebnis mit dem Erwartungswert vergleicht, auf dem Any-Datentyp basiert.
10. Implementieren Sie den Testfall. Der Vergleich auf Korrektheit des Ergebnisses erfolgt mit der Methode `AssertEqual`, welche bereits im Basisfunktionsbaustein implementiert ist.

Die Tests werden so lange ausgeführt, bis der Testfall mit der Funktion `Succeded()` verlassen wird oder ein Fehler auftritt. Auf diese Weise ist es möglich, Testfälle so zu gestalten, dass diese iterativ durchlaufen werden.

Der Steuerungscode für den Test eines Summationsbausteins könnte also vereinfacht so aussehen:

```
{ attribute 'Name':='TB_Sum-Test' }
{ attribute 'Timeout':='t#20s' }
{ attribute 'Owner':='TestOwner' }
FUNCTION_BLOCK FB_TestSum EXTENDS FB_TestCaseBase
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    myFB : FB_Sum;
    expectedValue : UINT;
    result: UINT;
```

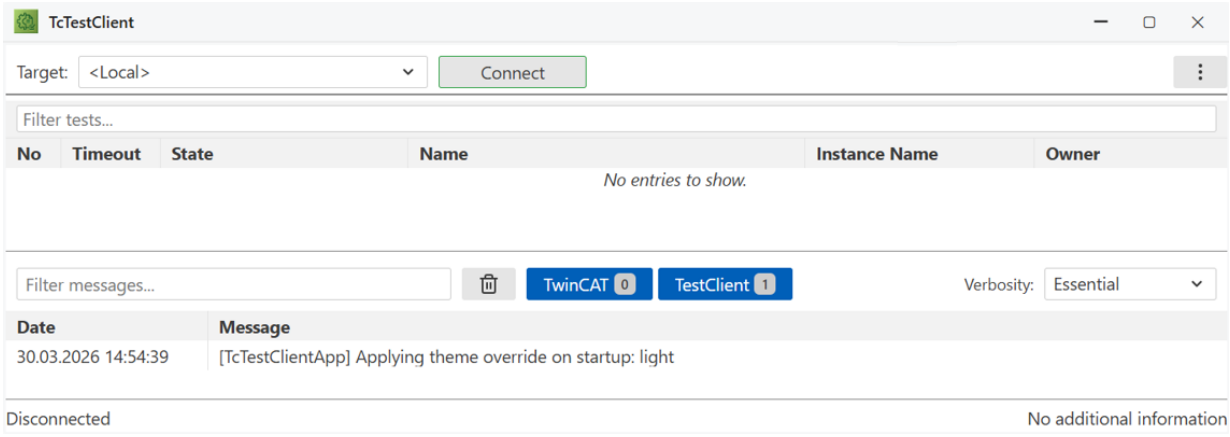
```
END_VAR
// test 2 + 2 = 4
myFB(var1:= 2, var2:= 2, result=> result);

expectedValue:=4;
AssertEqual(expected:= expectedValue, actual:= result, message:= 'Addition with the FB_Sum function
block is incorrect!');
Succeeded();
```

Sind alle Testfälle implementiert, fahren Sie wie folgt fort:

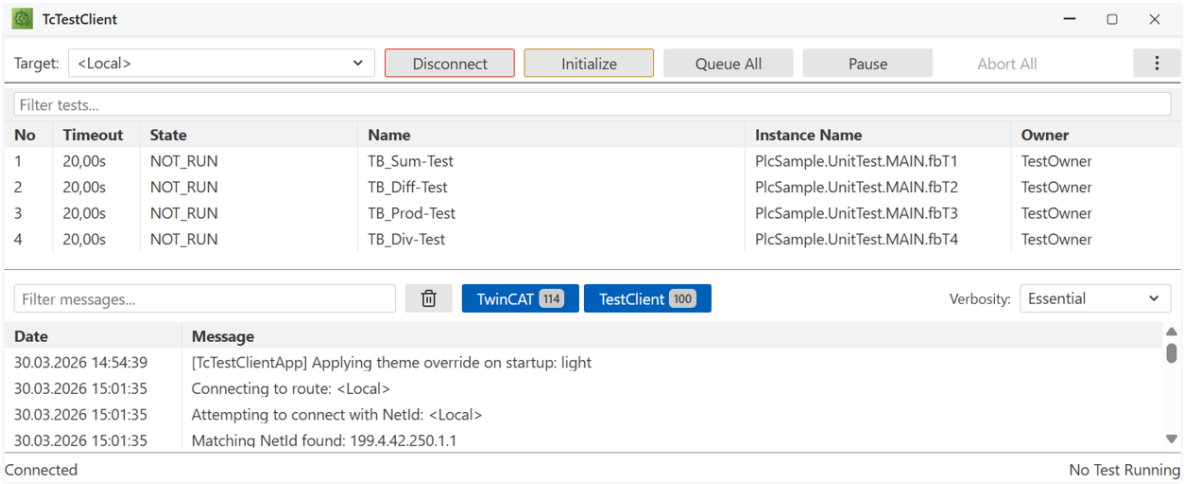
11. Stellen Sie sicher, dass auf dem SPS-Projekt die Option **Autostart Boot Project** aktiv ist.
12. Aktivieren Sie das Projekt auf dem gewünschten Zielsystem.
13. Öffnen bzw. starten Sie einen Test-Client und führen Sie die Tests aus.

Test-Client mit User-Interface:

14. Starten Sie den Test-Client.



15. Wählen Sie mit der Auswahlbox der Option **target** das Zielsystem aus, auf dem das Test-Projekt ausgeführt wird.
16. Betätigen Sie danach die Connect-Taste direkt hinter der Zielsystemauswahl
    ⇨ Es werden im Test-Client alle Testfälle in einer tabellarischen Ansicht dargestellt.



17. Durch Betätigen des Buttons **Queue All** starten Sie die Ausführung aller Testfälle.

⇨ Eine detaillierte Beschreibung der Funktionen der Benutzeroberfläche des Test-Clients finden Sie im Kapitel Referenz, Benutzeroberfläche [▶ 26].

Kommandozeilen-Client:

1. Führen Sie das folgende Kommando per Skript oder per Kommandozeile auf:

```
TcTestClientCli.exe --target 199.4.42.250.1.1 --saveResultsAsJunit "C:
\temp\testJunit.junit.xml" --tcStartupRetries 10 --mode checkMemory
```

Der TestClientCli wird standardmäßig in das *Verzeichnis C:\Program Files (x86)\Beckhoff\TwinCAT\Functions\TE1040-TestFramework\TcTestClientCli* installiert.

- Die Option `--target` verweist dabei auf das Zielsystem, auf welchem die Testfälle ausgeführt werden.
- Die Option `--saveResultAsJunit` aktiviert das Schreiben der Ergebnisse im JUnit-Format und zeigt auf die Datei, in welcher die Ergebnisse gespeichert werden.
- Die Option `--TcStartupRetries` legt die Anzahl der Versuche fest, wie häufig der Test-Client versucht einen Test zu starten.
- Die Option `--mode checkMomory` aktiviert in diesem Fall, dass der Test-Client nach jedem Testdurchlauf den TwinCAT-Router-Speicher überprüft. Auf diese Weise ist es möglich, Memory-Leaks innerhalb der geschriebenen Funktionsbausteine zu finden. Die Ausführung dieser Option geht mit einer geringfügig höheren Laufzeit einher.

