SoMachine

ETEST

User Guide

06/2017

EIO0000002388.00

www.schneider-electric.com

Table of Contents

Safety Information 5

About the Book 9

Chapter 1 Introduction 13

General Information 14

ETEST Objects 15

Methods 17

Macros 21

Chapter 2 Using ETEST 25

Add an ETEST Object 26

Edit Test Cases 27

Use Test Series 28

Use Test Resources 30

Execute Test 32

Record Measured Values 33

Evaluate Test Results 34

Glossary 35

Index 37

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

BEFORE YOU BEGIN

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

⚠ WARNING

UNGUARDED EQUIPMENT

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

START-UP AND TEST

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check be made and that enough time is allowed to perform complete and satisfactory testing.

⚠ WARNING

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

Software testing must be done in both simulated and real environments.

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:
- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

OPERATION AND ADJUSTMENTS

The following precautions are from the NEMA Standards Publication ICS 7.1-1995 (English version prevails):

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.
- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer's instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer's instructions and the machinery used with the electrical equipment.
- Only those operational adjustments actually required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

About the Book

At a Glance

Document Scope

This document describes the ETEST framework of the Logic Builder.

Validity Note

This document has been updated for the release of SoMachine V4.3.

Related Documents

Title of documenation	Reference number
PD_ETest Library Guide	*EIO0000002394 (ENG)*; *EIO0000002395 (FRE)*; *EIO0000002396 (GER)*; *EIO0000002397 (ITA)*; *EIO0000002398 (SPA)*; *EIO0000002399 (CHS)*
SoMachine Programming Guide	*EIO0000000067 (ENG)*; *EIO0000000069 (FRE)*; *EIO0000000068 (GER)*; *EIO0000000071 (SPA)*; *EIO0000000070 (ITA)*; *EIO0000000072 (CHS)*

You can download these technical publications and other technical information from our website at *http://www.schneider-electric.com/ww/en/download*.

Product Related Information

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Chapter 1
Introduction

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
General Information	14
ETEST Objects	15
Methods	17
Macros	21

General Information

Overview

ETEST is a framework to automate unit tests for program code with SoMachine.

Automated tests help to verify the correctness of the implemented functions. The consequential execution of unit tests after every modification of the program code helps to improve the quality. The ETEST framework helps simplify the definition and execution of reusable tests.

Theoretical Background

Unit tests are part of the software development process. They are used to verify the accuracy of the programming units of an application, for example, of functions or function blocks. In this context, you can verify whether a program complies with the predefined specification.

Unit tests are a suitable preliminary stage for integration tests. Integration tests, in turn, are suitable for testing several interdependent units working together. As opposed to unit tests, integration tests are executed manually.

Every modification of program code can lead to potential errors. To help avoid errors, execute all test cases every time you have modified the program code.

These repetitions (regression tests) help to verify that the modifications of the program code do not result in any unintended side effects on unmodified parts of the software, and that the modified system continues to comply with the specified requirements.

In the course of test-driven development, unit tests are generated and maintained in parallel to the application development. Automated, reproducible unit tests allow you to trace the effects of the modifications you made in the program code.

Library PD_ETest

The PD_ETest library provides a collection of assistant function blocks for the ETEST framework of the Logic Builder. When you create a **Test case** object in the Logic Builder, the PD_ETest library is automatically integrated in your project. Without this library, ETEST cannot be used.

