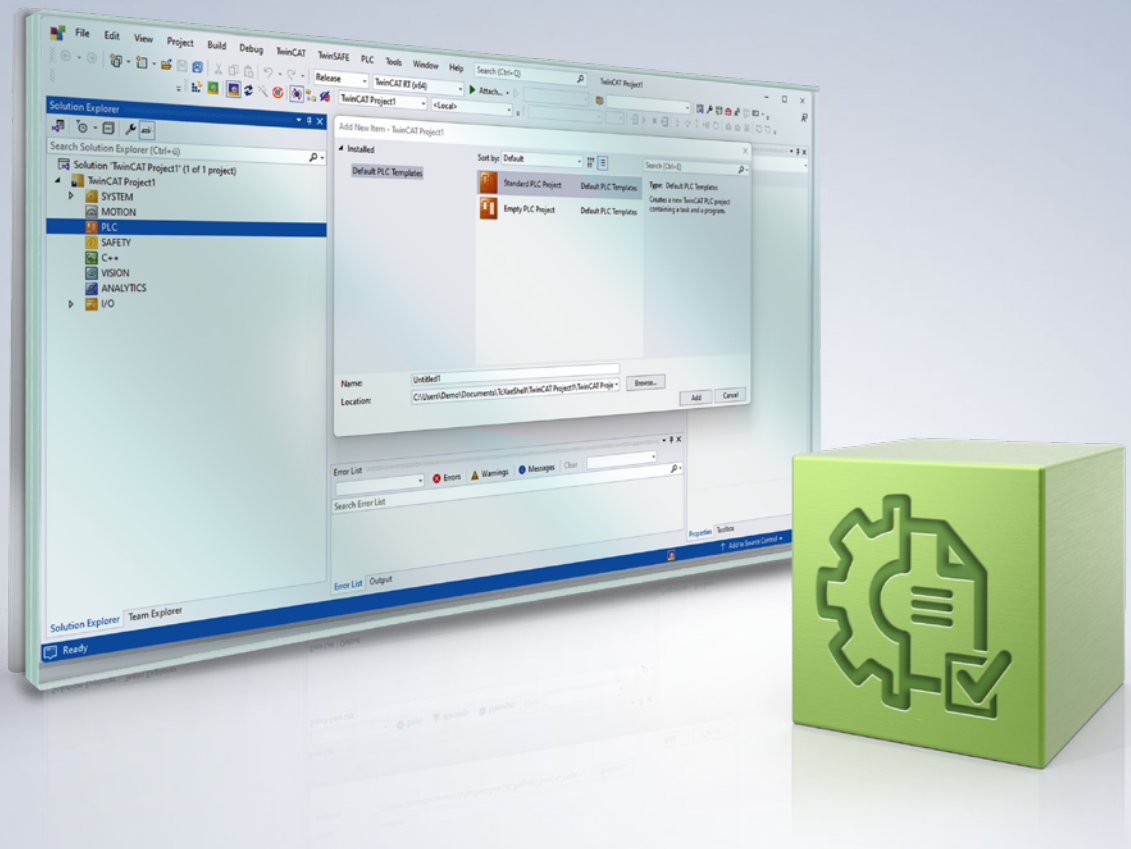
BECKHOFF New Automation Technology

Manual | EN

TF1140

TwinCAT 3 | TestFramework



2026-08-31 | Version: 0.2.0

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
The documentation and the following notes and explanations must be complied with when installing and commissioning the components.
The trained specialists must always use the current valid documentation.

The trained specialists must ensure that the application and use of the products described is in line with all safety requirements, including all relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been compiled with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without notice.
Claims to modify products that have already been supplied may not be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

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.
If third parties make use of the designations or trademarks contained in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

Copyright

Third-party trademarks

Trademarks of third parties may be used in this documentation. You can find the trademark notices here: https://www.beckhoff.com/trademarks.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

Test-Driven Development (TDD) is a method for developing control code in which tests are written to verify the functions of the control code before the implementation of those functions takes place. Therefore, the chosen implementation does not affect the creation of test cases. After the control code has been implemented, the test cases are run repeatedly until they no longer fail, thereby confirming that the system functions as intended. The goal of this approach is to improve code quality.

TDD is often used in conjunction with other agile software development methods, such as continuous integration, to improve not only code quality but also the speed at which code can be changed, and to reduce the time it takes to deliver new functions to end customers.

3 Installation

System Requirements

Technical data	Description
Operating system	Windows 10
Target platform	Windows
Minimum TwinCAT version	TwinCAT 3.1.4026
TwinCAT licenses	TF1140 TwinCAT 3 TestFramework

TwinCAT Package Manager: Installation (TwinCAT 3.1 Build 4026)

Detailed instructions on installing products can be found in the chapter Managing TwinCAT software in the TwinCAT 3.1 Build 4026 installation instructions.

Install the following workload to be able to use the product:

- TF1140 | TwinCAT 3 TestFramework

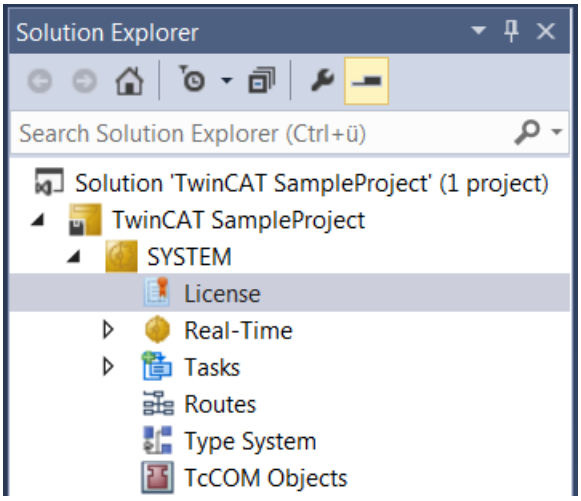
Licensing

The TF1140 TwinCAT 3 TestFramework provides you with a 7-day trial license that allows you to create and run your tests locally on your engineering device. To use the automatic testing function–for example, tests in a pipeline–you need a license for the full version of the product. Below you will find explanations of both types of licensing.

Licensing the 7-day test version of a TwinCAT 3 Function

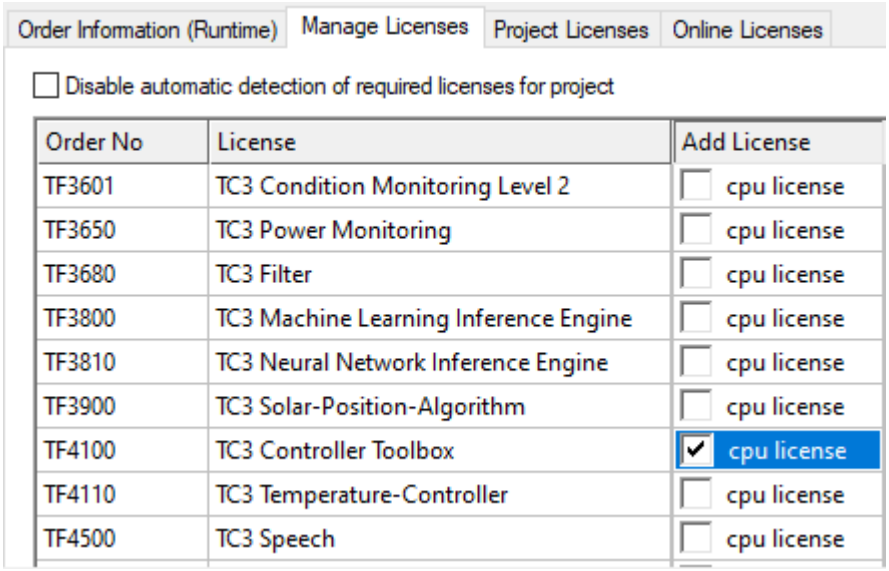
A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.
4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.

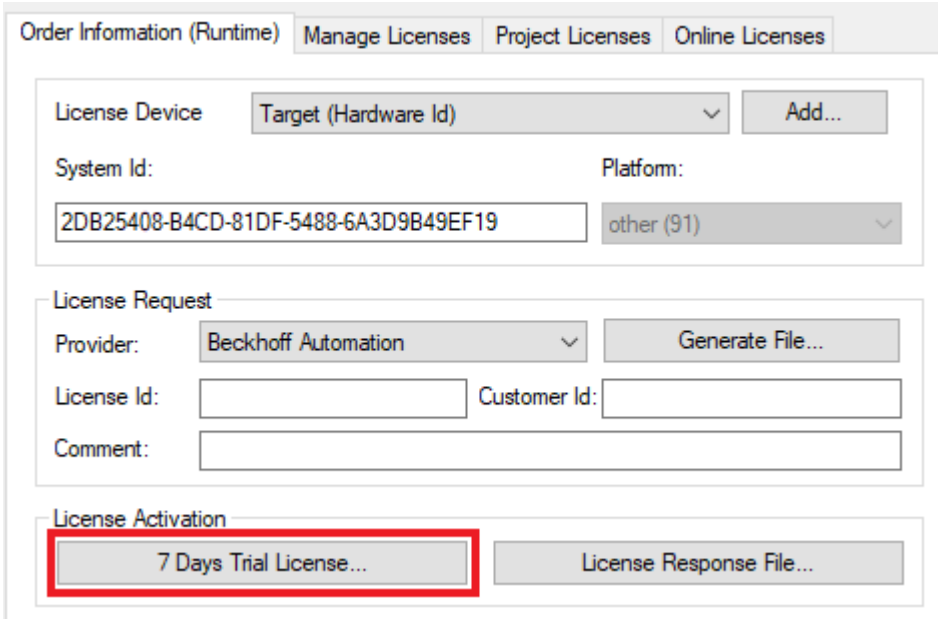


⇨ The TwinCAT 3 license manager opens.