6 Kommandozeilen Aufruf

Zur Automatisierung der Testausführung, z. B. in einem CI/CD-Workflow, bietet sich der Kommandozeilen-Client an. Dieser kann wie folgt in einem Skript aufgerufen werden:

```
TcTestClientCli.exe --target 199.4.42.250.1.1 --saveResultsAsJunit C:\temp\testJunit.junit.xml" --
tcStartupRetries 10 --mode checkMemory
```

Dabei verbindet sich der Test-Client mit dem Zielsystem, welches Sie mit der Option `--target` übergeben. Darüber hinaus wird mit dem oben gezeigten Aufruf festgelegt, dass die Ergebnisse im JUnit-Format ausgegeben und unter dem oben angegebenen Pfad gespeichert werden.

Eine ausführliche Liste aller Kommandozeilenparameter und deren Bedeutung können Sie mit dem Befehl

```
TcTestClientCli.exe --help
```

erfragen.

7 SPS API

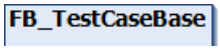
In diesem Kapitel wird die SPS API des TwinCAT 3 TestFrameworks beschrieben. Hierzu gibt es zwei SPS-Bibliotheken, welche zusammen mit dem TF1140 verwendet werden können. Die SPS-Bibliothek Tc3_PlcTestFramework und die Tc3_PlcTestUtilitiesGeneric. Die Erste müssen Sie verpflichtend einbinden. Sie enthält unter anderem die Basisklasse für die Test-Fälle (FB_TestCaseBase).

Die SPS-Bibliothek Tc3_PlcTestUtilitiesGeneric enthält weiterführende Bausteine mit Komfortfunktionen, die ansonsten in der SPS-Applikation ausprogrammiert werden müssten (Zugriff auf die Parameter eines TwinCAT-3-Laufzeit-Moduls, eine Liste mit ADS-Fehlercodes, etc.).

7.1 Tc3_PlcTestFramework

7.1.1 Funktionsbausteine

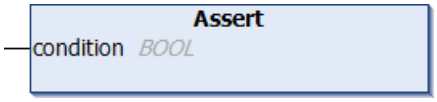
7.1.1.1 FB_TestCaseBase



Der FB_TestCaseBase ist der Basisfunktionsbaustein für neue Testfälle. Dieser Baustein bringt bereits eine Vielzahl an Methoden mit, welche die Verwaltung eines Testfalls betreffen. Dies betrifft zum Beispiel die Methode `FB_init`, in der sich der Baustein automatisch beim Test-Controller anmeldet. Diese Methode müssen und sollten Sie nicht überschreiben, wenn Sie einen neuen Testfall implementieren. Darüber hinaus stehen Methoden u. a. zum Vergleichen der Ergebnisse oder zum Weiterschalten der Test-State-Maschine zur Verfügung. Diese sind im Folgenden beschrieben. Der Basistestfall hat darüber hinaus die folgenden internen Variablen:

Name	Typ	Default-Wert	Bedeutung
Variablen, die per Attribute oder die FB_init übergeben werden.			
sName	STRING(255)	-	Name des Testfalls
Timeout	TIME	0s	Timeout des Testfalls
sOwner	STRING(255)	-	Eigentümer des Testsfalls
Interne Variablen, die über Methoden modifiziert werden können.			
nCurStep	UDINT	0	Statusvariable für eine mögliche Schrittkette
bDone	BOOL	FALSE	Zeigt an, ob die Abarbeitung des Bausteins vollständig ist.
bError	BOOL	TRUE	Fehlerauswertung des Bausteinaufrufes
sFinalMessage	STRING(255)		Meldung, die nach dem Test übergeben wird.
sInstName	STRING(255)		Instanzname des Bausteins (wird automatisch ermittelt)

7.1.1.1.1 Assert



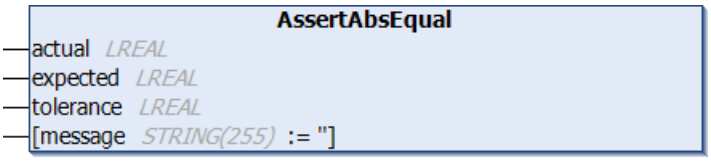
Die `Assert`-Methode kann verwendet werden, um das Fehlschlagen des aktuellen Testfalles zu kennzeichnen. Im Gegensatz zur AssertEqual-Methode [▶ 18], welche selbst das Ergebnis mit dem Erwartungswert vergleicht, muss hier der Vergleich im Applikations-Code des Testfalles durchgeführt werden

und das Ergebnis des Vergleichs als Bool-Wert der Methode mitgegeben werden. Soll zudem eine Fehlermeldung an den aufrufenden Testclient übergeben werden, verwenden Sie die AssertMSG-Methode [▶ 18] .

Beispiel:

```
Assert(condition);
```

7.1.1.1.2 AssertAbsEqual



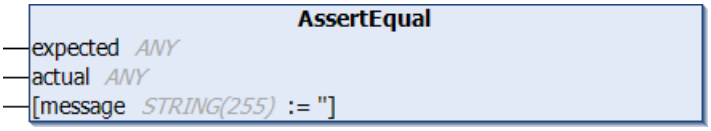
Analog zur `AssertEqual`-Methode wird diese Methode ebenfalls verwendet, um zu prüfen, ob der erwartete Wert eines Testfalls mit dem tatsächlichen Ergebnis übereinstimmt.

Ebenso wird auch hier im Fehlerfall eine Meldung ausgegeben. Allerdings wird bei der `AssertAbsEqual` geprüft, ob der Erwartungswert und das Ergebnis im Betrag (der absolute Wert) innerhalb der Toleranz, die ebenfalls mit angegeben werden kann, nahe genug beieinander liegen.

Beispiel:

```
AssertAbsEqual(actualValue, expectedValue, tolerance);
```

7.1.1.1.3 AssertEqual



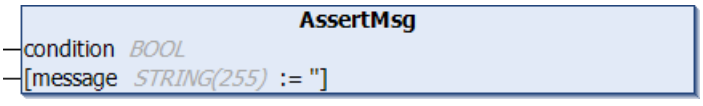
Die `AssertEqual`-Methode kann verwendet werden, um das Ergebnis eines Testfalls zu prüfen und im Fehlerfall eine entsprechende selbst zu definierende Fehlermeldung auszugeben. Der Vergleich des Erwartungswert mit dem Ergebnis des Testfalls findet dabei in der `AssertEqual`-Methode selbst statt. Die `AssertEqual`-Methode basiert auf dem ANY-Datentyp. Da die Übergabe der Parameter aus diesem Grund typisiert erfolgen muss, muss der Erwartungswert vorher einer entsprechenden typisierten Variablen zugewiesen werden und kann nicht direkt im Aufruf der `AssertEqual`-Methode erfolgen.

Beispiel:

```
expectedValue:=<value>;
AssertEqual(expected:= expectedValue, actual:= result, message:= 'Error message to be displayed in
the client!');
```

`Result` ist hierbei das „ausgerechnete“ Ergebnis des Testfalls.

7.1.1.1.4 AssertMsg

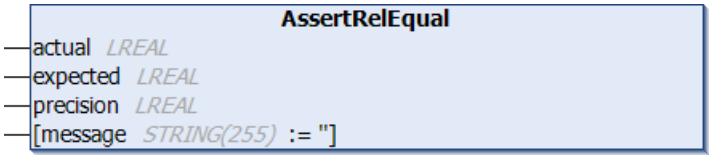


Die `AssertMsg`-Methode kann verwendet werden, um das Ergebnis eines Testfalls zu prüfen und im Fehlerfall eine entsprechende selbst zu definierende Fehlermeldung auszugeben. Sie erweitert die Funktion der `Assert`-Methode also um die Fehlermeldung.