Licensing

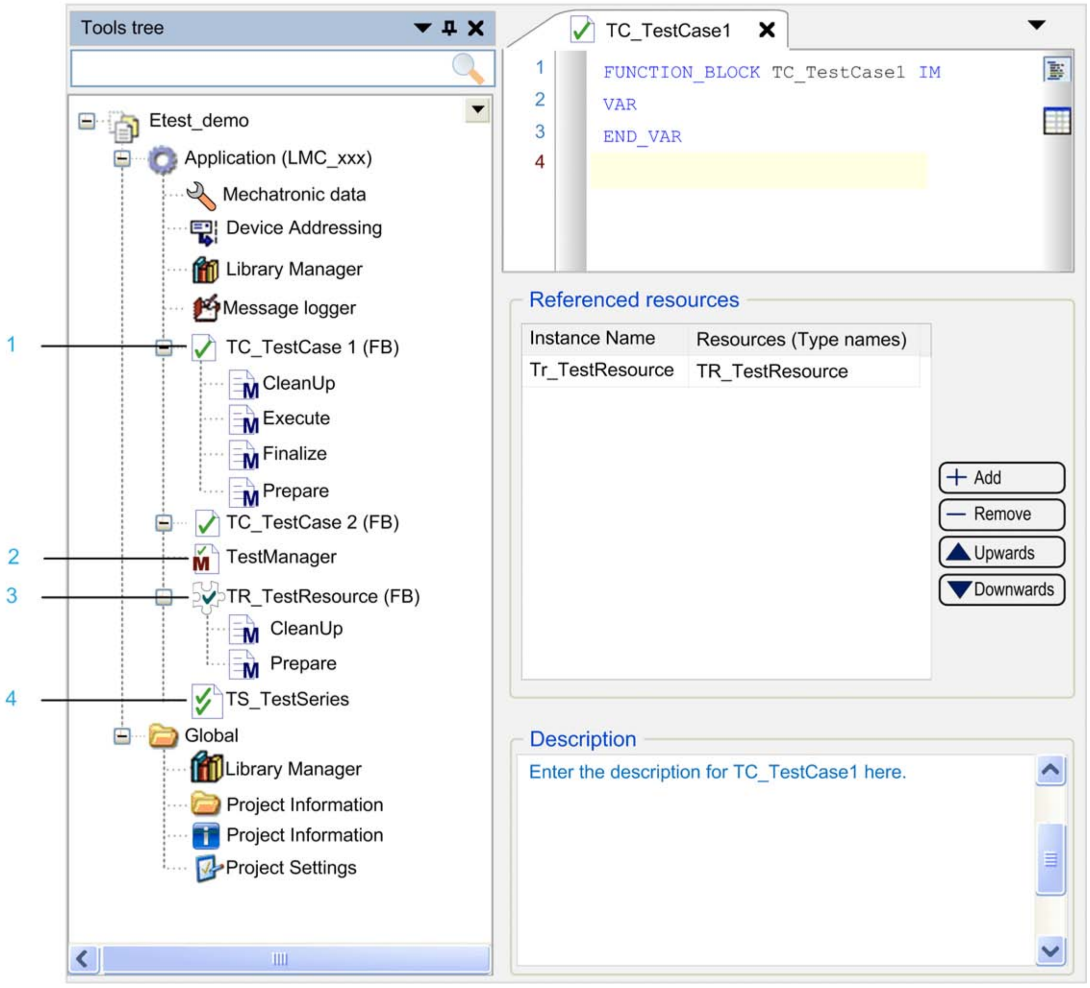
A valid license is required to run the ETEST framework.

After successful installation, the 42-days trial license is automatically activated if a permanent license is not available. To activate a permanent license, start the Schneider Electric **License Manager** software that is installed along with the ETEST framework.

ETEST Objects

Overview

The graphic shows the ETEST objects available in the Logic Builder **Tools tree** as subnodes of an **Application** node:



```
FUNCTION_BLOCK TC_TestCase1 IM
VAR
END_VAR
```

Instance Name	Resources (Type names)
Tr_TestResource	TR_TestResource

Enter the description for TC_TestCase1 here.

ETEST object	Description	Legend item
TestCase	● Central component of ETEST. ● Basically a function block. It uses the object-oriented extensions of the IEC 61131-3 standard that are available in Logic Builder. ● Implements the interface `IF_TestCase`. ● Contains methods *(see page 17)*, which in turn can use macros *(see page 21)*.	1
TestManager	● A test manager is an administration function block. Test cases and test series are performed by this function block as a sub process of the associated application. ● If tests shall be executed for an application, the following conditions must be fulfilled: ❍ A **TestManager** object must be available as a subnode of the **Application** node. ❍ A task must reference this object. Both are automatically created if you insert an ETEST object under an **Application** node.	2
TestResource	● Encapsulates the access to as well as the initialization of hardware and larger data structures used by test cases. ETEST creates only one instance per resource. ETEST initializes the resources before they are used (by calling their `Prepare` method). ETEST de-initializes resources after they have been used (by calling their `CleanUp` method). ● Before a test case is executed, ETEST verifies that the resources integrated into this test case are initialized and that the case is given a reference to the instance of the resource. After the test case has been completed, ETEST verifies that the resource is de-initialized again. ● Further resources can be integrated in resources (as in test cases). Cyclical dependencies between resources are not allowed.	3
TestSeries	● Can be used for grouping test cases. ● Can reference test cases or further test series.	4