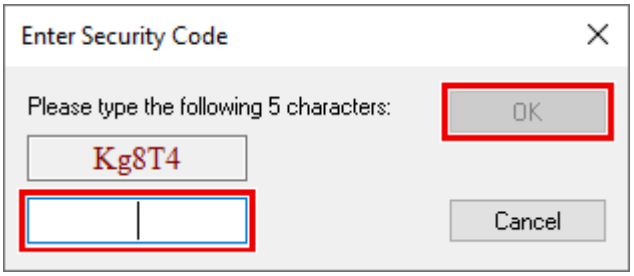
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



6. Open the **Order Information (Runtime)** tab.
   ⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**
7. Click **7-Day Trial License...** to activate the 7-day trial license.



   ⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.



8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.

10. Restart the TwinCAT system.

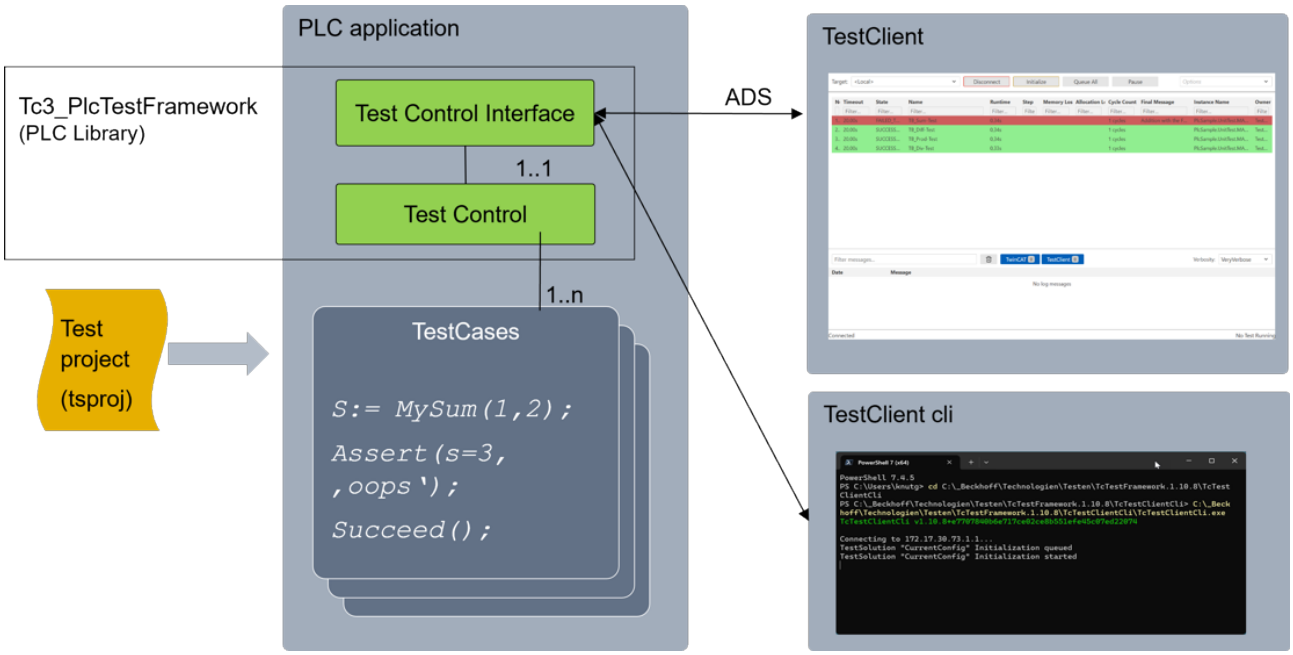
⇨ The 7-day trial version is enabled.

Licensing the full version of a TwinCAT 3 Function

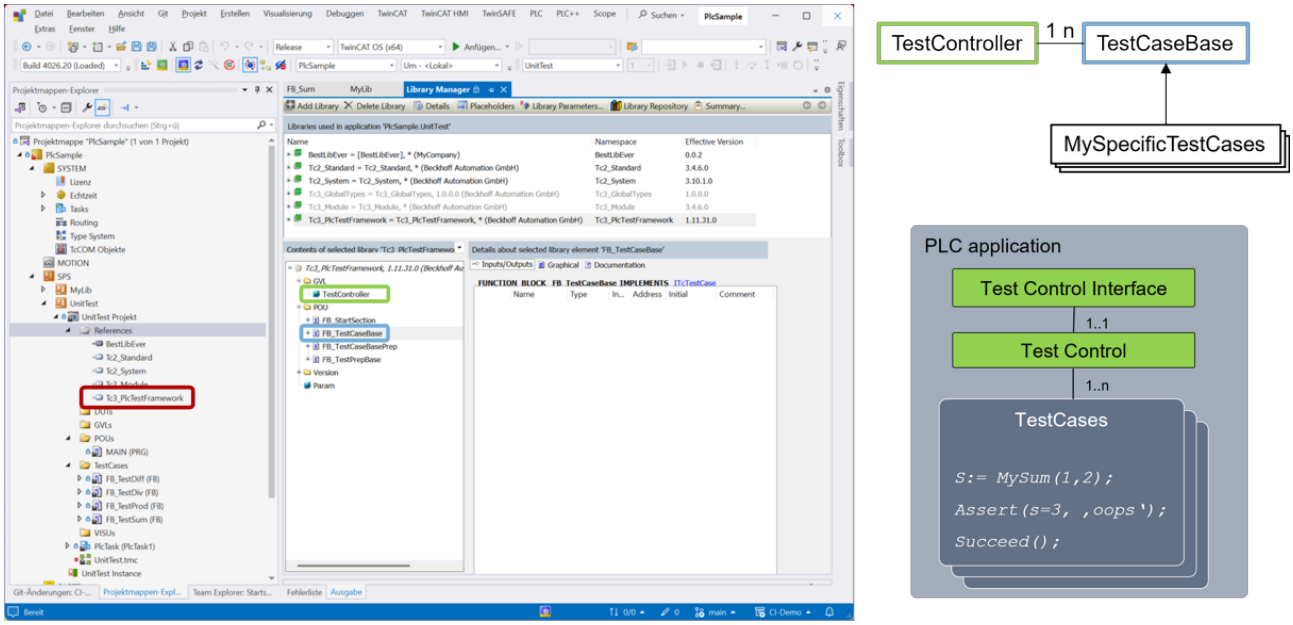
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

4 Technical introduction

The UnitTestFramework consists of a PLC library, which must be implemented into the test project, a test client with a user interface, and a test client without a user interface. The latter is used for command line calls and can be used to automatically run tests as part of a CI/CD pipeline. The following figure shows a schematic representation of the set-up.



If the control code to be tested is located in a PLC library, that library is also implemented in the test project in the version being tested. This can be done automatically, for example, using the Automation Interface. The test project contains the test cases for the various function blocks in the library. Each test case is its own function block. The base class for the test cases already includes functionality that allows the test cases to register with a test controller. It is located in the global variable list of the test framework library (see the following figure).



As soon as a test client establishes a connection to the target system on which a test project has been activated, the test cases are automatically retrieved from the test controller and represented as a list in the test client's GUI. From here, you can now run the test cases either all at once or individually. The test client without a GUI does not display the test cases; instead, it begins executing the test cases immediately after being launched. The test results are saved as a file in JUnit or HTML format. They can also be exported in these two formats from the test client with a GUI.

5 Quickstart

The following chapter is intended to provide an easy introduction to using the TwinCAT 3 TestFramework. In principle, the TestFramework can be used in two different ways.

- The test cases are located in a TwinCAT PLC project, and the control code to be tested is implemented as a PLC library. This use case is intended for testing PLC libraries.
- The test cases and the control code to be tested are located in the same PLC project; a compiler switch is used, for example, to determine whether the test cases are also compiled (see the chapter on conditional compilation/conditional pragmas).

This quick start guide covers the first use case, as this is likely the more common scenario.

1. Create a new TwinCAT 3 project.
2. Create a new PLC project.
3. Add the Tc3_PlcTestFramework PLC library to the PLC project.
4. Add the PLC library to be tested, unless the test cases are included directly in the application's control code.