Beispiel:

```
AssertMsg(condition:=condition, message:= 'Error message to be displayed in the client!');
```

7.1.1.1.5 AssertRelEqual

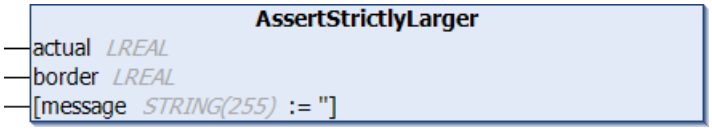


Die Methode `AssertRelEqual` prüft anhand der Methode `IsRelEqual`, ob zwei Gleitkommawerte (Ergebnis des Tests und der Erwartungswert) innerhalb einer definierten relativen bzw. absoluten Toleranz liegen. Ist dies nicht der Fall, schlägt der Testfall fehl und übergibt die angegebene Fehlermeldung.

Beispiel:

```
AssertRelEqual(actual:= result, expected:= expectedValue, precision, message:= 'Error message to be displayed in the client!')
```

7.1.1.1.6 AssertStrictlyLarger

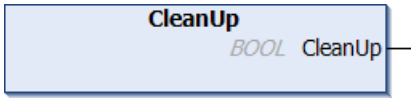


Die Methode `AssertStrictlyLarger` testet, ob ein Wert (Ergebnis eines Tests) strikt größer ist als eine definierte Grenze.

Beispiel:

```
AssertStrictlyLarger (actual:= result, Border:= BorderValue, message:= 'Error message to be displayed in the client!')
```

7.1.1.1.7 Cleanup



`Cleanup` ist eine Methode, die für das Aufräumen der Daten eines Testfalles verwendet werden kann. Diese Methode müssen Sie überschreiben und implementieren, wenn sie benötigt wird.

7.1.1.1.8 GetCurrentStep

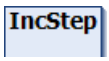
Wie bereits im Kapitel FB_TestCaseBase [▶ 17] beschrieben, gibt es interne Variablen, die bei der Implementierung der TestCases verwendet werden können. Unter anderem die Variable `nCurStep` für die Verwendung als Statusvariable eines Case-Konstrukts, mit dessen Hilfe Testfälle in Form einer State Machine strukturiert werden können. Die Methode `GetCurrentStep` gibt den aktuellen Wert der Statusvariable `nCurStep` aus. Soll eine andere Status-Variable verwendet werden, können Sie diese Methde überschreiben.

7.1.1.1.9 GoToStep

Wie bereits im Kapitel FB_TestCaseBase [▶ 17] beschrieben, gibt es interne Variablen, die bei der Implementierung der TestCases verwendet werden können. Unter anderem die Variable `nCurStep` für die Verwendung als Statusvariable eines Case-Konstrukts, mit dessen Hilfe Testfälle in Form einer State Machine strukturiert werden können. Die Methode `GoToStep`, setzt den Wert der Statusvariable `nCurStep`

auf den übergebenen Wert. Somit springen Sie im nächsten Durchlauf des Case-Konstrukts in genau diesen Schritt. Soll eine andere Statusvariable verwendet werden, muss diese Methode überschrieben werden, wenn sie gebraucht wird.

7.1.1.1.10 IncStep



Wie bereits im Kapitel FB_TestCaseBase [▶ 17] beschrieben, gibt es interne Variablen, die bei der Implementierung der TestCases verwendet werden können. Unter anderem die Variable `nCurStep` für die Verwendung als Statusvariable eines Case-Konstrukts, mit dessen Hilfe Testfälle in Form einer State Machine strukturiert werden können. Die Methode IncStep, inkrementiert den Wert der Variable `nCurStep` um 1. Soll eine andere Statusvariable verwendet werden, muss diese Methode überschrieben werden, wenn sie verwendet werden soll.

7.1.1.1.11 IncStepIf

Wie die Methode IncStep [▶ 20], inkrementiert diese Methode die interne Statusvariable des Case-Konstruktes um 1, wenn eine Bedingung erfüllt ist. Die Prüfung der Bedingung selbst erfolgt im Applikationscode und das Boolsche Ergebnis müssen Sie beim Aufruf der Methode mit übergeben. Soll eine andere Statusvariable als `nCurStep` verwendet werden, muss diese Methode überschrieben werden, wenn sie Anwendung finden soll.

Beispiel

```
IncStepIf(Condition);
```

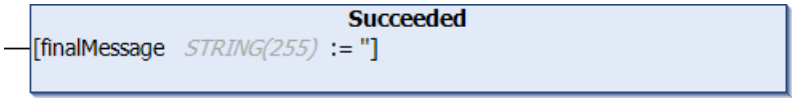
7.1.1.1.12 IsRelEqual

Die Methode `IsRelEqual` prüft, ob zwei Gleitkommazahlen vom Typ LREAL innerhalb einer Toleranz liegen. Wenn beide Werte größer sind als der Wert in `absCompScale` angegeben, wird die relative Abweichung zwischen beide Werten anhand der Toleranz überprüft. Ist einer der Werte unterhalb der Toleranz findet ein absoluter Vergleich statt. Die Methode findet unter anderem Verwendung in der Methode AssertRelEqual [▶ 19].

Beispiel:

```
IsRelEqual (actual:= result, expected:= expectedValue, precision)
```

7.1.1.1.13 Succeeded

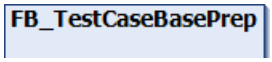


Die Methode `succeeded` wird aufgerufen, um zu bestätigen, dass der Testfall erfolgreich ausgeführt wurde. Der Testfall wird so lange ausgeführt, bis entweder eine `Assert`-Methode das Fehlschlagen des Testfalls feststellt, der eingestellte Timeout die Abarbeitung des Testfalls stoppt, oder durch den Aufruf von `succeeded` bestätigt wird, dass der Testfall fehlerfrei durchgelaufen ist.

Beispiel:

```
succeeded();
```

7.1.1.2 FB_TestCaseBasePrep



FB_TestCaseBasePrep ist eine Erweiterung von FB_TestCaseBase [▶ 17]. Bei der Instanziierung dieses Funktionsbausteins können Sie einen Funktionsbaustein des Typs FB_TestPrepBase [▶ 21] angeben. Dadurch ist es möglich, die Schritte in einen separaten Funktionsbaustein auszulagern, um den Arbeitspunkt eines Testfalls zu erreichen. Dieser bietet die nahezu dieselben Methoden zum Detektieren von Fehlern, um auszuwerten, ob bereits auf dem Weg zum Arbeitspunkt eines Testfalls Fehler auftreten.

Beispiel der Verwendung bzw. Zuordnung eines Vorbereitungsschritts zu einem Testfall:

Deklaration des Testfalls anhand der erweiterten Testfallklasse FB_TestCaseBasePrep:

```
FUNCTION_BLOCK FB_TestCase1 EXTENDS FB_TestCaseBasePre
…
```

Deklaration des Vorbereitungsschritts:

```
FUNCTION_BLOCK FB_ TestCasePrep1 EXTENDS FB_TestPrepBase
…
```

Instanziierung im Deklarationsteil der Main-POU des Testprogramms:

```
    fbPreperation    : FB_TestCasePrep1;
    fbTestCase       : FB_TestCase1(fbPreperation);
```

7.1.1.2.1 FB_Init

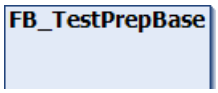
Die FB_Init Methode ist eine Art Konstruktor eines Funktionsbausteins. Zusätzliche Variablen, die darin angegeben werden, müssen bei der Deklaration einer Funktionsbaustein-Instanz dieses Types gesetzt werden.
Im Falle des Funktionsbausteins FB_TestCaseBasePrep [▶ 21] ist dies ein Interface-Pointer des Types ITcTestPreparation. Da jeder Funktionsbaustein zur Testvorbereitung dieses Interface automatisch über die Ableitung vom Funktionsbaustein FB_TestPrepBase [▶ 21] verfügt, kann bei der Instanziierung eines FB_TestCaseBasePrep [▶ 21]-Funktionsbausteins der benötigte Vorbereitungsschritt mit angegeben werden.

