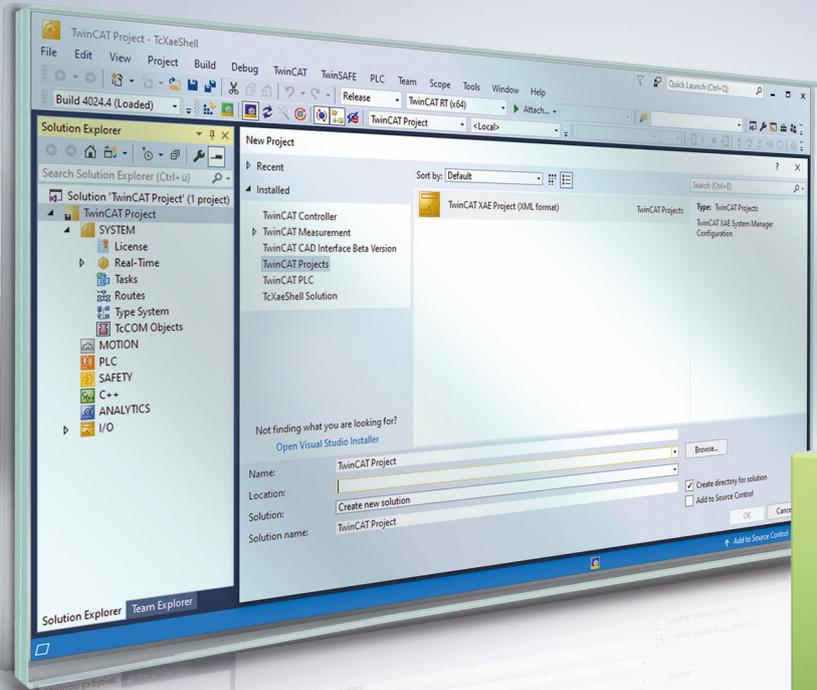


# BECKHOFF New Automation Technology

Manual | EN

# TF1910

TwinCAT 3 | UML





# Table of contents

|   |           |
|---|-----------|
| <b>1 Foreword</b> .....                           | <b>5</b>  |
| 1.1 Notes on the documentation .....              | 5         |
| 1.2 For your safety .....                         | 5         |
| 1.3 Notes on information security.....            | 7         |
| <b>2 Overview</b> .....                           | <b>8</b>  |
| <b>3 Installation</b> .....                       | <b>9</b>  |
| 3.1 System requirements .....                     | 9         |
| 3.2 Installation .....                            | 9         |
| <b>4 Settings</b> .....                           | <b>10</b> |
| 4.1 Options.....                                  | 10        |
| 4.2 UML compiler version .....                    | 11        |
| <b>5 Commands</b> .....                           | <b>15</b> |
| 5.1 Create bitmap.....                            | 15        |
| 5.2 Grid Enabled .....                            | 16        |
| 5.3 Grid Disabled .....                           | 16        |
| <b>6 UML class diagram</b> .....                  | <b>17</b> |
| 6.1 Basic principles .....                        | 17        |
| 6.2 Commands.....                                 | 19        |
| 6.2.1 Creating a new class diagram.....           | 19        |
| 6.2.2 Adding existing elements to a diagram ..... | 20        |
| 6.2.3 Editing a class diagram .....               | 22        |
| 6.3 Editor.....                                   | 24        |
| 6.4 Elements .....                                | 25        |
| 6.4.1 Class .....                                 | 25        |
| 6.4.2 Interface .....                             | 31        |
| 6.4.3 Global Variable List.....                   | 34        |
| 6.4.4 User-defined data type.....                 | 36        |
| 6.4.5 Variable declaration .....                  | 39        |
| 6.4.6 Property.....                               | 39        |
| 6.4.7 Method .....                                | 40        |
| 6.4.8 Action .....                                | 41        |
| 6.4.9 Composition .....                           | 41        |
| 6.4.10 Association.....                           | 44        |
| 6.4.11 Implementation.....                        | 46        |
| 6.4.12 Generalization .....                       | 48        |
| 6.4.13 Note.....                                  | 50        |
| <b>7 UML state diagram</b> .....                  | <b>51</b> |
| 7.1 Basic principles .....                        | 51        |
| 7.2 Commands.....                                 | 54        |
| 7.2.1 Creating a new Statechart.....              | 54        |
| 7.2.2 Editing Statechart.....                     | 55        |
| 7.2.3 Go to definition .....                      | 58        |
| 7.2.4 Find All References.....                    | 58        |

|          |  |           |
|----------|--|-----------|
| 7.2.5    | Add Watch.....                           | 58        |
| 7.3      | Editor.....                              | 58        |
| 7.4      | Elements.....                            | 59        |
| 7.4.1    | Start State.....                         | 59        |
| 7.4.2    | End State.....                           | 60        |
| 7.4.3    | State.....                               | 60        |
| 7.4.4    | Composite State.....                     | 64        |
| 7.4.5    | Fork.....                                | 74        |
| 7.4.6    | Choice.....                              | 76        |
| 7.4.7    | Transition.....                          | 77        |
| 7.4.8    | Completion Transition.....               | 79        |
| 7.4.9    | Exception Transition.....                | 80        |
| 7.4.10   | Note.....                                | 83        |
| 7.5      | Object Properties.....                   | 84        |
| 7.6      | Online Mode.....                         | 84        |
| <b>8</b> | <b>FAQ.....</b>                          | <b>87</b> |
| <b>9</b> | <b>Samples.....</b>                      | <b>89</b> |
| 9.1      | UML class diagram.....                   | 89        |
| 9.1.1    | 1 Basics.....                            | 89        |
| 9.1.2    | 2 Simple machine.....                    | 89        |
| 9.2      | UML state diagram.....                   | 90        |
| 9.2.1    | 1 Lamp.....                              | 90        |
| 9.2.2    | 2 Pedestrian traffic light.....          | 91        |
| 9.2.3    | 3 SaveText simulation.....               | 91        |
| 9.2.4    | 4 Call Behavior - Basis.....             | 93        |
| 9.2.5    | 5 Call behavior - transition action..... | 96        |

# 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.

For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.

The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

### Disclaimer

The documentation has been prepared 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.

No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

### Trademarks

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

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

### Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702  
and similar applications and registrations in several other countries.

## EtherCAT®

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

### Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The distribution and reproduction of this document as well as the use and communication of its contents without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

## 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 [here](#).

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](#).

## 2 Overview

### UML: general information

Unified Modeling Language (UML) is a graphical language that can be used for software analysis, design and documentation. UML is particularly suitable for object-oriented implementations. The unified modelling of the PLC application creates an easy-to-follow software documentation that can also be analyzed and discussed by departments other than the Software Development Dept.

### Diagram categories

Some of the UML diagrams can be categorized as structure diagrams, others as behavior diagrams. Structure diagrams are mainly used for static modelling and analysis, since they represent the software architecture schematically. Behavior diagrams are used for dynamic modelling. They are executable models, from which program code can be generated directly.

### UML in TwinCAT 3.1 PLC

With the integration of UML (Unified Modelling Language) in TwinCAT 3.1, two additional editors for modelling of PLC software are available. The existing TwinCAT PLC programming languages are extended by the UML classes and the UML Statechart.