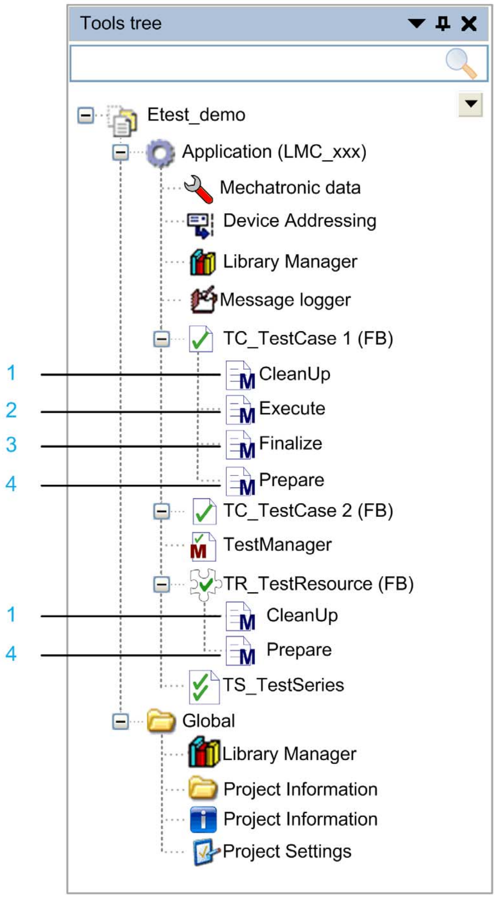
NOTE:
An ETEST object can be executed if the following conditions apply:
- It is a **TestCase** or a **TestSeries** object that is a subnode of the **Application** node.
- It has been selected by an executable **TestSeries**.

Example: A **TestCase** which is a node of the **Tools tree** can be executed if it is selected in a **TestSeries** which is a subnode of an **Application** node.

Methods

Overview

Test cases and resources contain predefined methods. These methods are called by the test framework during test execution. The contents of these methods, the test logic, is programmed by you as a user. The methods predefined by ETEST do not differ in their behavior from the general methods described in the SoMachine Programming Guide *(see SoMachine, Programming Guide)*.



1 `CleanUp`: Cleaning after the test case.
2 `Execute`: Execution of the test case.
3 `Finalize`: Cleaning after the test case. (Called once in the cycle in which `Execute` is terminated.)
4 `Prepare`: Preparation of the test case.

Methods of Test Cases and Resources

Test cases contain the methods:
- `Prepare`
- `Execute`
- `Finalize`
- `CleanUp`

Resources contain the methods:
- `Prepare`
- `CleanUp`

These methods must be included in every test object. In addition, test cases and resources may contain any number of other methods.

NOTE: The methods of test cases and resources must only be written in structured text (ST). Otherwise an error may be generated.

Sequence of Methods During Test Execution

During the execution of a test case, the methods are called by the ETEST framework in the following sequence, regardless of the order that they appear in the **Tools tree**:

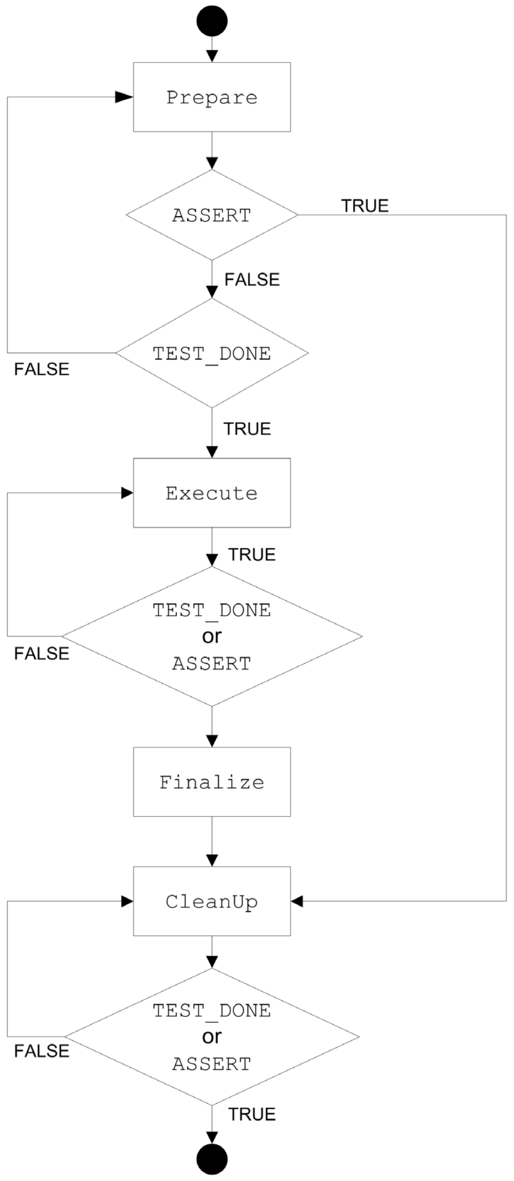
Sequence	Method	Call type	Description	End condition
1	`Prepare`	Cyclic	Preparatory measures for the test are executed, for example, initialization of variables or positional control of an axis.	● The macro ASSERT has been evaluated to FALSE. ● Initialization has been successfully completed. (The macro `TEST_DONE` has been called.)
2	`Execute`	Cyclic	Contains the test itself.	● The macro ASSERT has been evaluated to FALSE. ● The test has been successfully completed. (The macro `TEST_DONE` has been called.)
3	`Finalize`	Once	The method is called up if the following conditions apply: ● The method `Execute` is exited with an error. ● The method `Execute` is exited regularly at the end of the test case in the same cycle. ● The method `Execute` is canceled by user input.	Is directly followed by the `CleanUp` method.
4	`CleanUp`	Cyclic	Resets the test object to the initial state. It can then be reused later.	● The macro ASSERT has been evaluated to FALSE. ● The test has been successfully completed. (The macro `TEST_DONE` has been called.)

During the execution of a test case, precisely one method of a test case is called in each cycle. An exception is the method `Finalize`, which is used in the same cycle as the last call of `Execute`.

Within the methods of test cases and resources, macros *(see page 21)* can be used.

Calling Methods

The diagram shows how the ETEST framework calls test methods:



Macros

Overview

Specific macros are available in the methods *(see page 17)* of ETEST objects. You can use these macros to control the test execution and determine the result of a test case.

Macros are called in the same way as methods. In contrast to a function call, a macro call can write variables of the calling method and can trigger a return of the calling method.

Macros are allowed in the following methods of test cases and resources:
- `Prepare`
- `Execute`
- `CleanUp`

You can also use macros in methods that are called with `ASSERT_SUBMETHOD`.

NOTE: Using macros in other methods of test cases and resources can lead to errors. In particular, do not use macros in the method `Finalize`.

Macros in other objects do not have any effect. This is indicated by a message that is displayed in the **Messages** view.

Macros in the Methods of Test Cases and Resources

In the methods of test cases and resources, the following macros are possible:

Macro call-up	Description
`ASSERT(condition)`	FALSE: Execution is not completed successfully. TRUE: Execution is completed successfully.
`ASSERT(condition, message)`	FALSE: Execution is not completed successfully. TRUE: Execution is completed successfully. `message` is the diagnostic message that is displayed as a result.
`ASSERT_INCONCLUSIVE(condition, message)`	FALSE: Test result is inconclusive. The test result is inconclusive to indicate that a test could not be carried out. The cause may be, for example, that a hardware device required for the test is not available on the test system. TRUE: Test is completed successfully.
`ASSERT_SUBMETHOD(method_call(i_xFirstCall, i_ifErrorLogger))`	With `Method_call`, methods can be called which in turn can use macros in order to influence the control flow of the calling method. FALSE: An error has been detected during the method call-up. An `ASSERT` in a method called by `ASSERT_SUBMETHOD` that has been evaluated to FALSE has the same effect as an `ASSERT` that has been evaluated to FALSE directly in the calling method. In contrast to this, an `ASSERT` in a method called without using `ASSERT_SUBMETHOD` does not have any effect on the calling method. When you use the `ASSERT_SUBMETHOD`, make sure to use the exact syntax concerning the parameter declaration and return type (also refer to example 2 *(see page 23)*): ● Declare `i_ifErrorLogger` in the submethod analog to the `Execute` method. ● Declare and use the return type `ET_TestReturn` in the submethod analog to the `Execute` method. ● The `i_ifErrorLogger` has to be transferred to the submethod as well.
`TEST_DONE()`	The execution of the test is completed successfully (without any errors detected). A simple `RETURN` only ends the current cycle. The method will then be called again in the next cycle.