Implementation of the test cases:

Follow these steps for each test case you want to implement:

5. Create a new function block that is derived from the FB_TestCaseBase block in the Tc3_PlcTestFramework PLC library.
6. Enter the name of the test case as an attribute above the name of the function block, exactly as it should appear in the test log or the test clients. If you do not want to specify a name, the name of the POU will automatically be used as the name of the test case.
7. Optionally, use attributes to specify a timeout and the test case owner.
   ⇨ The declaration part of the function block could then look like this:
   ```
   { attribute 'Name':='TB_Sum-Test' }
   { attribute 'Timeout':='t#20s' }
   { attribute 'Owner':='TestOwner' }
   FUNCTION_BLOCK FB_TestSum EXTENDS FB_TestCaseBase
   VAR_INPUT
   END_VAR
   …
   ```
8. In the variable declaration of the function block, declare the function block you want to test
9. Also declare variables that will hold the expected result of the test case and the actual result. This explicit definition is necessary because the method that compares the result with the expected value is based on the `Any` data type.
10. Implement the test case. The result is verified for correctness using the method `AssertEqual`, which is already implemented in the base function block.

The tests run until the test case is exited using the `Succeeded()` function or an error occurs. This makes it possible to design test cases so that they can be run iteratively.

The control code for testing a summation block could look something like this, in simplified form:

```
{ 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);
```

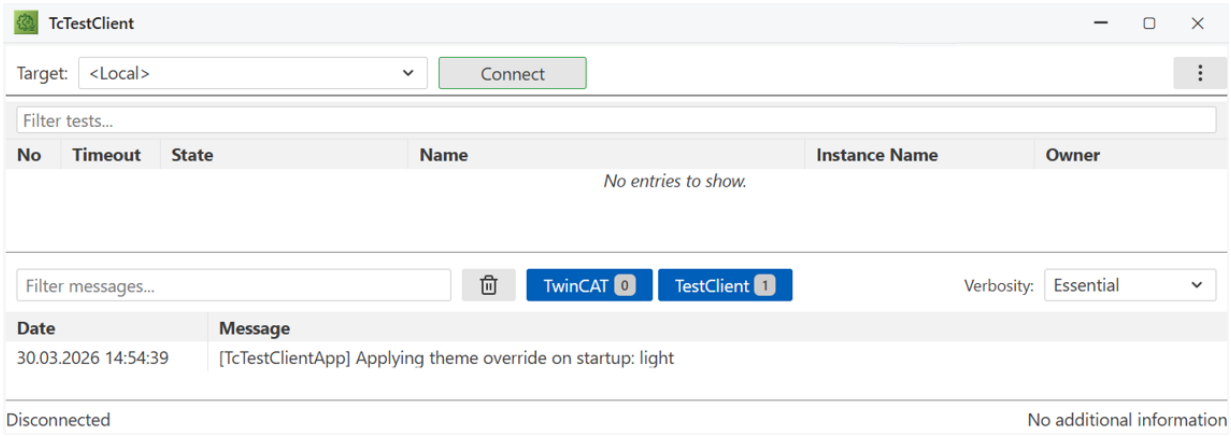
```
expectedValue:=4;
AssertEqual(expected:= expectedValue, actual:= result, message:= 'Addition with the FB_Sum function
block is incorrect!');
Succeeded();
```

Once all test cases have been implemented, proceed as follows:

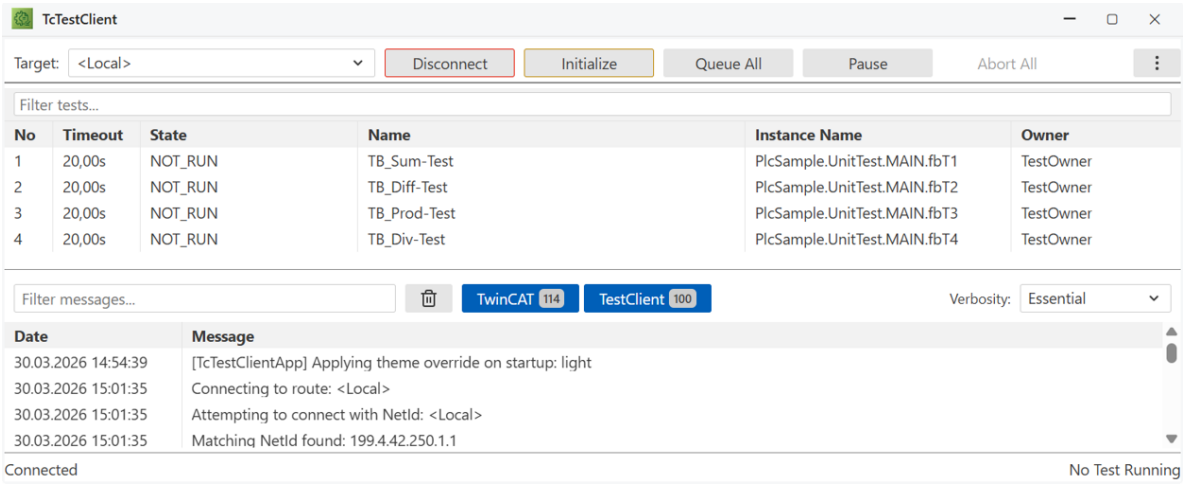
11. Make sure that the **Autostart Boot Project** option is active in the PLC project.
12. Activate the project on the desired target system.
13. Open or launch a test client and run the tests.

Test client with user interface:

14. Start the Test Client.



15. Use the **target** choice drop-down box to select the target system on which the test project will run.
16. Then press the Connect button located directly next to the target system selection
    ⇨ All test cases are represented in a tabular view in the test client.



17. Click the **Queue All** button to start running all test cases.

⇨ For a detailed description of the test client's user interface functions, see the chapter Reference User Interface [▶ 32].

Command line client:

1. Run the following command using a script or from the command line:

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

By default, TestClientCli is installed in the *directory C:\Program Files (x86)\Beckhoff\TwinCAT\Functions\TE1040-TestFramework\TcTestClientCli*.

- The `--target` option refers to the target system on which the test cases are executed.
- The `--saveResultAsJunit` option enables the results to be written in JUnit format and specifies the file where the results will be saved.
- The `--TcStartupRetries` option specifies the number of attempts–that is, how often the test client tries to start a test.
- In this case, the option `--mode checkMomory` enables the test client to check the TwinCAT router memory after each test run. This makes it possible to find memory leaks within the function blocks you've written. Using this option results in a slightly longer runtime.

6 Command Line Invocation

The command line client is ideal for automation of test execution, for example, in a CI/CD workflow. It can be called in a script as follows:

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

The test client establishes a connection to the target system, which you specify using the `--target` option. In addition, the call shown above specifies that the results should be output in JUnit format and saved at the path indicated above.

You can view a detailed list of all command line parameters and their meanings using the command

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

inquire about.

7 PLC API

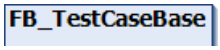
This chapter describes the PLC API of the TwinCAT 3 TestFramework. There are two PLC libraries available for this purpose that can be used in conjunction with the TF1140. The Tc3_PlcTestFramework PLC library and the Tc3_PlcTestUtilitiesGeneric. You are required to include the first one. Among other things, it contains the base class for the test cases (FB_TestCaseBase).

The Tc3_PlcTestUtilitiesGeneric PLC library contains additional function blocks with convenience functions that would otherwise have to be programmed into the PLC application (access to the parameters of a TwinCAT 3 runtime module, a list of ADS error codes, etc.).

7.1 Tc3_PlcTestFramework

7.1.1 Function Blocks

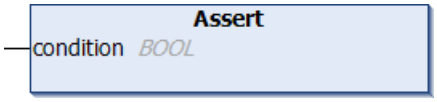
7.1.1.1 FB_TestCaseBase



FB_TestCaseBase is the base function block for new test cases. This function block already includes a wide range of methods related to test case management. This applies, for example, to the method `FB_init`, in which the function block automatically registers with the test controller. You do not need to–and should not–override this method when implementing a new test case. In addition, methods are available for, among other things, comparing the results or advancing the test state machine. These are described below. The base test case also has the following internal variables:

Name	Type	Default value	Meaning
Variables passed via attributes or the FB_init method.			
sName	STRING(255)	-	Test Case Name
Timeout	TIME	0 s	Test case timeout
sOwner	STRING(255)	-	Test Case Owner
Internal variables that can be modified via methods.			
nCurStep	UDINT	0	Status variable for a possible step sequence
bDone	BOOL	FALSE	Indicates whether the function block has completely finished executing.
bError	BOOL	TRUE	Error Analysis of the function block call
sFinalMessage	STRING(255)		Message that is passed after the test.
sInstName	STRING(255)		Instance name of the function block (determined automatically)

7.1.1.1.1 Assert



The `Assert` method can be used to indicate that the current test case has failed. Unlike the AssertEqual method [▶ 18], which compares the result to the expected value on its own, the comparison here must be performed in the test case's application code, and the result of the comparison must be passed to the method as a Boolean value. If you also want to pass an error message to the calling test client, use the AssertMSG method [▶ 19] .