Instanziierung im Deklarationsteil der Main-POU des Testprogramms:

```
   fbPreperation    : FB_TestCasePrep1;
   fbTestCase       : FB_TestCase1(fbPreperation);
```

Verändern Sie diese Methode nur mit Bedacht, um die Funktionsweise des Testframeworks nicht negativ zu beeinflussen.

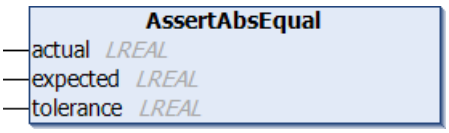
7.1.1.3 FB_TestPrepBase



Funktionsbaustein für die Vorbereitungsschritte eines Testfalls. Dieser Baustein kann verwendet werden, um die Schritte zum Erreichen des Arbeitspunkts eines Testfalls in einen separaten Funktionsbaustein auszulagern. Das hat den Vorteil, dass diese Schritte von verschiedenen Testfällen wiederverwendet werden können. Auch der Vorbereitungsfunktionsbaustein verfügt über die entsprechenden Methoden, um Fehler bei der Ausführung der Vorbereitungsschritte zu detektieren. Der Basisvorbereitungsschritt hat die folgenden internen Variablen, die über die im Folgenden beschriebenen Methoden verändert werden können.

Name	Typ	Default-Wert	Bedeutung
nCurStep	UDINT	0	Statusvariable für eine mögliche Schrittkette
bDone	BOOL	FALSE	Zeigt an, ob die Abarbeitung des Bausteins vollständig ist.
bError	BOOL	TRUE	Fehlerauswertung des Bausteinaufrufs
sFinalMessage	STRING(255)		Meldung, die nach dem Test übergeben wird.

7.1.1.3.1 AssertAbsEqual



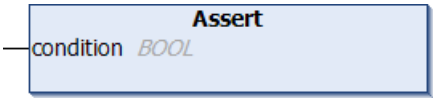
Analog zur `AssertEqual`-Methode wird diese Methode ebenfalls verwendet, um zu prüfen, ob der erwartete Wert eines Testvorbereitungsschritts mit dem tatsächlichen Ergebnis übereinstimmt.

Ebenso wird auch hier im Fehlerfall eine Meldung ausgegeben. Allerdings wird bei der AssertAbsEqual geprüft, ob der Erwartungswert und das Ergebnis im Betrag (vom absoluten Wert her) nahe genug beieinander liegen. Der Betrag muss innerhalb der Toleranz liegen, die Sie ebenfalls angeben können.

Beispiel:

```
AssertAbsEqual(actualValue, expectedValue, tolerance);
```

7.1.1.3.2 Assert

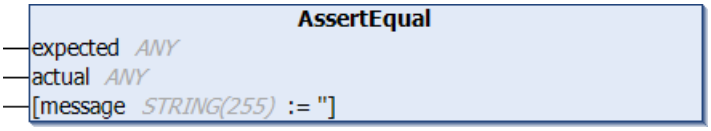


Die `Assert`-Methode kann verwendet werden, um das Fehlschlagen eines der Vorbereitungsschritte des folgenden Testfalls zu kennzeichnen. Im Gegensatz zur AssertEqual-Methode [▶ 22], welche selbst das Ergebnis mit dem Erwartungswert vergleicht, muss hier der Vergleich im Applikationscode des Vorbereitungsschritts durchgeführt werden und das Ergebnis des Vergleichs als Bool-Wert der Methode mitgegeben werden. Soll zudem eine Fehlermeldung an den aufrufenden Testclient übergeben werden, sollte die AssertMSG-Methode [▶ 23] verwendet werden.

Beispiel:

```
Assert(condition);
```

7.1.1.3.3 AssertEqual



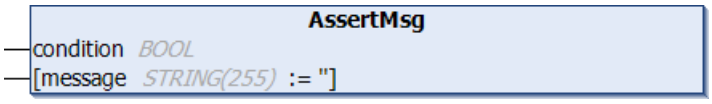
Die `AssertEqual`-Methode kann verwendet werden, um das Ergebnis eines Testvorbereitungsschrittes zu prüfen und im Fehlerfall eine entsprechende selbst zu definierende Fehlermeldung auszugeben. Der Vergleich des Erwartungswert mit dem Ergebnis des Vorbereitungsschrittes findet dabei in der AssertEqual-Methode selbst statt. Die AssertEqual-Methode basiert auf dem ANY-Datentyp. Da die übergabe der Parameter aus diesem Grund typisiert erfolgen muss, muss der Erwartungswert vorher einer entprechenden typisierten Variablen zugewiesen werden und kann nicht direkt im Aufruf der AssertEqual-Methode erfolgen.

Beispiel:

```
expectedValue:=<value>;
AssertEqual(expected:= expectedValue, actual:= result, message:= 'Error message to be displayed in
the client!');
```

`Result` ist hierbei das „ausgerechnete“ Ergebnis des Testvorbereitungsschritts.

7.1.1.3.4 AssertMsg

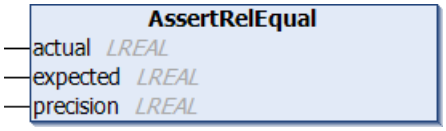


Die `AssertMsg`-Methode kann verwendet werden, um das Ergebnis eines Testvorbereitungsschrittes zu prüfen und im Fehlerfall eine entsprechende selbst zu definierende Fehlermeldung auszugeben. Sie erweitert die Funktion der Assert-Methode also um die Fehlermeldung.

Beispiel:

```
AssertMsg(condition:=condition, message:= 'Error message to be displayed in the client!');
```

7.1.1.3.5 AssertRelEqual



Die Methode `AssertRelEqual` prüft anhand der Methode *IsRelEqual*, ob zwei Gleitkommawerte (Ergebnis des Vorbereitungsschrittes und der Erwartungswert) innerhalb einer definierten relativen bzw. absoluten Toleranz liegen. Ist dies nicht der Fall, schlägt der Testfall fehl und übergibt die angegebene Fehlermeldung.

Beispiel:

```
AssertRelEqual(actual:= result, expected:= expectedValue, precision, message:= 'Error message to be
displayed in the client!')
```

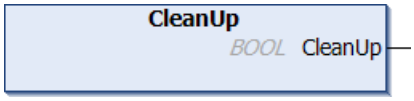
7.1.1.3.6 AssertStrictlyLarger

Die Methode `AssertStrictlyLarger` testet, ob ein Wert (Ergebnis eines Tests) strikt größer ist als eine definierte Grenze.

Beispiel:

```
AssertStrictlyLarger (actual:= result, Border:= BorderValue, message:= 'Error message to be
displayed in the client!')
```

7.1.1.3.7 Cleanup



`Cleanup` ist eine Methode, die für das „Aufräumen“ der Daten eines Vorbereitungsschritts verwendet werden kann, sofern diese nicht für den Testfall selbst benötigt werden. Diese Methode müssen Sie überschreiben und implementieren, wenn sie benötigt wird.