Macros Stopping the Execution of Methods

The methods `Prepare`, `Execute` and `CleanUp` of a test case or a resource are called cyclically until an `ASSERT` is evaluated to FALSE or `TEST_DONE` is called. If a `TEST_DONE` is called or if an `ASSERT` is evaluated to FALSE, the execution of the method stops, cancelling the rest of the execution of the method.

If...	Then ...
the execution of a test is ended via `TEST_DONE`	the test has been successfully completed.
an assertion in the method `Execute` or `Prepare` is evaluated to FALSE	the test case did not complete successfully.

NOTE: If an assertion in the method `CleanUp` is evaluated to FALSE, the execution of the test is stopped. This is categorized as an error during initialization or de-initialization. A message is displayed in Logic Builder. It indicates that the test has been canceled and that a reset of the controller is required.

Example 1

In the following program code example, the macros `ASSERT` and `TEST_DONE` are used in the implementation part of the `Execute` method.

```
diResult := MyAdditionFunction(4, 3);
ASSERT(diResult = 7,  'MyAdditionFunction addition result not correct')
;
TEST_DONE();
```

Example 2

In the following program code example, the macros `ASSERT` and `TEST_DONE` are used to test the function `my_add` (a simple addition).

In the test case, a `TestMyAdd` method was created. It is called multiple times by the `Execute` method of the test case to test the function `my_add` under different boundary conditions and with different parameters.

```
(* Methode Execute *)
(* Declaration part*)
METHOD Execute : PD_ETest.ET_TestReturn
VAR_INPUT
   i_xFirstCall : Bool;
   i_ifErrorLogger : PD_ETest.IF_ErrorLogger;
END_VAR
VAR_OUTPUT
   q_rProgress : REAL;
   q_sState . STRING[255];
```

```
END_VAR
VAR
END_VAR
(* Implementation part*)
ASSERT_SUBMETHOD (TestMyAdd(12, 2, 14, i_ifErrorLogger));
ASSERT_SUBMETHOD (TestMyAdd(12, 3, 15, i_ifErrorLogger));
ASSERT_SUBMETHOD (TestMyAdd(12, 4, 16, i_ifErrorLogger));
TEST_DONE();

(* Submethod TestMyAdd *)
(* Declaration part *)
METHOD TestMyAdd : PD_ETest.ET_TestReturn
VAR_INPUT
   i_a : INT;
   i_b : INT;
   i_result : INT;
   i_ifErrorLogger : PD_ETest.IF_ErrorLogger;
END_VAR
VAR
   iResult : INT;
END_VAR
(* Implementation part *)
iResult : my_add(i_a, i_b);
ASSERT(iresult = i_result);

(* Function my_add *)
(* Declaration part *)
FUNCTION my_add : INT
VAR_INPUT
   a : INT;
   b : INT;
END_VAR
VAR
END_VAR
(* Implementation part *)
iResult : my_add := a + b;
```

Chapter 2
Using ETEST

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Add an ETEST Object	26
Edit Test Cases	27
Use Test Series	28
Use Test Resources	30
Execute Test	32
Record Measured Values	33
Evaluate Test Results	34

Add an ETEST Object

Add an ETEST Object

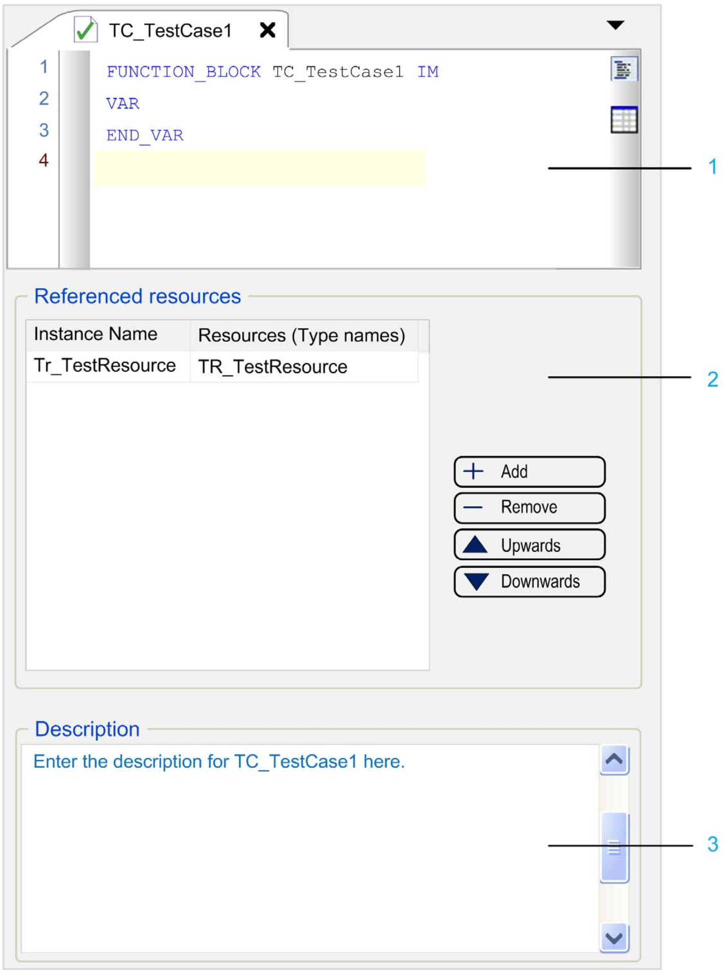
To add an ETEST object to your project, proceed as follows:

Step	Action
1	Right-click the **Application** node in the **Tools tree** and execute the **Add Object...** command from the context menu.
2	Select the ETEST object you want to add (**Test Case**, **Test resources**, **Test Series**). **Result:** The **ETEST Object** dialog box opens.
3	Rename your object by typing a name in the box **Name**. **NOTE:** Choose a name that complies to the IEC standard. Do not use special characters, leading digits, or spaces within the name. The name must not exceed a length of 32 characters. If you do not rename the object, a name is given by default.
4	Click the **Add** button. **Result:** The selected ETEST object is added to the project and appears as a new node in the **Tools tree**.

Edit Test Cases

Overview

The structure of test cases is similar to function blocks, but there is no editor available for the implementation section. Only the methods of test cases are implemented. To implement the test case, use the method editor by double-clicking the method in the **Tools tree**.



```
FUNCTION_BLOCK TC_TestCase1 IM
VAR
END_VAR
```

Referenced resources

Instance Name	Resources (Type names)
Tr_TestResource	TR_TestResource

Description

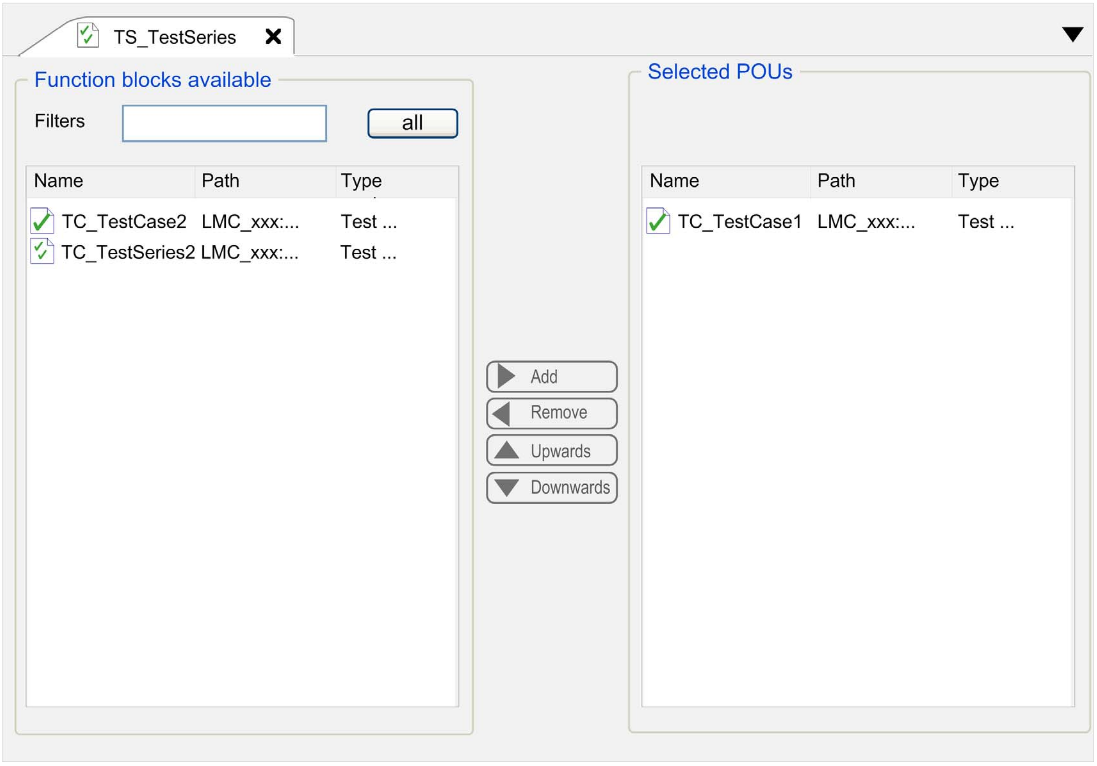
Enter the description for TC_TestCase1 here.