-  : the functionality of the object **UML class diagram**
-  : Programming a POU in the implementation language **UML Statechart**

### UML class diagram

The UML class diagram belongs to the group of UML structure diagrams and can be used for schematic representation of the software architecture. In this way it is possible to represent object classes and the elements contained within them, as well as object relationships in a transparent manner.

### UML Statechart

The UML Statechart is part of the UML behavior diagrams and is used for dynamic software modeling. It can be used for a graphic specification of the dynamic response or the state-dependent system behavior. Compilation of the statechart generates program code, so that the state machine can be executed directly. The development process is supported by an online debugging option.

### Advantages

There are many advantages to using UML diagrams for the analysis, design and/or documentation of software. The essential aspects are covered in the following points:

- First of all, a graphic illustration that doesn't focus on technical details offers a good overview with which to check software requirements before the implementation. This avoids an incomplete or erroneous implementation of the application.
- The development of a well-conceived software architecture is greatly supported by the graphic illustration of the control code. Such an architecture forms the basis for the simple and goal-oriented implementation of even complex systems or requirements. Furthermore, a well-conceived software architecture can contribute towards the development of autonomous modules that can be reused, saving time and costs. In general, a well-planned software leads as a rule to fewer programming errors and thus to a higher code quality.
- Graphic access to the software facilitates maintenance and debugging.
- An generally understandable documentation of the software is usually created with the help of UML diagrams. On the one hand this can be used as a coordination tool in the development team, for example to exchange ideas and concepts or to define requirements. On the other, UML diagrams can be used to illustrate the control application to other technology specialists, for example mechanical engineers or process technicians.

## 3 Installation

### 3.1 System requirements

UML class diagram:

| Development environment |
|-------------------------|
| TwinCAT v3.1.4018.16    |

UML statechart:

| Development environment | Target platform          | Library placeholder to include |
|-------------------------|--------------------------|--------------------------------|
| TwinCAT v3.1.4016.0     | PC or CX (x86, x64, ARM) | UML statechart types           |

### 3.2 Installation

**Engineering:**

The engineering components of the function "TF1910 | TC3 UML" are contained in the TC3.1 XAE setup (both UML classes and UML Statechart).

You can use these directly after installing the XAE. No license is required for this.

**Runtime:**

If the UML Statechart is used, the required runtime components are installed with the TC3.1 XAE or XAR setup.

You will require the TF1910 runtime license for the additional runtime component UML Statechart. For further information please refer to the documentation on Licensing.

You can test the UML Statechart with a 7-day test license without the license for TF1910. You can continuously regenerate the trial license during the test phase. Refer here also to the documentation for the TwinCAT-3 test licenses.

## 4 Settings

### 4.1 Options

In the TwinCAT UML options (**Tools > Options > TwinCAT > PLC Environment > UML**) you can configure the settings relating to the UML editors for the entire project. Modified options take effect when the dialog is closed, even in UML editors that are already open.

#### General

The screenshot shows the 'General' tab of the UML Options dialog. It features three tabs: 'General', 'Statechart', and 'Class Diagram'. Under the 'General' tab, there is a 'Grid space' input field with a value of 10 and a spin button. Below this, there are two checked checkboxes: 'Snap to grid' and 'Display grid points'.

- **Gridspace:** enter a whole number that will be used as the gridspace in pixels. Default: 10
- **Snap To Grid:** activate this option in order to align all the elements in the UML editors to the grid.
- **Display grid points:** activate this option in order to display the grid points in the UML editors.

#### Statechart

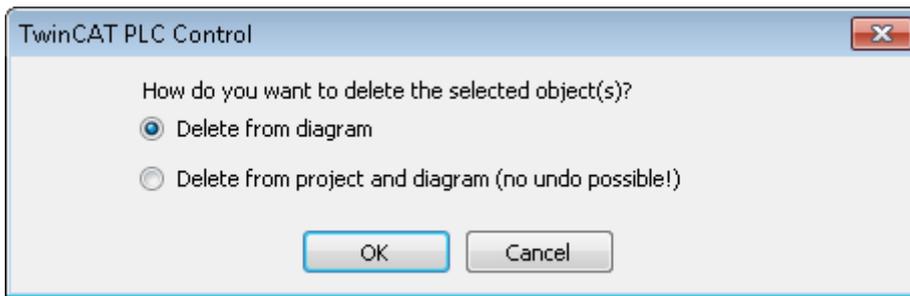
The screenshot shows the 'Statechart' tab of the UML Options dialog. It features three tabs: 'General', 'Statechart', and 'Class Diagram'. Under the 'Statechart' tab, there is a single unchecked checkbox labeled 'Position Condition and Action of a relation together'.

- **Position Condition and Action of a relation together:**  
Activate the option to synchronously move a guard condition and an action belonging to the same transition in the statechart.

#### Class diagram

The screenshot shows the 'Class Diagram' tab of the UML Options dialog. It features three tabs: 'General', 'Statechart', and 'Class Diagram'. Under the 'Class Diagram' tab, there are three checkboxes: 'Show prompt when deleting objects from diagram' (checked), 'Skip Refactoring Preview' (unchecked), and a section titled 'Hiding variables' containing 'Hide internal variables' (unchecked) and 'Hide signature variables' (unchecked).

- **Show prompt when deleting objects from diagram:**  
Objects can be deleted either from the diagram only or from the diagram and from the project. There are two possible procedures for this:
  - If an object is marked in the class diagram, two command symbols appear above the object in order to delete the object from the diagram only or from both the diagram and the project.
  - Alternatively a marked object can be deleted by pressing the [Del] key. If the option to display a selection window is disabled, the object is deleted by default only from the diagram. If the setting is activated, a selection window appears when deleting with which you can configure whether the object should be deleted only from the diagram or also from the project.



- **Skip Refactoring Preview:**

If this option is activated and refactoring is initiated in the diagram, the project-wide change is carried out without first opening the **Refactoring** dialog with a preview of all change points.

## Hide variables



Available from TC3.1 Build 4026

To reduce the amount of information within the class diagram to the desired focus, the following options are available.

- **Hide internal variables:**

If this option is enabled, internal variables are not displayed in the class diagram. These include VAR, VAR\_TEMP, VAR\_STAT and VAR\_INST.

- **Hide signature variables:**

If this option is enabled, signature or interface variables are not displayed in the class diagram. These include VAR\_INPUT, VAR\_OUTPUT and VAR\_IN\_OUT.

## 4.2 UML compiler version

The UML compiler version can be changed in the following dialogs:

- PLC project properties
- ProfileUpdate dialog

In addition, the following option is available with regard to the UML compiler version:

- Autoupdate UML Profile



The UML compiler version is only relevant if the UML Statechart language is used.