Syntax

```
METHOD Assert
VAR_INPUT
    condition : BOOL
END_VAR
```

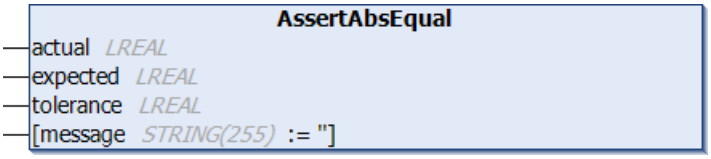
Inputs

Name	Type	Description
condition	BOOL	A BOOL value that indicates whether the test was successful.

Example

```
Assert(condition);
```

7.1.1.1.2 AssertAbsEqual



Similar to the `AssertEqual` method, this method is also used to verify whether the expected value of a test case matches the actual result.

Similarly, a message is displayed here as well in the event of an error. However, the `AssertAbsEqual` function checks whether the expected value and the result are close enough to each other in absolute value, that is, whether they fall within a tolerance that can also be specified.

Syntax

```
METHOD PROTECTED AssertAbsEqual
VAR_INPUT
    actual          : LREAL;
    expected      : LREAL;
    tolerance      : LREAL;
    message         : STRING(255) := ''; // optional assert message
END_VAR
```

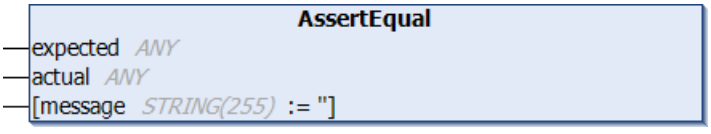
Inputs

Name	Type	Description
actual	LREAL	Value determined by the test performed
expected	LREAL	Expected value for the test case.
tolerance	LREAL	Permissible tolerance for this test case between the expected value and the calculated value.
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.1.3 AssertEqual



The `AssertEqual` method can be used to check the result of a test case and, if an error occurs, to display a custom error message. The comparison of the expected value with the result of the test case takes place within the `AssertEqual` method itself. The `AssertEqual` method is based on the ANY data type. Since the parameters must be passed as typed values for this reason, the expected value must first be assigned to a corresponding typed variable and cannot be assigned directly within the call to the `AssertEqual` method.

Syntax

```
METHOD PUBLIC AssertEqual
VAR_INPUT
    expected : ANY;
    actual : ANY;
    message : STRING(255) := ''; // optional assert message
END_VAR
```

Inputs

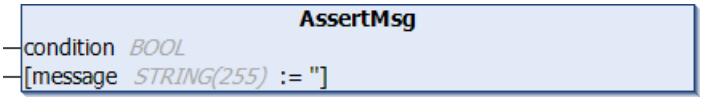
Name	Type	Description
expected	ANY	Expected value for the test case
actual	ANY	Value determined by executing the test case
message	STRING(255)	Error message if the test fails.

Example

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

`Result` is the “calculated” result of the test case.

7.1.1.1.4 AssertMsg



The `AssertMsg` method can be used to check the result of a test case and, if an error occurs, to display a custom error message. It thus extends the functionality of the `Assert` method to include an error message.

Syntax

```
METHOD AssertMsg
VAR_INPUT
    condition : BOOL
    message : STRING(255) := '';
END_VAR
```

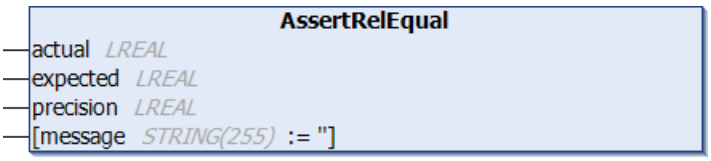
Inputs

Name	Type	Description
condition	BOOL	A BOOL value that indicates whether the test was successful.
message	STRING(255)	Error message if the test fails.

Example:

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

7.1.1.1.5 AssertRelEqual



The method `AssertRelEqual` uses the method `IsRelEqual` to check whether two floating-point values (the test result and the expected value) fall within a defined relative or absolute tolerance. If this is not the case, the test case fails and returns the specified error message.

Syntax

```
METHOD PROTECTED AssertRelEqual
VAR_INPUT
    actual          : LREAL;
    expected      : LREAL;
    precision      : LREAL;
    message         : STRING(255) := ''; // optional assert message
END_VAR
```

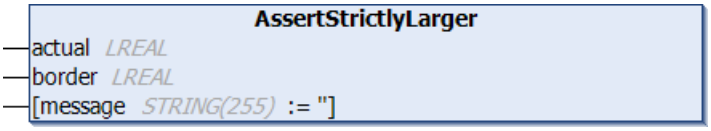
Inputs

Name	Type	Description
actual	LREAL	Value determined by executing the test case.
expected	LREAL	Expected value for the test case
precision	LREAL	Relative tolerance for comparison
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.1.6 AssertStrictLarger



The `AssertStrictlyLarger` method tests whether a value (the result of a test) is strictly greater than a defined threshold.

Syntax

```
METHOD PROTECTED AssertStrictlyLarger
VAR_INPUT
    actual          : LREAL;
    border          : LREAL;
    message         : STRING(255) := ''; // optional assert message
END_VAR
```

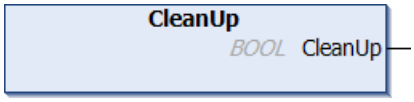
Inputs

Name	Type	Description
actual	LREAL	Value determined by executing the test case
border	LREAL	The threshold against which the calculated value is compared.
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.1.7 Cleanup



`Cleanup` is a method that can be used to clean up the data for a test case. You must override and implement this method when it is needed.

Syntax

```
METHOD CleanUp : BOOL
```

Return value

Name	Type	Description
Cleanup	BOOL	A return value that can be used to indicate that cleanup has been performed.

7.1.1.1.8 GetCurrentStep

As described in the chapter FB_TestCaseBase [▶ 17], there are internal variables that can be used during the implementation of test cases. Among other things, the variable `nCurStep` is used as a status variable for a “Case” construct, which can be used to structure test cases in the form of a state machine. The method `GetCurrentStep` returns the current value of the status variable `nCurStep`. If you want to use a different status variable, you can override this method.

Syntax

```
METHOD GetCurrentStep : UDINT
```

Return value

Name	Type	Description
GetCurrentStep	UDINT	Returns the current state of the state machine if it is used in the test case.

Example

```
currentStep:= GetCurrentStep();
```

7.1.1.1.9 GoToStep

As described in the chapter FB_TestCaseBase [▶ 17], there are internal variables that can be used during the implementation of test cases. Among other things, the variable `nCurStep` is used as a status variable for a “Case” construct, which can be used to structure test cases in the form of a state machine. The method

`GoToStep` sets the value of the status variable `nCurStep` to the value passed to it. This means that in the next iteration of the “Case” construct, you will jump to exactly this step. If a different status variable is to be used, this method must be overridden when needed.

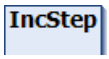
Syntax

```
METHOD GoToStep
VAR_INPUT
    step : UDINT; // target step
END_VAR
```

Inputs

Name	Type	Description
Step	UDINT	Sets the current state of the state machine to this value.

7.1.1.1.10 IncStep



As described in the chapter FB_TestCaseBase [▶ 17], there are internal variables that can be used during the implementation of test cases. Among other things, the variable `nCurStep` is used as a status variable for a “Case” construct, which can be used to structure test cases in the form of a state machine. The IncStep method increments the value of the variable `nCurStep` by 1. If a different status variable is to be used, this method must be overridden when it is to be used.

Syntax

```
METHOD IncStep
VAR_INPUT
END_VAR
```

7.1.1.1.11 IncStepIf

Like the IncStep method [▶ 22], this method increments the internal status variable of the “Case” construct by 1 when a condition is met. The condition itself is checked in the application code, and you must pass the Boolean result when calling the method. If a status variable other than `nCurStep` is to be used, this method must be overridden if it is to be applied.

Syntax

```
METHOD IncStepIf
VAR_INPUT
    condition : BOOL;
END_VAR
```

Inputs

Name	Type	Description
condition	BOOL	If this value is TRUE, the current state of the state machine is incremented.

Example

```
IncStepIf(Condition);
```

7.1.1.1.12 IsRelEqual

The `IsRelEqual` method checks whether two floating-point numbers of type “LREAL” fall within a specified tolerance. If both values are greater than the value specified in `absCompScale`, the relative deviation between the two values is checked against the tolerance. If any of the values fall below the tolerance, an absolute comparison is performed. Among other things, this method is used in the AssertRelEqual method [▶ 20].