7.1.1.3.8 GetCurrentStep

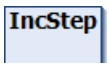
Wie bereits im Kapitel FB_TestPrepBase [▶ 21] beschrieben, gibt es interne Variablen, die bei der Implementierung der Vorbereitungsschritte verwendet werden können. Unter anderem die Variable `nCurStep` für die Verwendung als Statusvariable eines Case-Konstrukts, mit dessen Hilfe die State

Machine zum Erreichen des Arbeitspunkts eines Testfalls strukturiert werden kann. Die Methode `GetCurrentStep` gibt den aktuellen Wert der Statusvariable `nCurStep` aus. Soll eine andere Statusvariable verwendet werden, kann diese Methode überschrieben werden.

7.1.1.3.9 GoToStep

Wie bereits im Kapitel FB_TestPrepBase [▶ 21] beschrieben, gibt es interne Variablen, die bei der Implementierung der Vorbereitungsschrittes verwendet werden können. Unter anderem die Variable `nCurStep` für die Verwendung als Statusvariable eines Case-Konstrukts, mit dessen Hilfe die State Machine zum Erreichen des Arbeitspunkts eines Testfalls strukturiert werden kann. Die Methode `GoToStep` setzt den Wert der Statusvariable `nCurStep` auf den übergebenen Wert. Somit springt man im nächsten Durchlauf des Case-Konstrukts in genau diesen Schritt. Soll eine andere Statusvariable verwendet werden, muss diese Methode überschrieben werden, wenn sie gebraucht wird.

7.1.1.3.10 IncStep



Wie bereits im Kapitel FB_TestPrepBase [▶ 21] beschrieben, gibt es interne Variablen, die bei der Implementierung der Vorbereitungsschritte verwendet werden können. Unter anderem die Variable `nCurStep` für die Verwendung als Statusvariable eines Case-Konstrukts, mit dessen Hilfe die Vorbereitungsschritte in Form einer State Machine strukturiert werden können. Die Methode `IncStep` inkremtiert den Wert der Variable `nCurStep` um 1. Soll eine andere Statusvariable verwendet werden, muss diese Methode überschrieben werden, wenn sie verwendet werden soll.

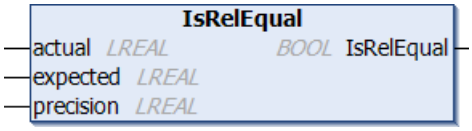
7.1.1.3.11 IncStepIf

Wie die Methode IncStep [▶ 24], inkrementiert diese Methode die interne Statusvariable des Case-Konstrukts um 1, wenn eine Bedingung erfüllt ist. Die Prüfung der Bedingung selbst erfolgt im Applikationscode und das Boolesche Ergebnis müssen Sie beim Aufruf der Methode mit übergeben. Soll eine andere Statusvariable als `nCurStep` verwendet werden, muss diese Methode überschrieben werden, wenn sie Anwendung finden soll.

Beispiel

```
IncStepIf(Condition);
```

7.1.1.3.12 IsRelEqual



Die Methode `IsRelEqual` prüft, ob zwei Gleitkommazahlen vom Typ LREAL innerhalb einer Toleranz liegen.

Wenn beide Werte größer sind als der Wert angegeben in `absCompScale`, wird die relative Abweichung zwischen beide Werten anhand der Toleranz überprüft. Ist einer der Werte unterhalb der Toleranz, findet ein absoluter Vergleich statt. Die Methode findet unter anderem Verwendung in der Methode AssertRelEqual [▶ 23].

Beispiel:

```
IsRelEqual (actual:= result, expected:= expectedValue, precision)
```

7.1.1.3.13 SucceededPreparation

SucceededPreparation

Die Methode `SucceededPreparation` wird aufgerufen, um zu bestätigen, dass die Testfallvorbereitung erfolgreich abgeschlossen wurde. Die Vorbereitungsschritte werden so lange ausgeführt, bis entweder eine `Assert`-Methode das Fehlschlagen des Vorbereitungsschritts feststellt, der eingestellte Time-Out die Abarbeitung der Testfallvorbereitung stoppt, oder durch den Aufruf von `SucceededPreparation` bestätigt wird, dass die Testfallvorbereitung fehlerfrei durchgelaufen ist und der Testfall nun aufgerufen werden kann.

Beispiel:

```
SucceededPreparation ();
```

7.1.2 GVL

7.1.2.1 TestController

Die Globale Variablenliste TestController enthält lediglich die Instanz des Funktionsbausteins TestCtrl. Dieser Funktionsbaustein stellt das Interface zu den TestClients zur Verfügung und muss in der Main POU des Testprojekts aufgerufen werden:

```
// Calling the TestController (which is located in the library and acts as the interface to the clients)
Tc3_PlcTestFramework.TestController.TestCtrl();
```

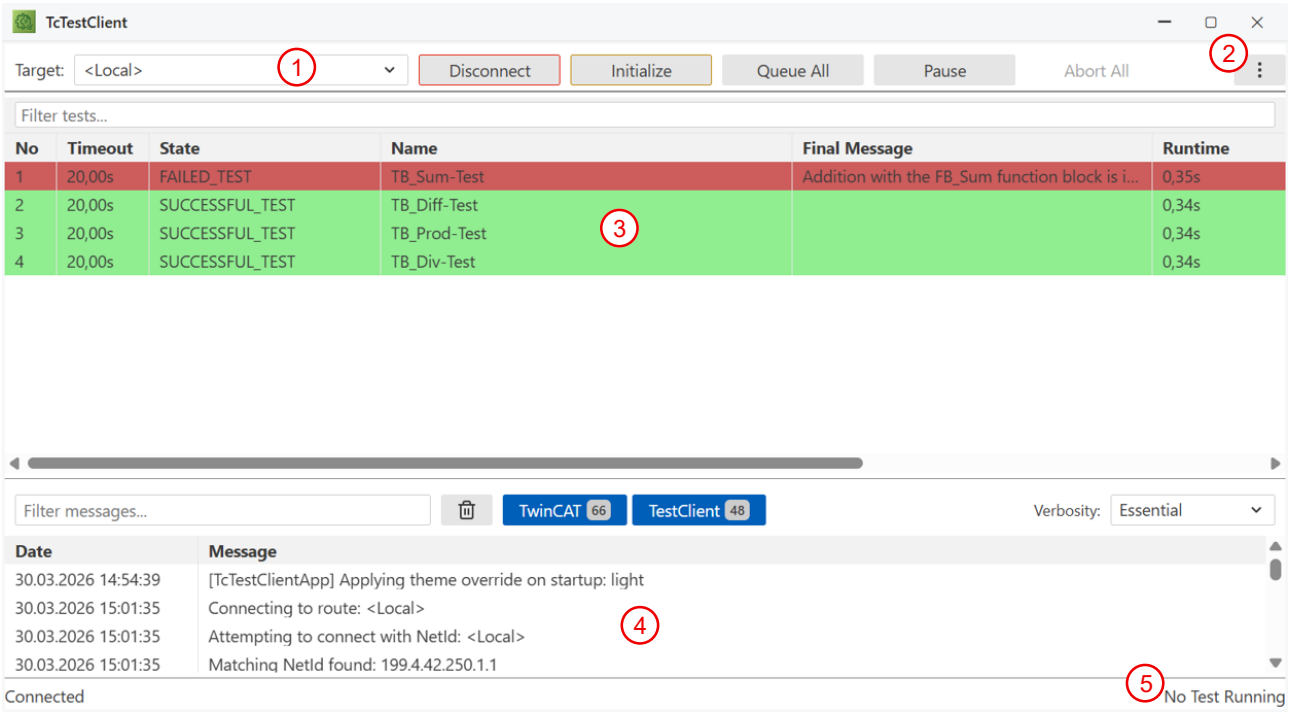
7.1.3 Parameter

Sie können für die Tc3_PlcTestFramework-Bibliothek die folgenden 3 Parameter setzen:

MaxTestCaseCount	Wie bereits im Kapitel Einführung [▶ 12] beschrieben, werden alle Testfälle automatisch beim Test-Controller angemeldet. Hier kann die Größe der internen Liste angepasst werden. Der Parameter ist vom Typ UDINT.
DefaultTimeout	Hier kann die Default-Timeout-Zeit für die Ausführung der Testfälle eingetragen werden. Wird ein Testfall nicht innerhalb dieser Zeit ausgeführt, wird er als fehlgeschlagen markiert und der nächste gestartet.
DefaultOwner	Default-Owner, der im grafischen Client bzw. in der Auswertung als JUnit oder HTML standardmäßig angezeigt wird, wenn er nicht für Testfälle explizit definiert wurde.

8 Referenz, Benutzeroberfläche

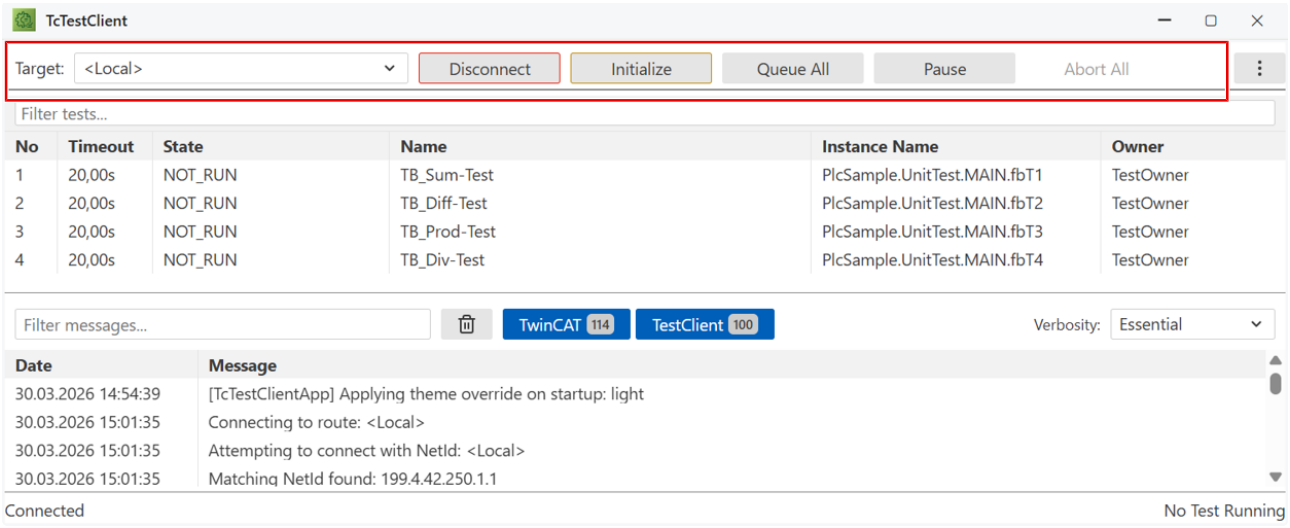
Die Benutzeroberfläche des TcTestClient mit UI besteht aus den folgenden Komponenten:



1. Toolbar mit den wichtigsten Schaltflächen
2. Menü Optionen
3. Test-Case-Fenster
4. Ausgabefenster
5. Statusanzeige

8.1 Toolbar

In der Toolbar finden Sie alle wichtigen Schaltflächen.

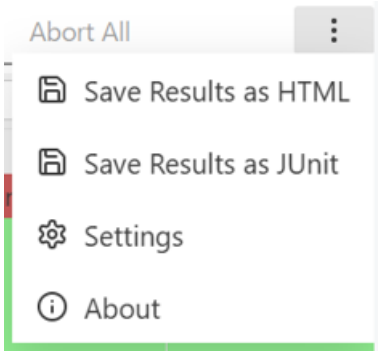


Schaltfläche	Beschreibung
Target	
Connect/ Disconnect	
Initialize	
Queue all	
Pause	
Abort all	

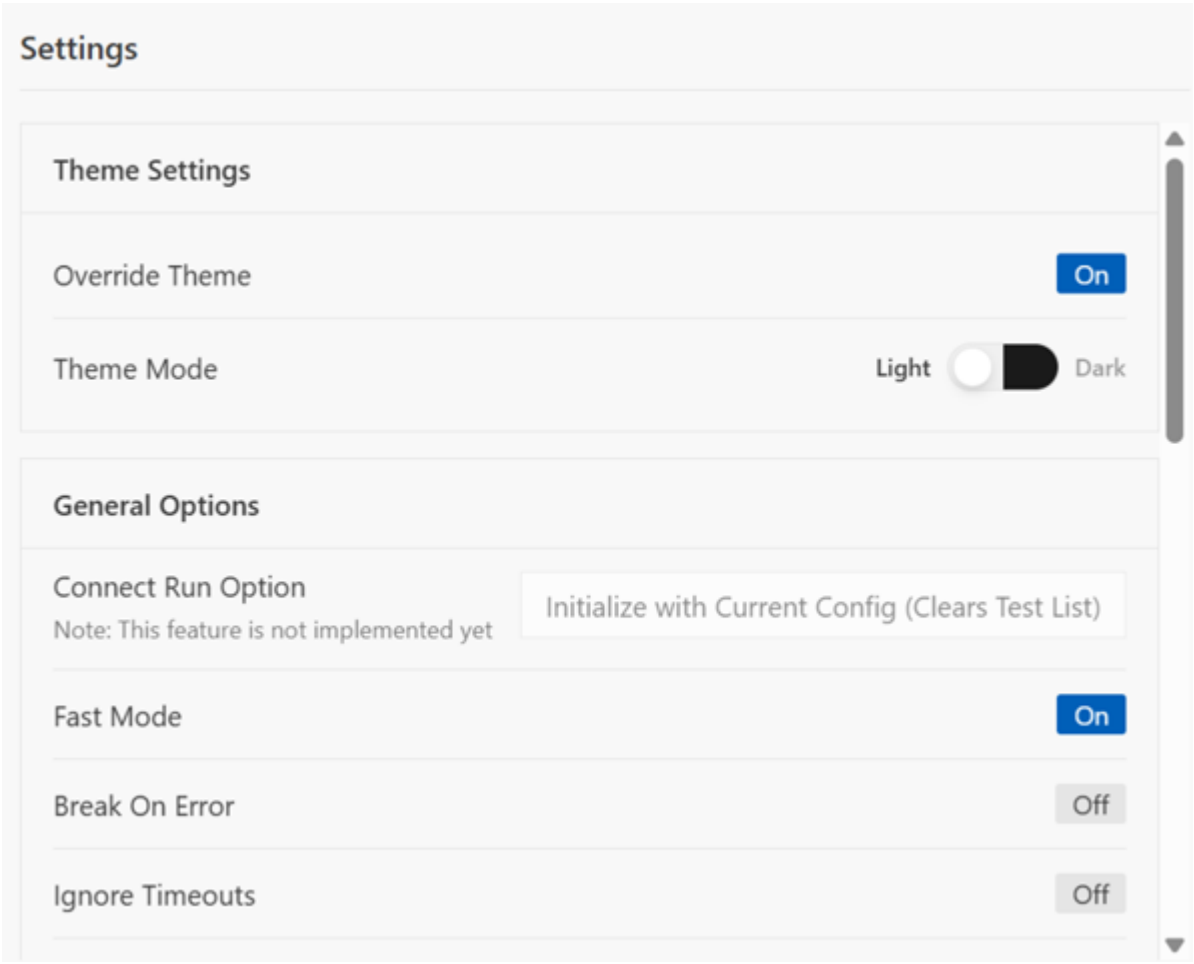
8.2 Settings Menü

Im Menü Settings sind alle Einstellungen enthalten, die im UI-Test-Client gesetzt werden können. Um Das Menü zu öffnen, gehen Sie wie folgt vor:

1. Klicken Sie per Rechtsklick auf das Menü-Icon (…).
2. Wählen Sie nun die Option Settings aus.



⇨ Das Menü Settings öffnet sich.