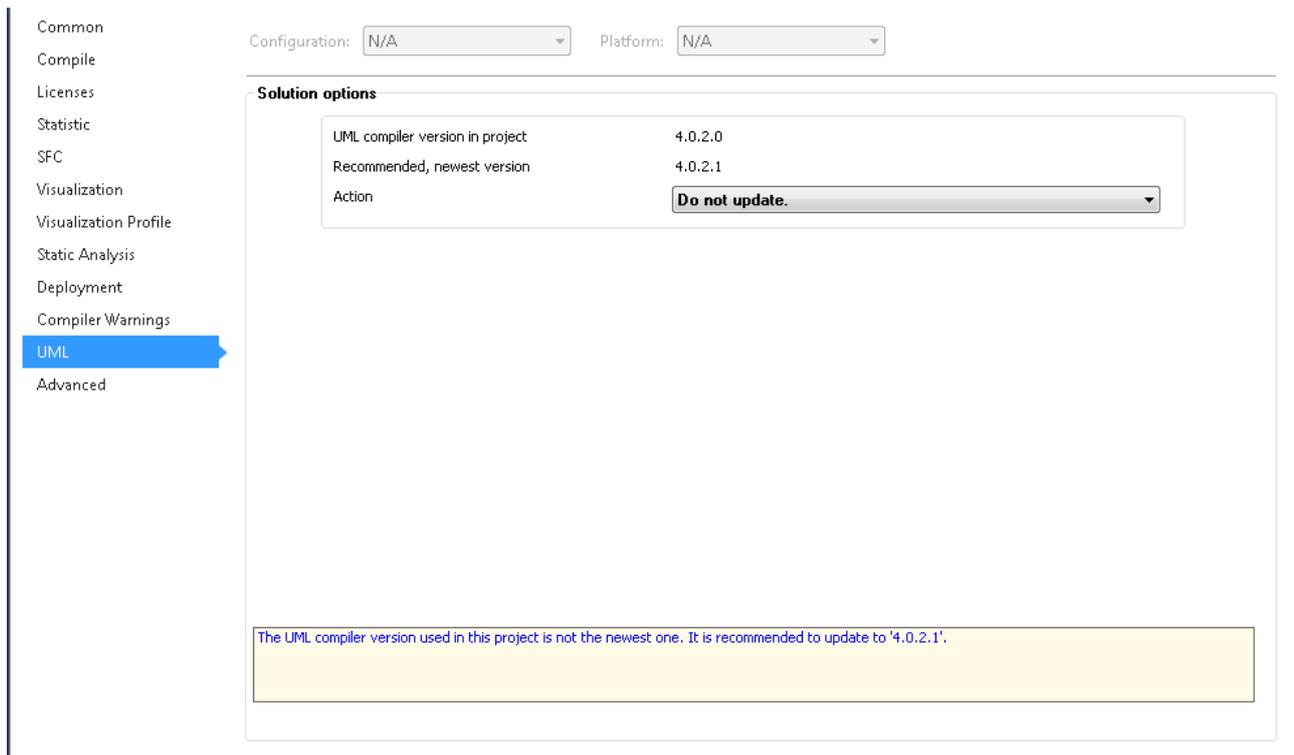


### Scope of the setting “UML compiler version”

The setting of the UML compiler version is a “solution option” and therefore does not only affect the PLC project, whose properties you are currently configuring, i.e. set UML compiler version applies to all PLC projects in the development environment.

## PLC project properties

You can change the UML compiler version in the properties of the PLC project. Open the PLC project properties and click on the category **UML**.



Configuration: N/A Platform: N/A

**Solution options**

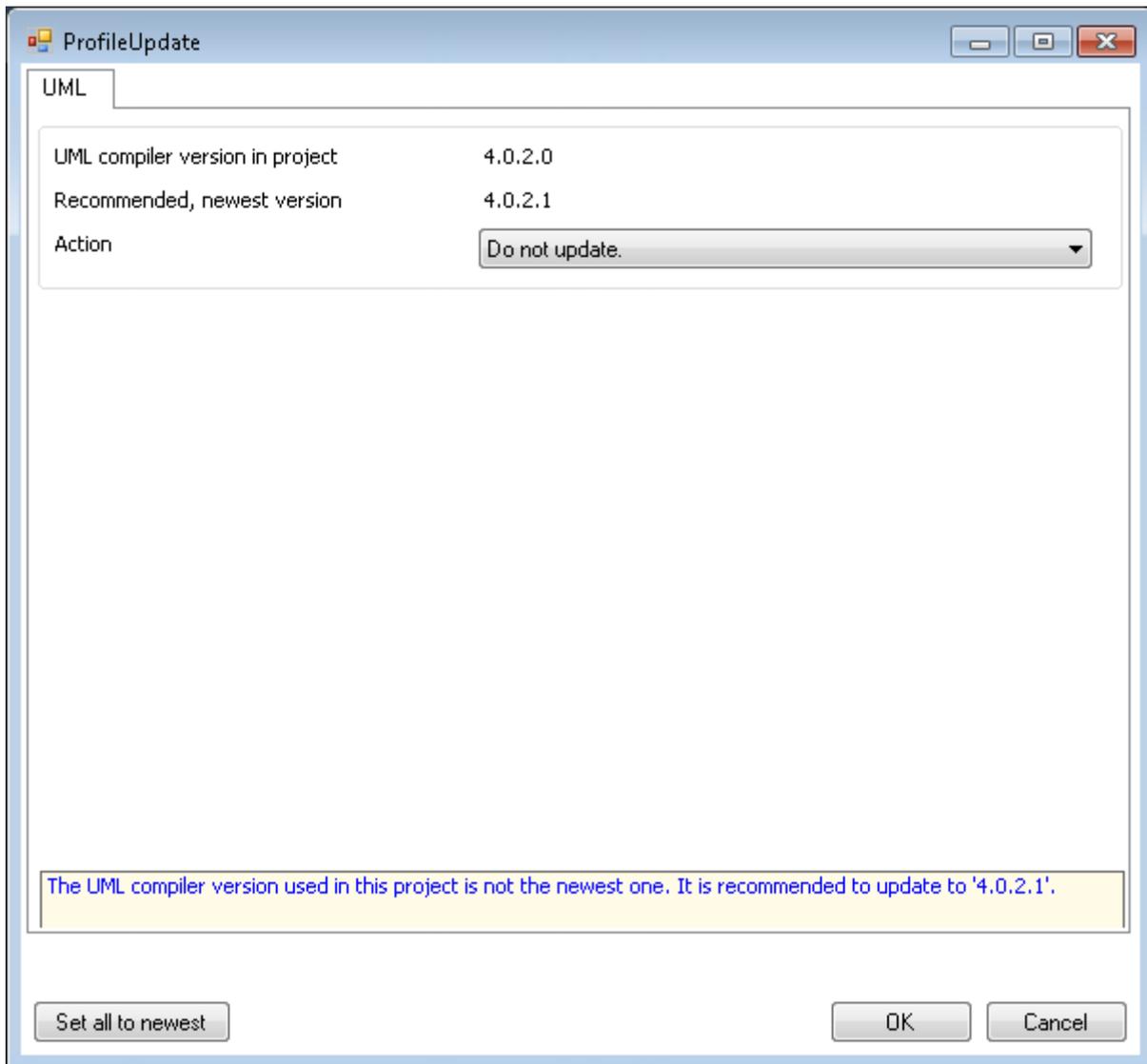
|                                 |                |
|---------------------------------|----------------|
| UML compiler version in project | 4.0.2.0        |
| Recommended, newest version     | 4.0.2.1        |
| Action                          | Do not update. |

The UML compiler version used in this project is not the newest one. It is recommended to update to '4.0.2.1'.