1 The declaration editor corresponds to the declaration section of a function block. You can switch between the text view and the table view of the editor using the buttons **Textual** and **Tabular**.
2 The **Referenced resources** area provides integrated test resources.
3 The **Description** area provides space for explanations regarding the test case.

Use Test Series

Overview

One project comprises several test cases. The **TestSeries** editor allows you to group several test cases into a test series to execute them in a sequence.



Function blocks available

Name	Path	Type
TC_TestCase2	LMC_xxx:...	Test ...
TC_TestSeries2	LMC_xxx:...	Test ...

Selected POUs

Name	Path	Type
TC_TestCase1	LMC_xxx:...	Test ...

The **TestSeries** editor consists of two areas:

Area	Description
Function blocks available	This list includes test cases and test series that are subnodes of the same **Application** node as the open test series. It also lists those test cases and test series that are located in the **Tools tree**. You cannot select a test series as a subnode of itself. Cyclical references between test series are not allowed.
Selected POUs	The selected POUs are displayed. The POUs are the test cases. A test series can select a POU directly only once. Indirectly, POUs can be selected several times. It is thus permissible that test series may embed other test series, and that these embedded test series in turn contain the same test case.

Button / box	Description
Add	Moves a selected test case from the list **Function blocks available** into the list **Selected POUs**.
Remove	Moves a selected test case from the list **Selected POUs** to the list **Function blocks available**.
Upwards	Moves a test case in the area **Selected POUs** upward in the list.
Downwards	Moves a test case in the area **Selected POUs** downward in the list.
Filters all	Enter a text in the **Filters** field to filter the list of **Function blocks available** in accordance with the entry. Click the button **all** to clear the filter.

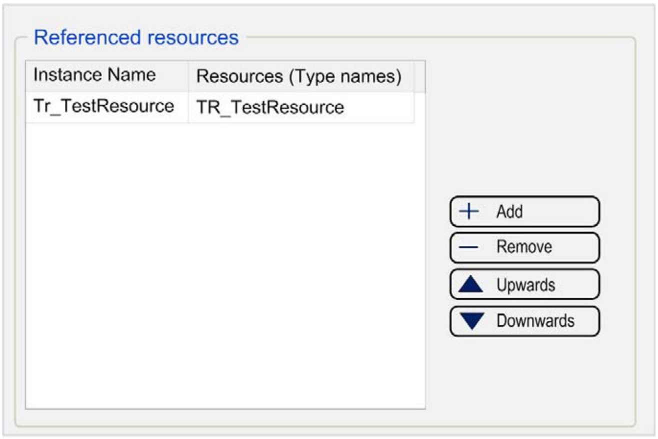
Mouse or Keyboard Action	Description
Double-click	Double-click an entry in the **Function blocks available** list to shift it to the **Selected POUs** list and vice versa.
Delete key	Moves a selected test case from the list **Selected POUs** to the list **Function blocks available**.
Ctrl+A shortcut	Selects all entries of the list.

Use Test Resources

Overview

A test resource encapsulates the access as well as the initialization of hardware and larger data structures used by test cases. Resources are created and processed just as test cases *(see page 27)*. A test case or a resource can bind a resource.

The declaration section for resources (**Referenced resources**) is located in the central area of the editor.



Instance Name	Resources (Type names)
Tr_TestResource	TR_TestResource

Button	Description
+ Add	Adds a test resource.
- Remove	Removes a test resource.
Upwards	Moves a selected test resource upward within the list.
Downwards	Moves a selected test resource downward within the list.

Mouse or Keyboard Action	Description
Double-click	Double-click an entry in the list of **Referenced resources** to edit it.
Delete key	Removes the test resource.
Ctrl+A shortcut	Selects the entries of the **Referenced resources** list.
Ctrl+click or **Shift**+click shortcut	Selects several entries of the **Referenced resources** list.