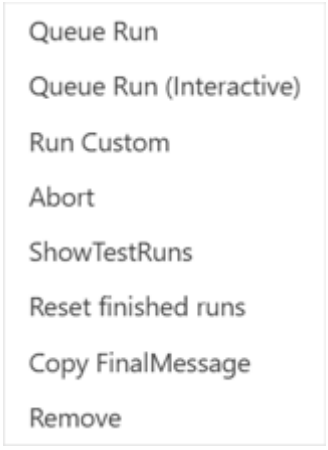


Theme Settings:	
Override Theme	Erlaubt es die Theme-Einstellungen vom System zu ignorieren. Aktiviert die Einstellung Theme Mode.
Theme Mode	Schaltet zwischen hellem und dunklem Theme um.
Generelle Optionen	
Fast Mode	
Break On Error	
Ingnore Timeouts	
Stop Iterating on first Failure	
Requeue Run on Exception	
Don’t Show Test Runs After Crash	
Memory Seetings:	
Check Router Memory	
Check Allocation Count	
Automatic Restart if Memory is Low	
Don’t Resart if Memory is Low	
Rounter Memory Limit (bytes)	
Timeout & Build Settings	
TwinCAT Startup Timeout (seconds)	
TestControl Timeout (Seconds)	
Increase Timeout By (milliseconds)	
TwinCAT Startup Retries	
Number of Parallel Builds	

8.3 Test-Case-Fenster

Nach dem Verbinden mit dem Zielsystem, werden alle Testfälle, die am Test-Controller angemeldet sind, in der Liste angezeigt.

1. Klicken Sie mit der rechten Maustaste auf einen Testfall im Test-Case-Fenster, um das Kontextmenü des Testfalls zu öffnen.



⇨ Das Kontext-Menü der einzelnen Testfälle enthält die oben gezeigten Optionen.

Option	Beschreibung
Queue Run	Startet die Testausführung des ausgewählten Testfalles.
Queue Run (Interactive)	Die Ausführung des ausgewählten Testfalles erfolgt interaktiv. Es wird abgefragt, ob Sie nach dem Starten der Laufzeit den Test starten wollen. Nach dem Test erfolgt die Abfrage, ob Sie wieder in den Konfig-Mode wechseln möchten. Dies dient dazu, die Möglichkeit zu besitzen, ein Scope entsprechend zu konfigurieren und zu starten, oder weitere Schritte auszuführen, die vor dem Start eines Tests wichtig sind, die aber nicht innerhalb des Tests ausprogrammiert werden konnten.
Run Custom	Untermenü, in dem es möglich ist, mehrere Runs desselben Testfalls anzustoßen.
Abort	Schließt das Kontextmenü. Dasselbe Ergebnis erhalten Sie, wenn Sie an eine beliebige andere Stelle der GUI klicken, die außerhalb des Kontextmenüs liegt.
Show TestRuns	
Reset finished runs	
Copy Final Message	
Remove	

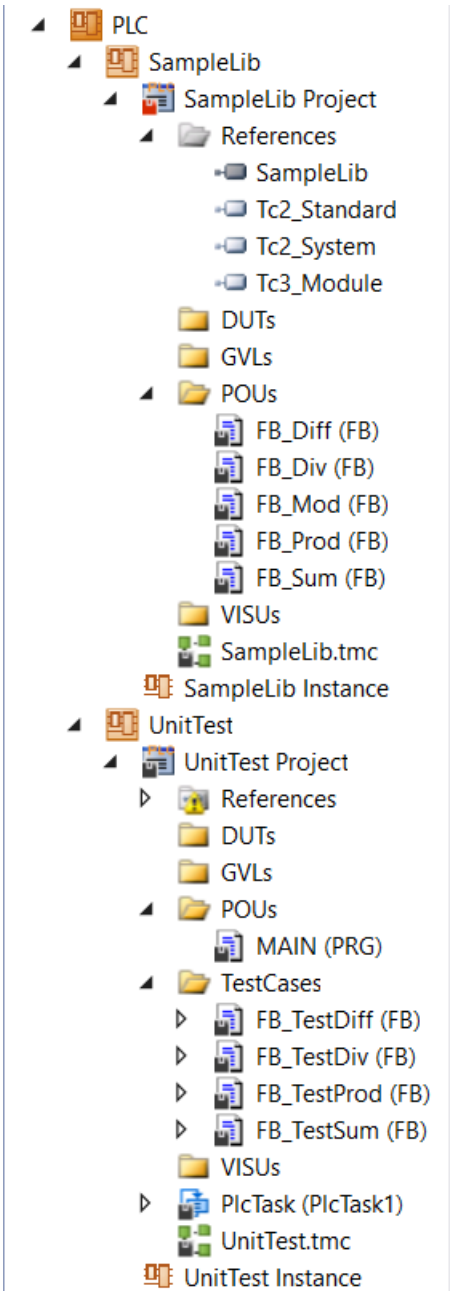
Damit in diesem Fenster Testfälle angezeigt werden, muss die Option **Autostart Boot Projekt** aktiviert sein.