Syntax

```
METHOD IsRelEqual : bool
VAR_INPUT
    actual          : LREAL;
    expected      : LREAL;
    precision      : LREAL;
END_VAR
VAR CONSTANT
    absCompScale : LREAL := 1E-12;
END_VAR
```

Inputs

Name	Type	Description
actual	LREAL	Value determined by executing the test case
expected	LREAL	Expected value for the test case
precision	LREAL	Relative tolerance for comparison

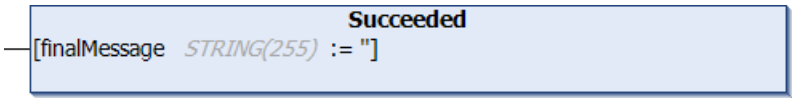
Return value

Name	Type	Description
IsRelEqual	BOOL	Returns the Boolean result of the comparison.

Example

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

7.1.1.1.13 Succeeded



The `succeeded` method is called to confirm that the test case was executed successfully. The test case runs until either a `Assert` method detects that the test case has failed, the configured timeout stops the test case from running, or calling `succeeded` confirms that the test case has completed error-free.

Syntax

```
METHOD Succeeded
VAR_INPUT
    finalMessage : STRING(255) := ''; // optional message
END_VAR
```

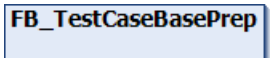
Inputs

Name	Type	Description	Default setting
finalMessage	STRING(255)	Final message when the test case has completed.	“”

Example

```
succeeded();
```

7.1.1.2 FB_TestCaseBasePrep



FB_TestCaseBasePrep is an extension of FB_TestCaseBase [▶ 17]. When instantiating this function block, you can specify a function block of the type FB_TestPrepBase [▶ 24]. This makes it possible to move the steps into a separate function block in order to reach the operating point of a test case. It offers virtually the same methods for detecting errors to determine whether errors occur even before a test case reaches its operating point.

Example of how to use or assign a preparation step to a test case:

Declaration of the test case using the extended test case class FB_TestCaseBasePrep:

```
FUNCTION_BLOCK FB_TestCase1 EXTENDS FB_TestCaseBasePre
…
```

Declaration of the preparation step:

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

Instantiation in the declaration part of the test program's main POU:

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

7.1.1.2.1 FB_Init

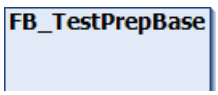
The FB_Init method is a kind of constructor for a function block. Any additional variables specified there must be set when declaring a function block instance of this type.
In the case of the function block ` FB_TestCaseBasePrep [▶ 24] `, this is an interface pointer of type `ITcTestPreparation`. Since every function block for test preparation automatically has this interface through its derivation from the function block FB_TestPrepBase [▶ 24], the required preparation step can be specified when instantiating a FB_TestCaseBasePrep [▶ 24] function block.

Instantiation in the declaration part of the test program's main POU:

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

Take great caution when modifying this method so as not to negatively affect the functionality of the TestFramework.

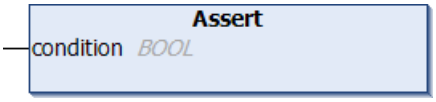
7.1.1.3 FB_TestPrepBase



Function block for the preparation steps of a test case. This function block can be used to move the steps required to reach the operating point of a test case into a separate function block. The advantage of this is that these steps can be reused in different test cases. The preparation function block also has the necessary methods to detect errors that occur during the execution of the preparation steps. The basic preparation step has the following internal variables, which can be modified using the methods described below.

Name	Type	Default value	Meaning
nCurStep	UDINT	0	Status variable for a possible step sequence
bDone	BOOL	FALSE	Indicates whether the function block has completely finished executing.
bError	BOOL	TRUE	Error Analysis of the function block call
sFinalMessage	STRING(255)		Message that is passed after the test.

7.1.1.3.1 Assert



The `Assert` method can be used to indicate that one of the preparation steps for the following test case has failed. Unlike the AssertEqual method [▶ 26], which compares the result to the expected value on its own, the comparison here must be performed in the application code of the preparation step, and the result of the comparison must be passed to the method as a Boolean value. If an error message is also to be passed to the calling test client, the AssertMSG method [▶ 27] should be used.

Syntax

```
METHOD Assert
VAR_INPUT
    condition : BOOL
END_VAR
```

Inputs

Name	Type	Description
condition	BOOL	A BOOL value that indicates whether the test was successful.

Example

```
Assert(condition);
```

7.1.1.3.2 AssertAbsEqual

Similar to the `AssertEqual` method, this method is also used to verify whether the expected value of a test preparation step matches the actual result.

Similarly, a message is displayed here as well in the event of an error. However, the “AssertAbsEqual” method checks whether the expected value and the result are close enough to each other in terms of magnitude (absolute value). The amount must fall within the tolerance range, which you can also specify.

Syntax

```
METHOD PROTECTED AssertAbsEqual
VAR_INPUT
    actual          : LREAL;
    expected      : LREAL;
    tolerance      : LREAL;
    message         : STRING(255) := ''; // optional assert message
END_VAR
```

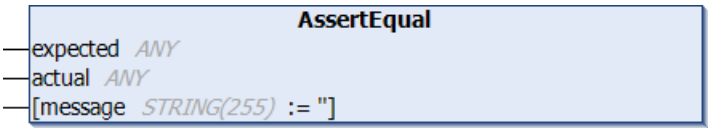
Inputs

Name	Type	Description
actual	LREAL	Value determined by the test performed
expected	LREAL	Expected value for the test case.
tolerance	LREAL	Permissible tolerance for this test case between the expected value and the calculated value.
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.3.3 AssertEqual



The `AssertEqual` method can be used to verify the result of a test preparation step and, if an error occurs, to display a custom error message. The comparison of the expected value with the result of the preparation step takes place within the AssertEqual method itself. The AssertEqual method is based on the ANY data type. Since the parameters must be passed as typed values for this reason, the expected value must first be assigned to a corresponding typed variable and cannot be assigned directly within the call to the AssertEqual method.

Syntax

```
METHOD PUBLIC AssertEqual
VAR_INPUT
    expected : ANY;
    actual : ANY;
    message : STRING(255) := ''; // optional assert message
END_VAR
```

Inputs

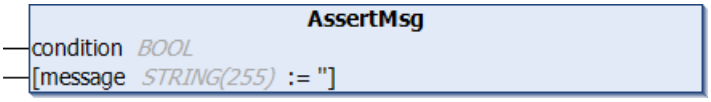
Name	Type	Description
expected	ANY	Expected value for the test case
actual	ANY	Value determined by executing the test case
message	STRING(255)	Error message if the test fails.

Example

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

`Result` is the “calculated” result of the test preparation step.

7.1.1.3.4 AssertMsg



The `AssertMsg` method can be used to verify the result of a test preparation step and, if an error occurs, to display a custom error message. It therefore extends the functionality of the “Assert” method to include an error message.

Syntax

```
METHOD AssertMsg
VAR_INPUT
    condition : BOOL
    message : STRING(255) := '';
END_VAR
```

Inputs

Name	Type	Description
condition	BOOL	A BOOL value that indicates whether the test was successful.
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.3.5 AssertRelEqual

The `AssertRelEqual` method uses the *IsRelEqual* method to check whether two floating-point values (the result of the preparation step and the expected value) fall within a defined relative or absolute tolerance. If this is not the case, the test case fails and returns the specified error message.

Syntax

```
METHOD PROTECTED AssertRelEqual
VAR_INPUT
    actual          : LREAL;
    expected      : LREAL;
    precision      : LREAL;
    message         : STRING(255) := ''; // optional assert message
END_VAR
```

Inputs

Name	Type	Description
actual	LREAL	Value determined by executing the test case.
expected	LREAL	Expected value for the test case
precision	LREAL	Relative tolerance for comparison
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.3.6 AssertStrictLarger

The `AssertStrictlyLarger` method tests whether a value (the result of a test) is strictly greater than a defined threshold.

Syntax

```
METHOD PROTECTED AssertStrictlyLarger
VAR_INPUT
    actual          : LREAL;
    border          : LREAL;
    message         : STRING(255) := ''; // optional assert message
END_VAR
```

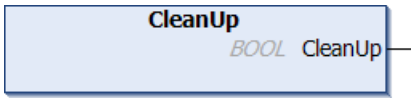
Inputs

Name	Type	Description
actual	LREAL	Value determined by executing the test case
border	LREAL	The threshold against which the calculated value is compared.
message	STRING(255)	Error message if the test fails.

Example

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

7.1.1.3.7 Cleanup



`Cleanup` is a method that can be used to “clean up” the data from a preparation step, provided that this data is not needed for the test case itself. You must override and implement this method when it is needed.

Syntax

```
METHOD CleanUp : BOOL
```

Return value

Name	Type	Description
Cleanup	BOOL	A return value that can be used to indicate that cleanup has been performed.

7.1.1.3.8 GetCurrentStep

As described in the FB_TestPrepBase [▶ 24] chapter, there are internal variables that can be used during the implementation of the preparation steps. Among other things, the variable `nCurStep` is used as a status variable for a “Case” construct, which can be used to structure the state machine so that it reaches the operating point of a test case. The method `GetCurrentStep` returns the current value of the status variable `nCurStep`. If you want to use a different status variable, you can override this method.