- **UML compiler version in project:** Indicates the UML compiler version currently used in the project.
- **Recommended, newest version:** Indicates the latest available UML compiler version, whose application is recommended.
- **Action:** In this dropdown menu you can select the required action. The action is executed directly when you select it. Example actions:
  - Do not update.
  - Update to 4.0.2.1

### ProfileUpdate dialog

If you open a PLC project, in which an outdated UML compiler version is used, a corresponding warning appears in the message window (“New version found for UML”). Double-click this warning to open the ProfileUpdate dialog, in which you can change the UML compiler version.



- **UML compiler version in project:** Indicates the UML compiler version currently used in the project.
- **Recommended, newest version:** Indicates the latest available UML compiler version, whose application is recommended.
- **Action:** In this dropdown menu you can select the required action. The action is executed when the dialog is confirmed via **OK**. Example actions:
  - Do not update.
  - Update to 4.0.2.1
- **Set all to newest:** Click the button to set the UML compiler version to the latest version.

### Autoupdate UML Profile

The option **Autoupdate UML Profile** is available in the **Advanced** category of the PLC project properties. It can be used to configure the automatic update behavior of the UML compiler version.

If you open a PLC project, in which an outdated UML compiler version is used, a corresponding warning appears in the message window ("New version found for UML").



In such a case, the UML compiler version is automatically set to the latest version, if the option **Autoupdate UML profile** is enabled. On such an automatic update of the UML compiler version, the message window shows a corresponding warning (e.g. "UML set from '4.0.2.0' to '4.0.2.1'").

: If the option **Autoupdate UML Profile** is disabled, the UML compiler version is not changed automatically. Double-click on the warning “New version found for UML” to open the [ProfileUpdate dialog \[► 12\]](#), in which you can change the UML compiler version manually.

The screenshot displays the 'UML' settings page in the Beckhoff software. On the left, a vertical navigation menu lists various settings categories: Common, Compile, Licenses, Statistic, SFC, Visualization, Visualization Profile, Static Analysis, Deployment, Compiler Warnings, UML, and Advanced. The 'Advanced' option is highlighted with a blue arrow. The main content area shows 'Configuration: N/A' and 'Platform: N/A' at the top. Below this, a section titled 'Solution options' contains three checkboxes: 'Secure Online Mode' (unchecked), 'Autoupdate Visu Profile' (unchecked), and 'Autoupdate Uml Profile' (checked).

## 5 Commands

### Common commands for all UML diagrams

If a UML diagram is selected in the project tree, the following command is available in the context menu:

- [Generate bitmap from the selected UML diagram \[▶ 15\]](#)

The alignment of the elements relative to a grid can be enabled or disabled within the editor of a UML diagram.

- [Grid activated \[▶ 16\]](#)
- [Grid deactivated \[▶ 16\]](#)

### 5.1 Create bitmap

The **Create bitmap** command is available when

- the focus is on the project tree, and
- a class diagram or statechart is selected in the project tree and the context menu is open

Use the command to create a bitmap from of a class diagram or statechart in BMP, PNG or JPG format and to save it in your file system.

When you execute the **Create bitmap** command, the associated dialog opens. This dialog offers the following configuration options.

#### Size of the longer side (in pixels):

- The value displayed refers to the edge length of the bitmap. Depending on the layout of the UML diagram, the bitmap is exported either in landscape or portrait format. If the diagram is in landscape format, the value contains the length of the bitmap. If the diagram is in portrait format, the value contains the height of the bitmap.

#### Save bitmap to desktop:

- : If this option is enabled, the object name is used as the file name, e.g. "FB\_UML\_SC.bmp" for an object with the name "FB\_UML\_SC". The BMP format is used as the file type.
  - Confirm the setting with [ OK ] to save the bitmap on the desktop.
 

**Notice** This option overwrites any existing files on the desktop with this name without warning.
  - Use [ Cancel ] to terminate the dialog without saving.
- : The name, the directory and the format of the bitmap are editable.
  - Confirming the setting with [ OK ] opens the standard dialog for saving a file. Enter a directory and file name there, select BMP, PNG or JPG as the format and exit the dialog with [ Save ]. The bitmap is saved in your file system.

#### Access by Automation Interface:

The **Create bitmap** command is reachable using the ConsumeXML method on the POU node via Automation Interface.

```
<TreeItem>
<PlcPouDef>
<Commands>
<CreateBitmapCommand>
<Active>true</Active>
<Parameters>
<FileName>d:\tmp\Bitmap.png</FileName>
<Width>1200</Width>
<Height>-1</Height> (* 1 adapts the height to the width to keep the ratio;
```

alternatively, you can enter a fix height which might deform the bitmap \*)

```
</Parameters>  
</CreateBitmapCommand>  
</Commands>  
</PlcPouDef>  
</TreeItem>
```

## 5.2 Grid Enabled

Symbol: 

If this command is available in the context menu of a focused class diagram or statechart, an element is aligned along the grid when changing its position. The option **Snap to grid with grid space** in the TwinCAT UML options (**Extras > Options > PLC Environment > UML**) is ticked.

If you perform the command, the diagram will switch to **Grid disabled**. The option won't be ticked.

See also:

- ["Grid Disabled" \[► 16\]](#)
- ["Options \[► 10\]"](#)

## 5.3 Grid Disabled

Symbol: 

If this command is available in the context menu of a focused class diagram or statechart, an element is positioned **without** a grid when changing its position. **Snap to grid with grid space** in the TwinCAT UML options (**Extras > Options > PLC Environment > UML**) is not ticked.

If you perform the command, the diagram will switch to **Grid enabled**. The option will be ticked.

See also:

- ["Grid Enabled" \[► 16\]](#)
- ["Options \[► 10\]"](#)

## 6 UML class diagram

In addition to the following information, please also note the [samples \[▶ 89\]](#) that give a first introduction to the tool.

### 6.1 Basic principles

The UML class diagram can be used to document, analyze, design and expand the structure of a (sophisticated) system. In the process, classes can be designed and relationships between them can be mapped. The clear representation of PLC program elements includes, among other things, inheritance and implementation relationships, so as to clearly visualizing interrelationships. A UML class diagram is therefore ideal for graphical system documentation and offers a comprehensible basis for conveying technical content.

The class diagram editor provides elements that map the object orientation of the project. Since the editor is embedded in the PLC area of the TwinCAT 3 development environment, automatic generation of code is possible. An extensive range of features and tools is integrated.