Each line of the table comprises a referenced resource. The column **Instance name** contains the name of the resource. This name is used if the resource is called from within a test object: A variable of this name is generated, which holds a reference for the resource.

The column **Resources (Type names)** contains the type name of the resource. This is the name of the test resource object which is to be integrated. In the online view, resources are displayed together with the other variables.

Execute Test

Overview

This section describes how to execute test cases and test series *(see page 28)*.

ETEST consists of two main test processes. One process runs within the Logic Builder on the PC. The other process runs on the controller. The controller process is integrated in the application which runs on the controller. Both processes communicate via a communication data structure.

Executing a Test

The following prerequisites must be fulfilled before a test can be executed:
- The application must contain a test object that is referenced in a task.
- The test case or the test series that shall be executed must be selected in the **Tools** tree.
- No other test must be running.

Execute a test as follows:

Step	Action
1	Right-click the node of the test case to be executed in the **Tools** tree. **Result**: The context menu opens.
2	Execute the command **Start test** from the context menu. **Result**: The test case is executed.

After you have started the test, the results view of the test opens. It indicates the progress of the test. You can stop a running test by clicking the **Cancel test run** icon in the toolbar.

NOTE: While a test is running, observe the following:
- Do not disconnect the controller.
- Do not log out.
- Do not stop or reset the application.
- Do not edit the project.

Doing so interrupts the test logic rendering any result invalid.

Record Measured Values

Overview

The example in this chapter illustrates how measured values can be recorded.

To achieve this, the following programming code must be called in a method that is executed within a test.

Example

```
TestManager.fbMeasurandList.AddMeasurand('module', 'name', ET_Measurand
Types.E_NotClassified, 'value');
```

For further information, refer to:

- `FB_MeasurandList` *(see SoMachine, PD_ETest, Library Guide)*
- `ET_MeasurandTypes` *(see SoMachine, PD_ETest, Library Guide)*

Every time a command is executed, a new measured value is recorded.

`TestManager` is the name of **TestManager**-object *(see page 15)* of the application.

Evaluate Test Results

Overview

The ETEST framework provides the test results of a completed test and shows the progress of a running test in the **Test results** view.

NOTE: If test cases are built for function blocks contained in libraries, verify the compatibility of any library updates as it concerns your test cases.

Executed Tests

The **Tests runs** list contains the tests that have been executed since Logic Builder has been started.

Buttons	Description
Export	Stores selected test results in XML format. (**.testresult*).
Import	Opens test results stored in XML format.
Print report	Opens an HTML report in the web browser.

Test Results

Click an entry in the **Tests runs** list to open the test in the area **Test results**.

Imported test results are also opened in the area **Test results** and shown in the list of the most recently executed tests. The area **Test results** is automatically opened when a test is started. It shows the result (or the state) of a test.

The area **Test results** provides two sections.
- At the top, there is the summary of the test results that contains the name of the executed POU and the total result.
- Below, there is a section with various tabs:

Tab	Description
Test results	Tree view which shows the test series and test cases. It displays the individual results for each node.
User comment	User name and comment. Shows by default the Windows user name. You can add a comment to the **Comment** box. This data is stored together with the test result.
Products	Lists the libraries available in the project during the test.
Measured values	Lists measured values that were recorded during the execution of the test. Used to measure time values and the size of data structures.

Glossary

A

application
A program including configuration data, symbols, and documentation.

C

configuration
The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

controller
Automates industrial processes (also known as programmable logic controller or programmable controller).

E

expansion bus
An electronic communication bus between expansion I/O modules and a controller.

I

I/O
(*input/output*)

P

program
The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

Index

A

add
- ETEST objects, *26*

C

CleanUp method, *17*

E

edit test cases, *27*
ETEST
- add objects, *26*
- examples, *21*
- general information, *14*
- macros, *21*
- objects, *15*

examples
- method of test case, *23*

Execute method, *17*
executing
- test, *32*

F

Finalize method, *17*

M

macros
- ETEST, *21*

O

objects
- ETEST, *15*

P

Prepare method, *17*

R

recording measured values, *33*
results
- ETEST, *34*

T

test
- cases, *15*
- cases, edit, *27*
- executing, *32*
- managers, *15*
- managers, recording measured values, *33*
- resources, *15*
- resources, use, *30*
- results, *34*
- series, *15*
- series, use, *28*

U

use
- test resources, *30*
- test series, *28*