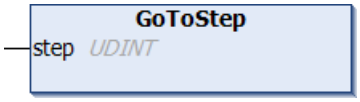
Syntax

```
METHOD GetCurrentStep : UDINT
```

Return value

Name	Type	Description
GetCurrentStep	UDINT	Returns the current state of the state machine if it is used in the test case.

7.1.1.3.9 GoToStep



As described in the chapter FB_TestPrepBase [▶ 24], there are internal variables that can be used during the implementation of the preparation step. Among other things, the variable `nCurStep` is used as a status variable for a “Case” construct, which can be used to structure the state machine so that it reaches the operating point of a test case. The `GoToStep` method sets the value of the status variable `nCurStep` to the value passed to it. This means that in the next iteration of the “Case” construct, the program jumps to exactly this step. If a different status variable is to be used, this method must be overridden when needed.

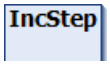
Syntax

```
METHOD GoToStep
VAR_INPUT
    step : UDINT; // target step
END_VAR
```

Inputs

Name	Type	Description
Step	UDINT	Sets the current state of the state machine to this value.

7.1.1.3.10 IncStep



As described in the FB_TestPrepBase [▶ 24] chapter, there are internal variables that can be used during the implementation of the preparation steps. Among other things, the variable `nCurStep` is used as a status variable for a “Case” construct, which can be used to structure the preparatory steps in the form of a state machine. The `IncStep` method increments the value of the `nCurStep` variable by 1. If a different status variable is to be used, this method must be overridden when it is to be used.

Syntax

```
METHOD IncStep
VAR_INPUT
END_VAR
```

7.1.1.3.11 IncStepIf

Like the IncStep method [▶ 29], this method increments the internal status variable of the “Case” construct by 1 when a condition is met. The condition itself is checked in the application code, and you must pass the Boolean result when calling the method. If a status variable other than `nCurStep` is to be used, this method must be overridden if it is to be applied.

Syntax

```
METHOD IncStepIf
VAR_INPUT
    condition : BOOL;
END_VAR
```

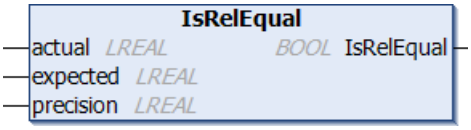
Inputs

Name	Type	Description
condition	BOOL	If this value is TRUE, the current state of the state machine is incremented.

Example

```
IncStepIf(Condition);
```

7.1.1.3.12 IsRelEqual



The `IsRelEqual` method checks whether two floating-point numbers of type “LREAL” fall within a specified tolerance.

If both values are greater than the value specified in `absCompScale`, the relative deviation between the two values is checked against the tolerance. If any of the values fall below the tolerance, an absolute comparison is performed. Among other things, this method is used in the AssertRelEqual method [▶ 27].

Syntax

```
METHOD IsRelEqual : bool
VAR_INPUT
    actual          : LREAL;
    expected      : LREAL;
    precision      : LREAL;
END_VAR
VAR CONSTANT
    absCompScale : LREAL := 1E-12;
END_VAR
```

Inputs

Name	Type	Description
actual	LREAL	Value determined by executing the test case
expected	LREAL	Expected value for the test case
precision	LREAL	Relative tolerance for comparison

Return value

Name	Type	Description
IsRelEqual	BOOL	Returns the Boolean result of the comparison.

Example

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

7.1.1.3.13 SucceededPreparation

SucceededPreparation

The method `SucceededPreparation` is called to confirm that test case preparation has been successfully completed. The preparation steps are executed until either an `Assert` method detects that a preparation step has failed, the configured timeout stops the test case preparation process, or a call to `SucceededPreparation` confirms that the test case preparation has completed error-free and the test case can now be executed.

Syntax

```
METHOD SucceededPreparation
VAR_INPUT
END_VAR
```

Example

```
SucceededPreparation ();
```

7.1.2 GVL

7.1.2.1 TestController

The global variable list TestController contains only the instance of the function block TestCtrl. This function block provides the interface to the TestClients and must be called in the main POU of the test project:

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

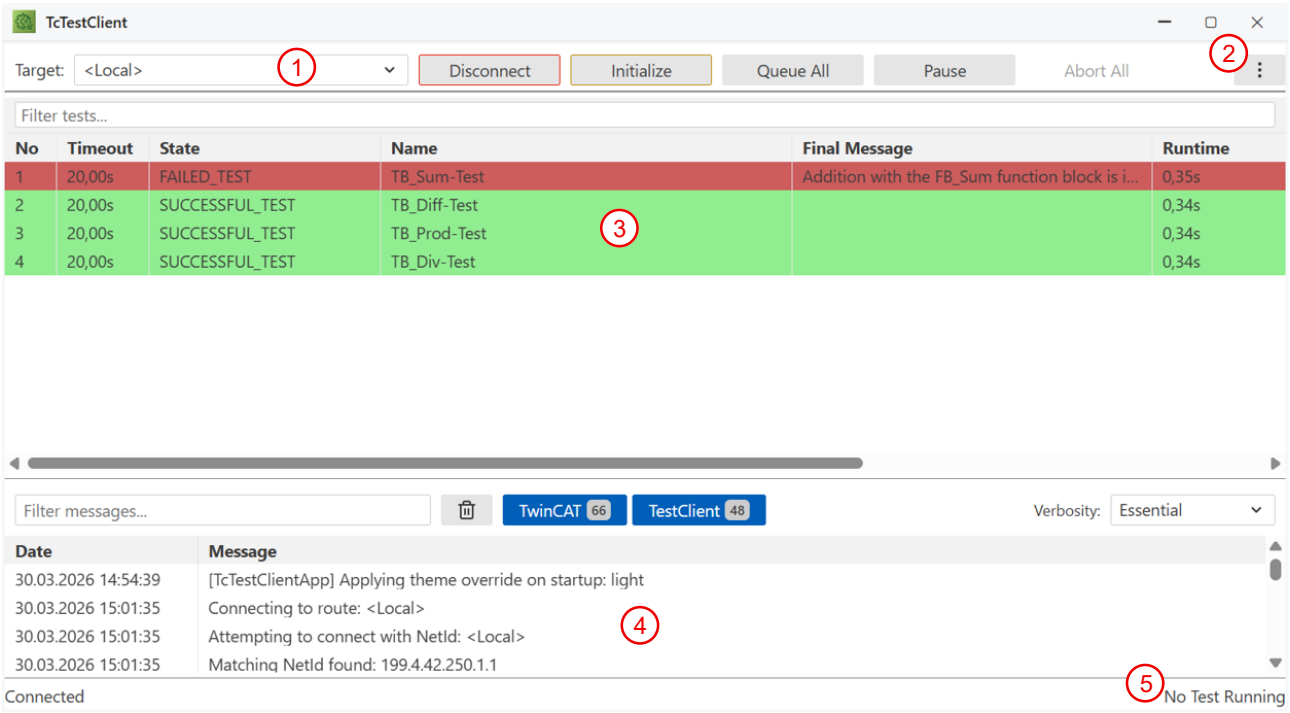
7.1.3 Parameters

You can set the following three parameters for the Tc3_PlcTestFramework library:

MaxTestCaseCount	As described in the Introduction [▶ 12] chapter, all test cases are automatically registered with the test controller. Here, you can adjust the size of the internal list. The parameter is of type UDINT.
DefaultTimeout	You can enter the default time-out period for executing the test cases here. If a test case does not complete within this period, it is marked as failed and the next test case is started.
DefaultOwner	Default owner, which is displayed by default in the graphical client or in the evaluation as JUnit or HTML if it has not been explicitly defined for test cases.