The class diagram can be used in two directions. The existing project structure can be imported into the class diagram, or selected elements of the existing project structure can be added to the class diagram. As a result, the already existing software architecture can be documented and analyzed. Secondly, the class diagram offers the possibility to change and expand existing PLC elements or the existing project structure. This modification can be carried out with the aid of the [class diagram editor \[▶ 24\]](#) and the associated [elements \[▶ 25\]](#) of the toolbox. These allow the software architecture to be changed and expanded and at the same time to be documented and analyzed.

| Terms from the object orientation  | Synonym in IEC 61131-3   |
|--|--|
| Class (UML: <i>class</i> )   | POU types: <ul style="list-style-type: none"> <li>• Program (PRG): PROGRAM</li> <li>• Function block (FB): FUNCTION_BLOCK</li> <li>• Function (FUN): FUNCTION</li> </ul>                       |
| Attribute (UML: <i>attribute</i> ) <ul style="list-style-type: none"> <li>• Internal variable</li> <li>• Parameter: {input}</li> <li>• Property: {property}</li> <li>• Output parameter: {output}</li> </ul> | Variable types: <ul style="list-style-type: none"> <li>• Variables: VAR</li> <li>• Input variables: VAR_INPUT</li> <li>• Property: PROPERTY</li> <li>• Output variables: VAR_OUTPUT</li> </ul> |
| Operation (UML: <i>operation</i> )   | <ul style="list-style-type: none"> <li>• Method: METHOD</li> <li>• Action</li> </ul>   |
| Interface (UML: <i>interface</i> )   | Interface: INTERFACE   |
|  | Global variable list (GVL): VAR_GLOBAL   |
|  | User-defined data type (DUT): TYPE   |

#### Synchronicity

Objects in the class diagram and the project are kept identical, so that user inputs affect both views. This means the corresponding objects in the project tree are changed automatically when objects are changed via the class diagram. On the other hand, modifications in the project tree automatically become visible in the class diagram, if the corresponding objects are shown in the class diagram.

#### Application options

In general, the following applies:

- Not all elements of a project have to be shown in the class diagram.
- Several class diagrams can be added to a project.

In the interest of clarity it is generally preferable to show only a few objects in a class diagram. In this way it is possible to create subject-related or project section-related class diagrams, as sample. The objects shown can have particular dependencies, so that the objects and their dependencies are clearly displayed. On the other hand, objects without explicit dependencies to one another can also be shown in a diagram, so that they can be compared in parallel.

Based on this, the class diagram can be used for various purposes:

- As design and development tool
- As analysis tool for existing projects
- As project navigator

Information on which commands or actions can be used for the individual purposes is provided below.

### As design and development tool

- Create a [new, empty class diagram \[▶ 19\]](#).
- [The class diagram can be edited \[▶ 22\]](#) by means of various action options.  
→ All inputs also affect the objects in the project and are immediately visible in the project tree.

### As analysis tool for existing projects

- [Add existing elements to the class diagram \[▶ 20\]](#).
- Analyze the existing project structure with the aid of the class diagram you created and [edit the diagram \[▶ 22\]](#) if required.  
→ All inputs also affect the objects in the project and are immediately visible in the project tree.

### As project navigator

- Open a class diagram and double-click on an element in the class diagram to open the corresponding editor.
- The declaration and implementation can be edited as usual, if required.  
→ Any changes of the declaration are automatically updated in the class diagram.

### Commands

The following commands or action options are available for the class diagram:

- [Creating a new class diagram \[▶ 19\]](#)
- [Adding existing elements to a diagram \[▶ 20\]](#)
- [Editing a class diagram \[▶ 22\]](#)

Also note the commands, which are available for all UML diagrams: [Common commands for all UML diagrams \[▶ 15\]](#)

### Access modifier



Available from TC3.1 Build 4026

The access modifier of a method or a property is displayed in the class diagram by means of a symbol. The following table shows which symbol stands for which access modifier.

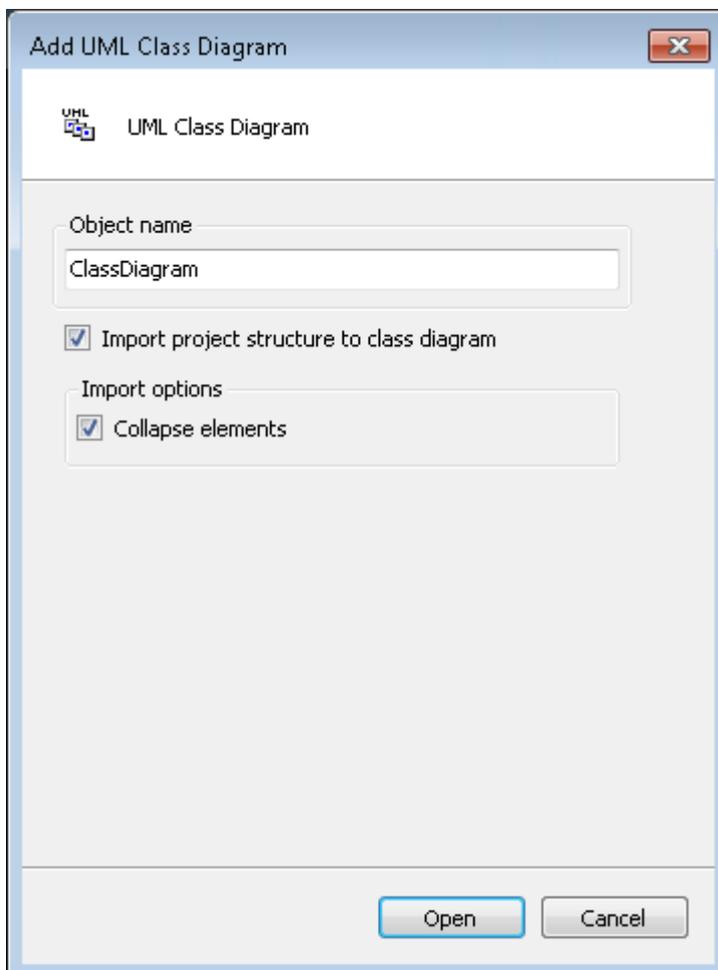
| Symbol | Access modifier |
|--------|-----------------|
| +      | PUBLIC          |
| #      | PROTECTED       |
| -      | PRIVATE         |
| ~      | INTERNAL        |

## 6.2 Commands

### 6.2.1 Creating a new class diagram

If you create a class diagram, you can optionally import the existing project structure. If this option is used, all relevant objects are imported into the class diagram. Relevant objects are programs, function blocks, functions, interfaces, DUTs, GVLs and their respective components (methods, properties, variables etc.).

1. In the context menu of the project tree select the command **Add object**, then select the **Class diagram** object.
2. In the **Add UML class diagram** dialog that opens enter a name for the class diagram.
3. **Enable** the option **Import project structure into class diagram** to import the existing project structure into the new class diagram.  
**Disable** the option to generate an empty class diagram.
4. Import options (only relevant for importing the project structure):  
**Enable** the option **Collapse elements** to display the element details (attribute or operation list) in minimized form.  
**Disable** the option to display the element details in expanded form.



5. Confirm the inputs and configurations with the **Open** button.  
⇒ The new class diagram object is added in the project tree, and the editor for the new diagram opens.

## 6.2.2 Adding existing elements to a diagram

Existing project elements can be added to a class diagram in a number of ways. On the one hand, several elements can be added simultaneously by importing the whole project structure or a selected folder via a command. On the other hand, an individual element can be added by dragging & dropping it onto the class diagram editor.

Depending on the number of elements, the following action options are available.