9 Beispiele

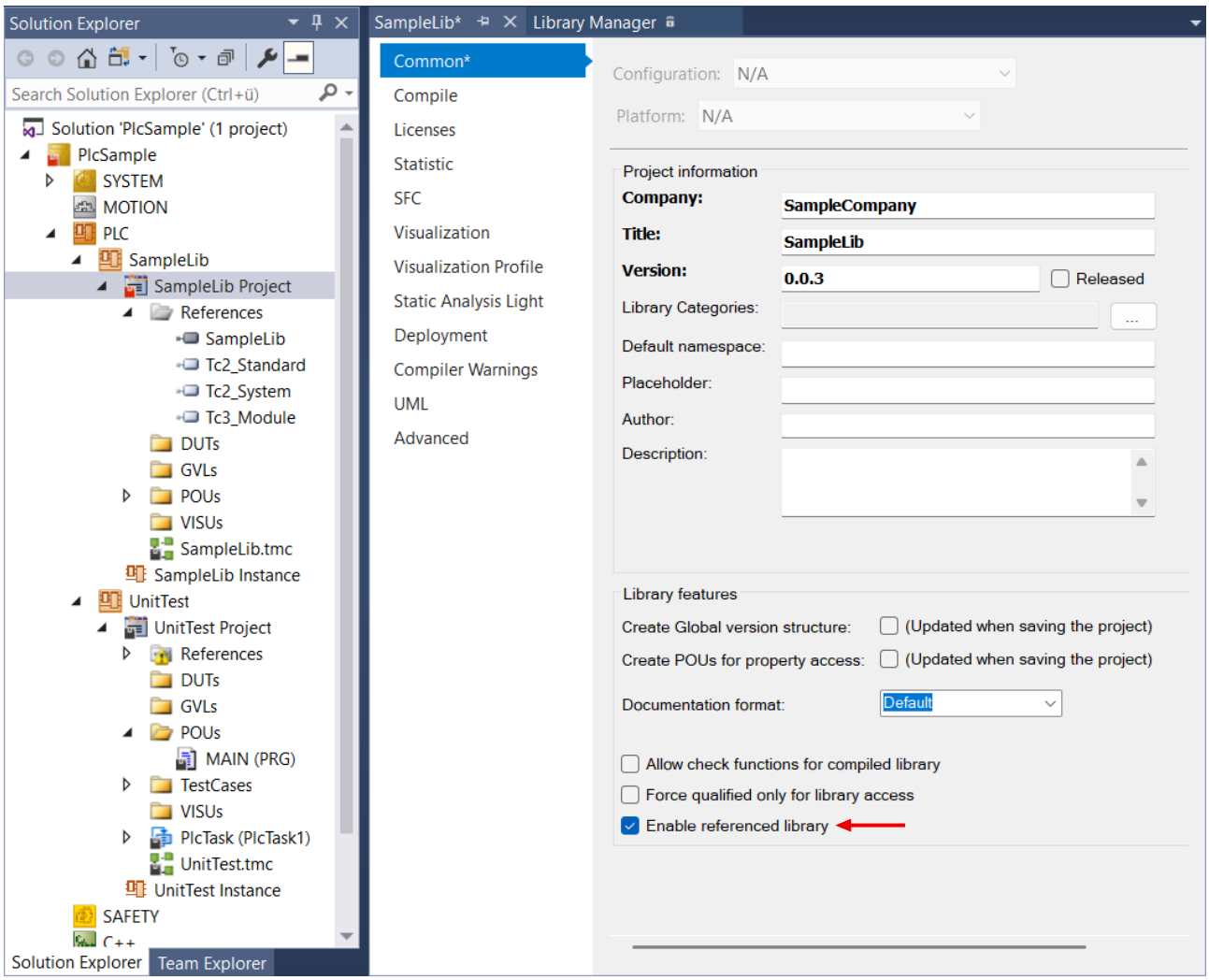
Ein Beispielprojekt für das TF1140 steht Ihnen im Beckhoff-Github unter folgendem Link zur Verfügung:
TF1140-Sample

Das Beispiel zeigt eine einfache Bibliothek, welche Funktionsbausteine für die Grundrechenarten zur Verfügung stellt. In dem Funktionsbaustein für die Addition (FB_Sum) ist zu diesem Zweck mit Absicht ein Fehler eingebracht (anstatt addiert wird subtrahiert).

Aufbau des Beispiels:

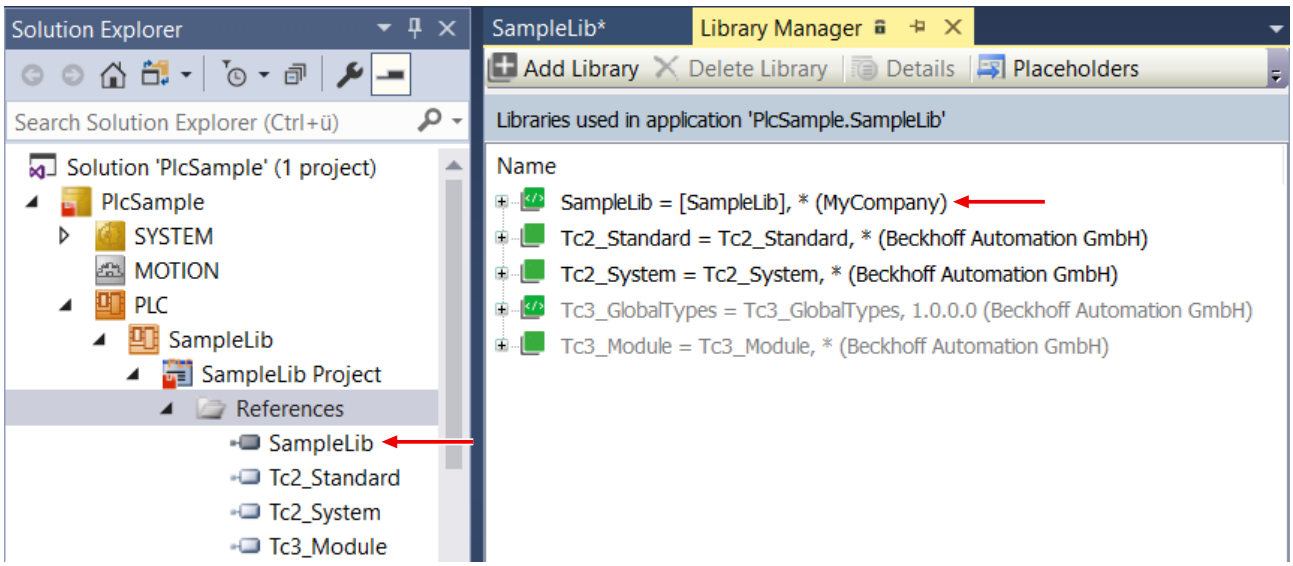


Das Beispiel enthält ein TwinCAT-Projekt. In diesem sind zwei SPS-Projekte hinterlegt. Ersteres ist die eigentliche Bibliothek (SampleLib), die entwickelt wird und deshalb in diesem Beispiel getestet werden soll. Diese ist mit der Eigenschaft „Referenzierung des Projektes als Bibliothek erlauben“ markiert, welche es seit der TwinCAT Version 4026 gibt.



Durch die Option ist es möglich, diese Bibliothek im selben TwinCAT-Projekt zu verwenden, ohne sie vorher auf dem System zu installieren. Dies vereinfacht das Testen vor allem in CI/CD Pipelines, da dieser zusätzliche Arbeitsschritt entfällt.
Um diesen Vorteil auszunutzen, ist es sinnvoll das Testprojekt für diese Bibliothek im selben TwinCAT-Projekt zu verwalten, so wie es im Beispiel gezeigt wird. Damit ist gegeben, dass der Quellcode für die Bibliothek und für die Tests im selben Source Code Repository verwaltet werden und somit auf dem aktuellen Stand sind.

Im Testprojekt selbst muss die zu testende Bibliothek entsprechend eingebunden werden. Referenzierte Bibliotheken werden im Bibliotheksverwalter mit eckigen Klammern und im Projekt selbst mit einem etwas dunkleren Icon dargestellt.



Zusätzlich zur Bibliothek, die getestet werden soll, muss die Bibliothek Tc3_PlcTestFramework eingebunden werden. Diese wird bei der Installation des Testframeworks mit ausgeliefert und enthält die benötigten Basis-Klassen/Funktionsbausteine, sowie die Schnittstelle zu den Testclients (siehe Kapitel Technische Einführung [▶ 12]).

Unter dem Ordner TestCases im UnitTest-SPS-Projekt finden Sie die einzelnen Testfunktionsbausteine, für die jeweiligen Bausteine in der Bibliothek. Diese müssen im Main-Programm instanziiert werden. Im Rumpf der Main-POU muss der Test-Controller-Baustein, der die Schnittstelle zu den Test-Clients bereitstellt, aufgerufen werden. Zum Ausführen der Tests gehen Sie vor wie im Kapitel Quickstart [▶ 13] beschrieben.

10 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Mehr Informationen:
www.beckhoff.com/tf1140

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com