8 Reference User Interface

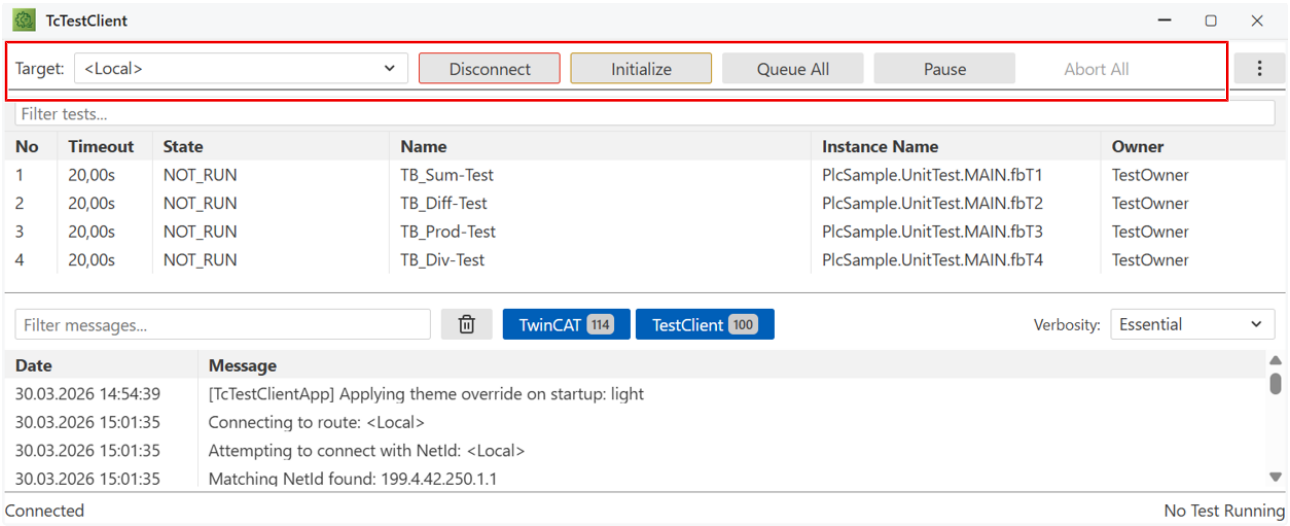
The user interface of the TcTestClient with UI consists of the following components:



1. Toolbar with the most important buttons
2. **Options** menu
3. Test Case window
4. Output window
5. Status display

8.1 Toolbar

You will find all the important buttons in the toolbar.

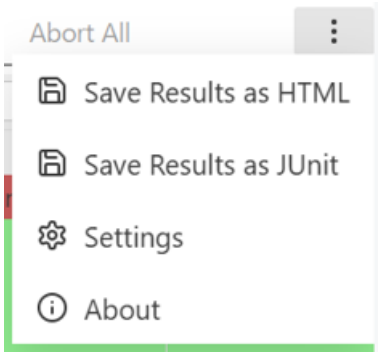


Button	Description
Target	The choice of the target system on which the test project has been activated.
Connect/Disconnect	Establish/terminate a connection to the selected target system.
Initialize	Initialize test cases. This happens automatically after a “Connect”. The manual choice is only necessary if the test cases are loaded from a configuration file. It may contain test cases distributed across various target systems.
Queue all	Start executing all test cases.
Pause	Pause the execution of the test cases.
Abort all	Stop executing the test cases.

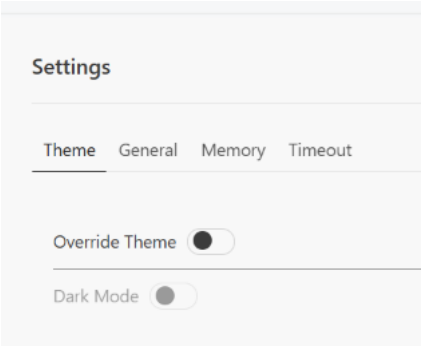
8.2 Settings Menu

The Settings menu contains all the settings that can be configured in the UI Test Client. To open the menu, follow these steps:

1. Right-click the menu icon (…).
2. Select **Settings**.



⇨ The **Settings** menu opens.

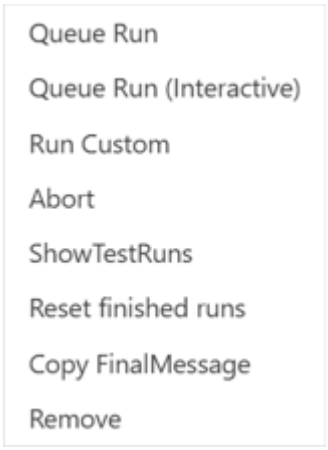


Option	Description
Theme settings	
Override Theme	Allows the system theme settings to be overridden. Enables the **Theme Mode** setting.
Theme Mode	Switches between light and dark themes.
General options	
Fast Mode	Toggles whether the runtime is reset once after each test case to minimize the impact of test cases on one another.
Break On Error	Specifies whether the entire queue is aborted after a test case fails.
Ignore Timeouts	Specifies whether timeouts are taken into account or not.
Stop Iterating on first Failure	Specifies whether the test case should be terminated after the first error occurs during iterative execution.
Requeue Run on Exception	Specifies whether the test execution is restarted if an exception occurs.
Don’t Show Test Runs After Crash	Specifies whether the results are displayed after a crash.
Auto-Detect ADS Port	Automatic detection of the ADS port in use.
PLC ADS Port	Explicit setting of the ADS port used by the test application.
Memory settings	
Check Router Memory	Specifies whether real-time memory is monitored for each test case. Before and after the test case, the availability of real-time memory is queried and compared.
Check Allocation Count	Specifies whether memory allocations should be counted.
Automatic Restart if Memory is Low	Specifies whether the test execution will automatically restart if there is insufficient memory.
Don’t ask to Restart if Memory is Low	Specifies that no requests should be made when storage is low. If the **Automatic Restart if Memory is Low** option is disabled, the system will not prompt you to restart manually when this option is enabled.
Router Memory Limit (bytes)	Defines a limit on the amount of executable real-time memory that must not be exceeded during testing.
Timeout & Build Settings	
TwinCAT Startup Timeout (seconds)	Timeout for the TwinCAT restart
TestControl Timeout (Seconds)	Timeout for the TestController in the PLC
Increase Timeout by (milliseconds)	Extends the timeouts entered for the test cases by this amount. This applies to all test cases.
TwinCAT Startup Retries	Number of attempts to restart TwinCAT

8.3 Test Case Window

After establishing a connection to the target system, all test cases registered with the test controller are displayed in the list.

1. Right-click a test case in the Test Case window to open the test case's context menu.



⇨ The context menu for each test case contains the options shown above.

Option	Description
Queue Run	Starts the test execution of the selected test case.
Queue Run (Interactive)	The selected test case is executed interactively. You will be asked if you want to start the test after the runtime has started. After the test, you will be asked if you want to return to Config mode. This allows you to configure and start a Scope as needed, or to perform other steps that are important before starting a test but could not be programmed within the test itself.
Run Custom	A submenu where you can trigger multiple runs of the same test case.
Abort	Closes the context menu. You will get the same result if you click anywhere else in the GUI outside the context menu.
Show TestRuns	Displays an overview table showing all test runs for a test case. This is typically used when you want to run a test case multiple times in a row to compare memory usage, the allocation counter, and execution times.
Reset completed runs	Deletes the results of all test runs that have been executed.
Copy Final Message	Copies the error message from a test case to the clipboard.
Remove	Deletes this test case from the list of test cases. To add them again, the test cases must be reinitialized.

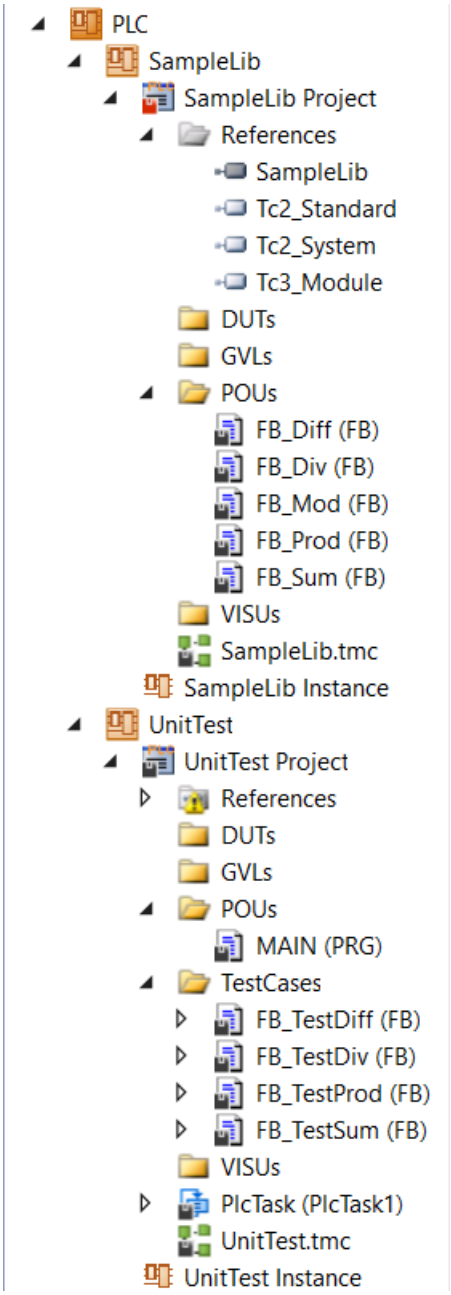
For test cases to be displayed in this window, the **Autostart boot project** option must be enabled.