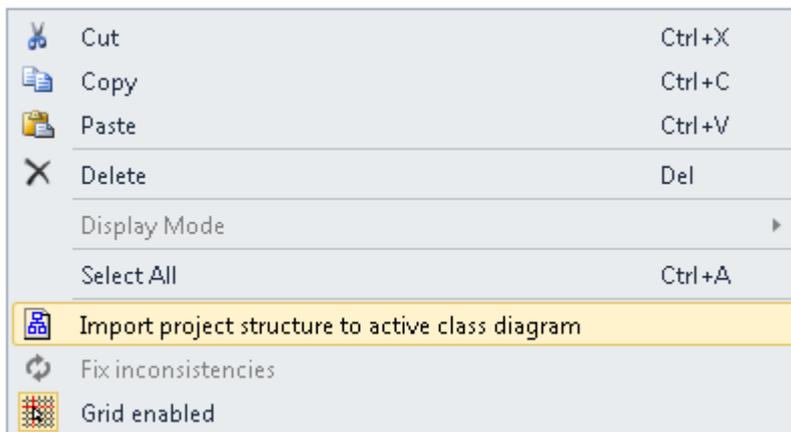
- Adding several existing elements:
  - Import whole project structure when a new diagram is created
  - Import whole project structure into an empty diagram
  - Import whole folder structure into an empty diagram
- Adding an individual existing element:
  - Visualize existing element from the project tree on the diagram
  - Visualize existing element from the cross-references on the diagram

### Import whole project structure when a new diagram is created

1. Create a new class diagram [► 19] and activate the option **Import project structure into class diagram**.
- ⇒ The new class diagram object is added to the project tree. The class diagram editor opens and shows the class diagram for the existing project.

### Import whole project structure into an empty diagram

1. Open an empty class diagram, which sits directly at the top project level in the project tree, not in a project folder.
2. Execute the command **Import project structure to active class diagram**, which is available in the context menu of the class diagram editor.

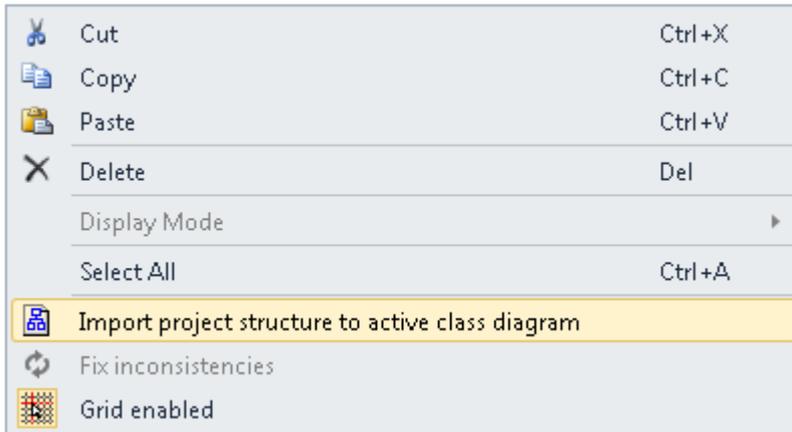


⇒ The class diagram shows the existing structure of the whole project.

### Import whole folder structure into an empty diagram

1. Open an empty class diagram from the folder whose structure you want to import into the class diagram.

- Execute the command **Import project structure to active class diagram**, which is available in the context menu of the class diagram editor.



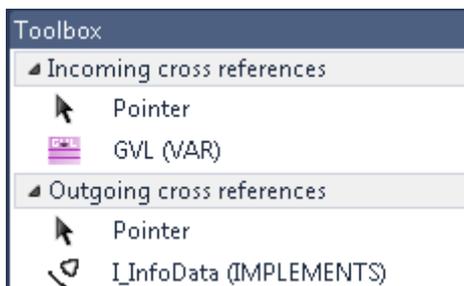
⇒ The class diagram shows the existing structure of the folder containing the class diagram.

### Visualize existing element from the project tree on the diagram

- Select an element of type POU, INTERFACE, GVL or DUT in the project tree and drag & drop it onto the opened class diagram. Drop it in a suitable location to visualize it there.
- ⇒ The corresponding element is shown in the diagram. If relationships with already shown elements exist, these are displayed automatically.

### Visualize existing element from the cross-references on the diagram

The class element, which has a relationship with the selected element but is not included in the class diagram, is shown in the **Toolbox** window under the heading **Incoming cross references** or **Outgoing cross references**. First, the relationship type that links the two elements is shown as a symbol. This is followed by the name of the target or source element. You can drag and drop the element onto the diagram, so that the element is shown in the class diagram.



### Showing cross-references

- Open the **Toolbox** window via the **View** menu.
  - Select a rectangle element in the opened class diagram, which has relationships that are not shown in the class diagram.
- ⇒ Under **Toolbox** the relationships with the elements are listed, which are not yet shown in the class diagram. Under **Incoming cross references** missing incoming relationships with source elements are listed. Under **Outgoing cross references** missing outgoing relationships are listed, together with target elements.

### Visualize existing element from the cross-references on the diagram

- Drag & drop the element, which is listed under **Incoming cross references** or **Outgoing cross references** and which is to be shown in the diagram, onto the class diagram. Drop it in a suitable location to visualize it there.

- ⇒ The corresponding element is shown in the diagram. The relationships with the already shown elements are displayed automatically.

### 6.2.3 Editing a class diagram

The actions available for editing a class diagram include the following. Further editing options can be found under [Editor \[► 24\]](#).

#### Adding a new element

1. Open the **Tools** window via the **View** menu.
  2. Select an [element \[► 25\]](#) in the **Tools** view and drag & drop it onto the opened class diagram. Drop it in a suitable location to visualize it there.
- ⇒ The new element is created in the project tree and shown in the diagram.

#### Deselect “Toolbox” view

- ✓ An element is selected in the **Toolbox** window. In the editor the cursor has the form of the selected element.
1. Press the right mouse button.
- ⇒ The element is deselected, and the standard **Pointer** element is selected.

#### Editing identifiers

1. Open the line editor of an identifier with two single clicks.
2. Enter a new name and confirm it via the [Enter] key or by clicking in an empty area of the diagram.



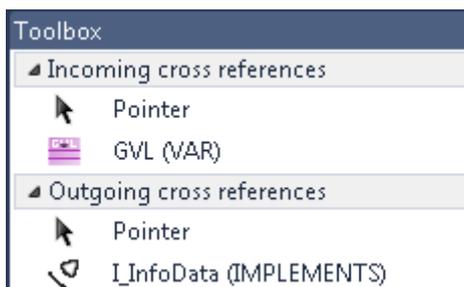
- ⇒ The identifier has the new name.

#### Visualize existing element from the project tree on the diagram

1. Select an element of type POU, INTERFACE, GVL or DUT in the project tree and drag & drop it onto the opened class diagram. Drop it in a suitable location to visualize it there.
- ⇒ The corresponding element is shown in the diagram. If relationships with already shown elements exist, these are displayed automatically.

#### Cross-references

The class element, which has a relationship with the selected element but is not included in the class diagram, is shown in the **Toolbox** window under the heading **Incoming cross references** or **Outgoing cross references**. First, the relationship type that links the two elements is shown as a symbol. This is followed by the name of the target or source element. You can drag and drop the element onto the diagram, so that the element is shown in the class diagram.