9 Examples

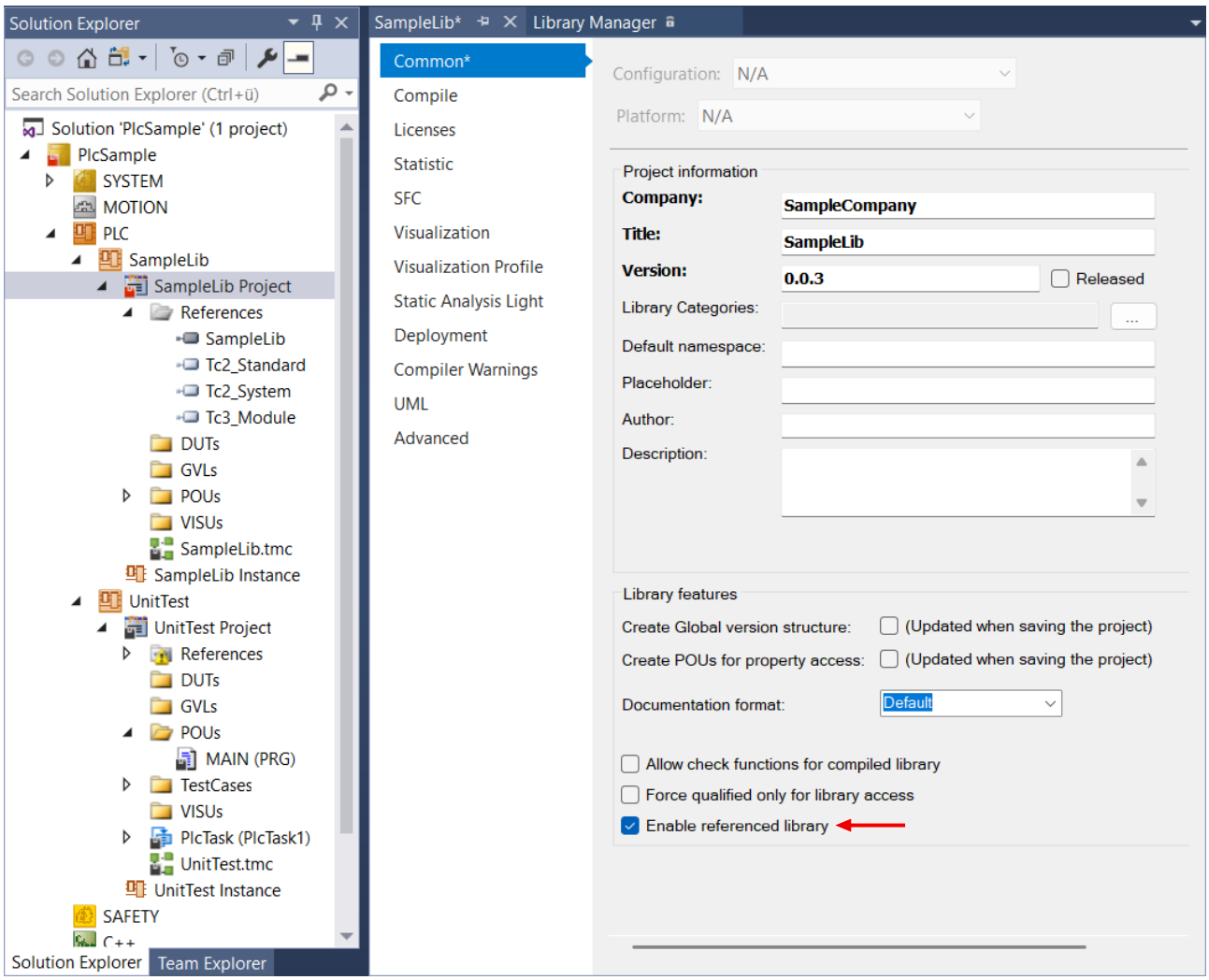
A sample project for the TF1140 is available on the Beckhoff GitHub at the following link: TF1140-Sample

This example shows a simple library that provides function blocks for basic arithmetic operations. In the function block for addition (FB_Sum), an error has been intentionally introduced for this purpose (subtraction occurs instead of addition).

Set-up of the example:

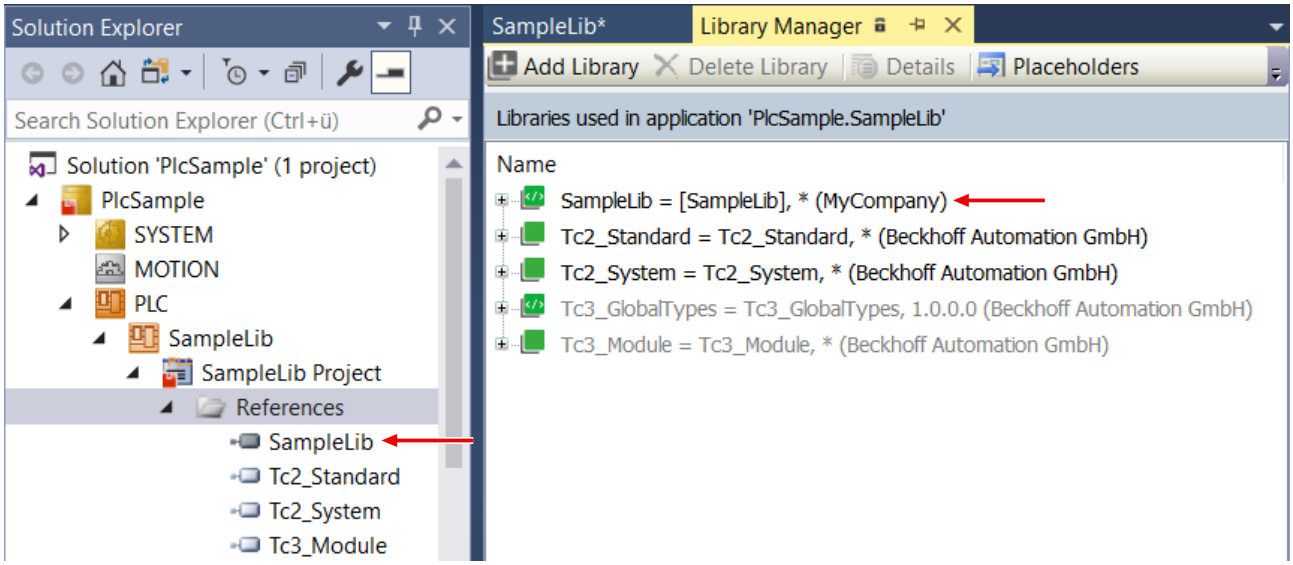


The example contains a TwinCAT project. Two PLC projects are stored in it. The former is the actual library (SampleLib), which is being developed and will therefore be tested in this example. This is marked with the property “Allow the project to be referenced as a library,” which has been available since TwinCAT version 4026.



This option makes it possible to use this library in the same TwinCAT project without first installing it on the system. This simplifies testing, especially in CI/CD pipelines, since it eliminates this additional step.
To take advantage of this, it makes sense to manage the test project for this library within the same TwinCAT project, as shown in the example. This ensures that the source code for the library and for the tests is managed in the same source code repository and is therefore up to date.

In the test project itself, the library to be tested must be implemented accordingly. Referenced libraries are represented in the library management interface with square brackets and in the project itself with a slightly darker icon.



In addition to the library to be tested, the Tc3_PlcTestFramework library must be implemented. This is included with the test framework installation and contains the necessary base classes and function blocks, as well as the interface to the test clients (see the chapter Technical introduction [▶ 12]).

Under the "TestCases" folder in the UnitTest PLC project, you will find the individual test function blocks; the corresponding blocks are located in the PLC library. These must be instantiated in the main program. In the Main-POU body, the test controller function block–which provides the interface to the test clients–must be called. To run the tests, follow the instructions in the chapter Quickstart [▶ 13].

10

10.1 Third-party components

This software contains third-party components.
Please refer to the license file provided in the following folder for further information:
C:\Program Files(x86)\Beckhoff\Legal

11 Support and Service

Beckhoff Support and Service provides assistance with questions regarding Beckhoff products and system solutions, as well as with their planning, commissioning, and operation.

Beckhoff subsidiaries and representative offices

Please contact your Beckhoff subsidiary or your representative office for local support and service for Beckhoff products. You can find the addresses of Beckhoff subsidiaries and representative offices at: www.beckhoff.com/worldwide.

For Beckhoff products and technologies, we offer Training programs, tutorials, and webinars.

Beckhoff Support

Beckhoff Support provides assistance with technical questions regarding the use, planning, programming, and commissioning of Beckhoff products.

☎ Hotline:	+49 5246 963-157
✉ Email:	support@beckhoff.com, Support form
Web page:	www.beckhoff.com/support

Beckhoff Service

Our service specialists are here to assist you with any questions you may have regarding after-sales service processes, such as replacement parts, repairs, etc.

☎ Hotline:	+49 5246 963-460
✉ Email:	service@beckhoff.com, Service form
Web page:	www.beckhoff.com/service

Download finder

The Download finder serves as a central source for downloadable documents and files related to Beckhoff products, such as application reports, technical documentation, drawings, and configuration files.

Beckhoff Information System

The Beckhoff Information System is the central online documentation resource for Beckhoff products and contains documentation on products and automation topics, as well as sample code.

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl, Germany

☎ Phone:	+49 5246 963-0
✉ Email:	info@beckhoff.com
Web page:	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.

More Information:
www.beckhoff.com/tf1140

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