#### Showing cross-references

1. Open the **Toolbox** window via the **View** menu.
2. Select a rectangle element in the opened class diagram, which has relationships that are not shown in the class diagram.
  - ⇒ Under **Toolbox** the relationships with the elements are listed, which are not yet shown in the class diagram. Under **Incoming cross references** missing incoming relationships with source elements are listed. Under **Outgoing cross references** missing outgoing relationships are listed, together with target elements.

### Visualize existing element from the cross-references on the diagram

1. Drag & drop the element, which is listed under **Incoming cross references** or **Outgoing cross references** and which is to be shown in the diagram, onto the class diagram. Drop it in a suitable location to visualize it there.
  - ⇒ The corresponding element is shown in the diagram. The relationships with the already shown elements are displayed automatically.

### Generate relationship between diagram elements

Option 1 – via icon:

1. Select an element in the opened class diagram.
2. Click on a relationship icon, which appears above the element (the selected element acts as source element).
3. Click on the target element of the selected relationship.
  - ⇒ You have created a relationship between the source and target element, which is active in the project and shown in the diagram.

Option 2 – via **Toolbox**:

1. Click on a relationship element from **Toolbox**.
2. In the opened class diagram first click on the source element, then on the target element.
  - ⇒ You have created a relationship between the source and target element, which is active in the project and shown in the diagram.

### Removing an element from the diagram

Option 1 – via icon:

1. Select the element in the class diagram, which you want to remove from the diagram.
2. Use the icon **Remove from diagram**, which appears above the element.
  - ⇒ The selected element is removed from the diagram.

Option 2 – via the [Del] key

1. Select the element in the class diagram, which you want to remove from the diagram.
2. Press the [Del] key.
  - If the [option \[▶ 10\]](#) **Show prompt when deleting objects from diagram** is disabled, the object is only deleted from the diagram by default.
  - If the option is enabled, select the option "Remove from diagram" in the dialog that opens.
  - ⇒ The selected element is removed from the diagram.

### Removing an element from the diagram and the project

Option 1 – via icon:

1. Select the element in the class diagram, which you want to remove from the diagram and the project.
2. Use the icon **Remove from project and diagram**, which appears above the element.
  - ⇒ The selected element is removed from the diagram and the project.

Option 2 – via the [Del] key

1. Enable the [option \[► 10\]](#) **Show prompt when deleting objects from diagram**.
2. Select the element in the class diagram, which you want to remove from the diagram and the project.
3. Press the [Del] key and select the option **Remove from project and diagram** in the dialog that opens.
  - ⇒ The selected element is removed from the diagram and the project.

### As project navigator

1. Open a class diagram and double-click on an element in the class diagram.
  - ⇒ The editor pertaining to the element opens.
2. If necessary, you can edit the declaration and implementation in the opened editor as usual.
  - ⇒ Any changes of the declaration are automatically updated in the class diagram.

### Multiple selections

- If **Pointer** is enabled in **Toolbox** (default), you can drag a rectangle over several elements in the class diagram while pressing the left mouse button. All elements covered by the rectangle are then selected.
- Multiple selections are also possible by successively selecting the required elements while pressing the [Ctrl] key.
- [Ctrl+A] or **Select all** selects all the elements within a rectangle, but no relationship elements.

### Refactoring

The following changes, which are made in the class diagram editor, can be applied simply to the complete project by refactoring:

- Renaming of variables, function blocks or properties
- Addition and removal of variables of the type VAR\_INPUT, VAR\_OUTPUT or VAR\_IN\_OUT

By default, the Refactoring functionality and the associated preview (**Refactoring** dialog) is activated in the class diagram. However, this can be restricted using the following options dialogs:

- Dialog: Options – PLC environment - Refactoring – UML class diagram
- Dialog: [Options – PLC environment – UML – Class diagram \[► 10\]](#)

Note above all that the preview dialog can be skipped in the options for the UML class diagram and that in this case the change will be applied project-wide without requesting confirmation.

## 6.3 Editor

A UML class diagram visualizes the static structure of a project and the elements declarations.



For a special or filtered view of the project, you can drag a selection of objects from the project tree into the class diagram, or selected elements can be removed from the diagram.

The class diagram can be edited through various actions. Further information on the editing options can be found under [Editing a class diagram \[► 22\]](#).

In addition, there are further, element-specific user inputs:

- [Edit class \[► 26\]](#)
- [Edit interface \[► 32\]](#)
- [Edit generalization \[► 49\]](#)
- [Edit realization \[► 47\]](#)
- [Edit association \[► 45\]](#)
- [Edit composition \[► 42\]](#)

## 6.4 Elements

The **Tools** window provides the elements of the class diagram. They can be added to the class diagram window via drag & drop. The elements can be positioned freely.

### Insert element

1. Open **View > Tools** to access the elements.
2. Drag an element into the class diagram window and drop it at a suitable location.

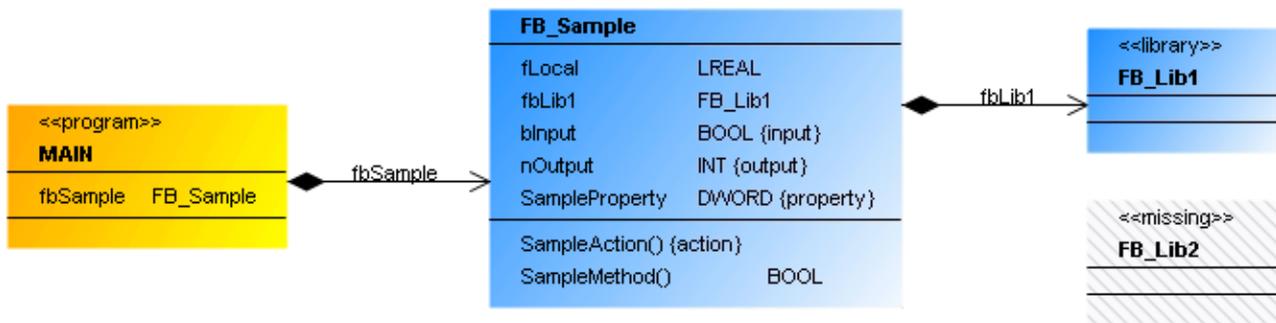
|   |  |
|---|--|
|    | Class [ <a href="#">▶ 25</a> ] (POU)   |
|    | Interface [ <a href="#">▶ 31</a> ]   |
|    | Global variable list [ <a href="#">▶ 34</a> ] (GVL)  |
|    | Data type [ <a href="#">▶ 36</a> ] (DUT)   |
|    | Variable Declaration [ <a href="#">▶ 39</a> ]  |
|    | Property [ <a href="#">▶ 39</a> ]  |
|   | Method [ <a href="#">▶ 40</a> ]  |
|  | Action [ <a href="#">▶ 41</a> ]  |
|  | Composition [ <a href="#">▶ 41</a> ] (VAR)   |
|  | Association [ <a href="#">▶ 44</a> ] (POINTER)<br>or<br>Association [ <a href="#">▶ 44</a> ] (REFERENCE) |
|  | Realization relationship [ <a href="#">▶ 46</a> ]<br>(IMPLEMENTS)  |
|  | Generalization [ <a href="#">▶ 48</a> ] (EXTENDS)  |
|  | Note [ <a href="#">▶ 50</a> ]  |

### 6.4.1 Class

A class is a logical unit in which data and operations are encapsulated. It also represents a variable type that can be instantiated. A class can have an FB\_Init method that is called when an instance is initialized.

A class can have the following relationship types:

- Composition: a class can contain other program elements.
- Association: a class can know other program elements.
- Realization: a class can implement an interface.
- Generalization: a class can inherit from another class.



A class is represented by a three-part rectangle.

- Orange with heading <<program>>: POU (PROGRAM)
- Blue without heading: POU (FUNCTION\_BLOCK or FUNCTION)
- Blue with heading <<library>>: POU (FUNCTION\_BLOCK or FUNCTION) from a library (library function block)
- Greyed with heading <<missing>>: POU (FUNCTION\_BLOCK or FUNCTION) from a library, which is currently not integrated in the project

The class name follows in bold.

All attributes are shown after the first dividing line. Each visible attribute has an identifier. If {input} in curly brackets is appended, it is a variable of type VAR\_INPUT; {output} indicates a variable of type VAR\_OUTPUT. A property has the ID {property}. An internal (non-visible) variable of type VAR has no ID:

<Attribute name> : <Data type> {'{input}' | '{output}' | '{property}'}

The second dividing line is followed by all operations for the class, i.e. its methods or actions. The name of the operation is followed by closing parentheses. This is followed by {action} to identify an action:

<action name>() {action}

If it is a method, the parentheses may indicate a variable transfer. If a return type is declared for a method, this follows in the right-hand column. Unlike actions ({action}), methods have no concluding ID:

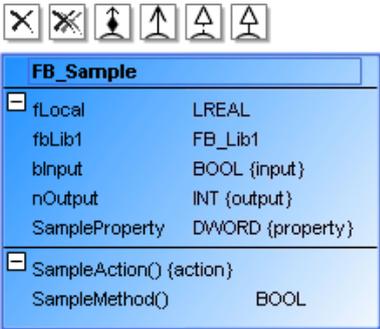
<Method name>(…) : <Return type>

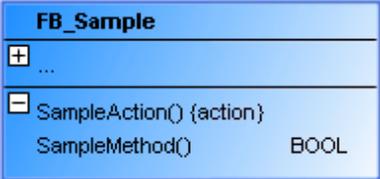
**Properties**

| “Property”   | Description                          |
|--------------|--------------------------------------|
| “Identifier” | Class name<br>Sample: FB_EventLogger |

**User inputs for a class**

The following user inputs are available if “Pointer” is enabled in the toolbox (default).

| User input in the class diagram   | Response in the class diagram   | Description  |
|---|---|--|
| <p>Select the tool "Class (POU)":</p>  <p>Click in an empty area of the diagram. The dialog "Add POU" opens. Enter a name for the new object, adjust the settings and close the dialog with "Add".</p> | <p>A class is created.</p>  | <p>The object exists in the diagram and in the project. The view in the project tree is updated automatically.</p>   |
| <p>Select a class.</p>  |  | <p>The class has expanded attribute and operation lists, which are identified with .</p> <p>Command icons for adding relationship elements are now visible to the left above the class.</p> <p>Tip: If the class has relationships that are currently not shown in the class diagram (missing rectangular elements), a list of "Incoming cross-references" and/or "Outgoing cross-references" appears in the "Toolbox" view. Below this the missing rectangular elements are listed, which you can drag &amp; drop into the class diagram for display.</p> |
| <p>Click on .</p>  | <p>The class is removed in the diagram, so that it is no longer displayed.</p>    | <p>Use "flat" removal of a class to remove it from the class diagram view only. The object still exists and is visible in the project tree.</p> <p>Tip: The class is shown in "Toolbox" under "Incoming cross-references" and/or "Outgoing cross-references", if a rectangle element with a relationship to the remote element is selected in the class diagram.</p>   |
| <p>Click on .</p>  | <p>The class is removed from the diagram and the project.</p>                     | <p>The object no longer appears as object in the class diagram or the project tree. It no longer exists.</p>   |
| <p>Click on  and then in an empty area of the diagram. The dialog "Add POU" opens. Enter a name for the new object, adjust the settings and close the dialog with "Add".</p>                         | <p>A composition arrow points from the existing class to the new class.</p>       | <p>The existing class contains an instance pointing to the new class. Sample:<br/>fbNew : FB_New;</p>  |
| <p>Click on , then on an existing class.</p>   | <p>A composition arrow points from the first class to the second class.</p>       | <p>The first class contains an instance pointing to the second class. Sample:<br/>fbExistent : FB_Existent;</p>  |

| User input in the class diagram  | Response in the class diagram   | Description   |
|--|---|---|
| Click on  and then in an empty area of the diagram. The dialog "Add POU" opens. Enter a name for the new object, adjust the settings and close the dialog with "Add". | A new class with an association arrow pointing to the existing class is created in the diagram.                   | A new class FB_New is created. The existing class now knows the new class and contains an association with the new class. Sample:<br>pNew : POINTER TO FB_New .                       |
| Click on  , then on an existing class.  | An association arrow points from the second class to the first class.   | The first class knows the second class. It now contains a new variable of type. Sample:<br>pExistent : POINTER TO FB_Existent;  |
| Click on  and then in an empty area of the diagram. The dialog "Add POU" opens. Enter a name, adjust the settings and close the dialog with "Add".                    | A generalization points from the existing class to the new class. The existing class inherits from the new class. | The existing class contains the declaration. Sample:<br>FUNCTION_BLOCK FB_Sub EXTENDS FB_New  |
| Click on  , then on an existing class.  | A generalization points from the first class to second class.   | The first class contains the declaration. Sample:<br>FUNCTION_BLOCK FB_Sub EXTENDS FB_Existent  |
| Click on  and then in an empty area of the diagram. The dialog "Add interface" opens. Enter an interface name, adjust the settings and close the dialog with "Add".  | A realization arrow points from the class to the new interface.   | The class implements the new interface. The declaration section of the class contains. Sample:<br>FUNCTION_BLOCK FB_Sample IMPLEMENTS I_SampleNew                                     |
| Click on  , then on an interface.   | A realization arrow points from the class to the interface.   | The class now implements the interface. The declaration section of the class contains. Sample:<br>FUNCTION_BLOCK FB_Sample IMPLEMENTS I_SampleExistent                                |
| Click on  .   |                                | The attribute or operation list is minimized.   |
| Click on  .   |   | The attribute or operation list expands.  |
| Click on the class name. Once it is selected, click on it once again.  | After the first click, the name has a blue border. After the second click the line editor opens.                  | The change is applied synchronously and automatically to the project. That is, the object name in the project tree and in the declaration section of the POU is adjusted immediately. |
| Click twice on an attribute or operation name. Then change the name in the line editor.  | After the first click, the name has a blue border. After the second click the line editor opens.                  | The change is applied synchronously and automatically to the project.   |

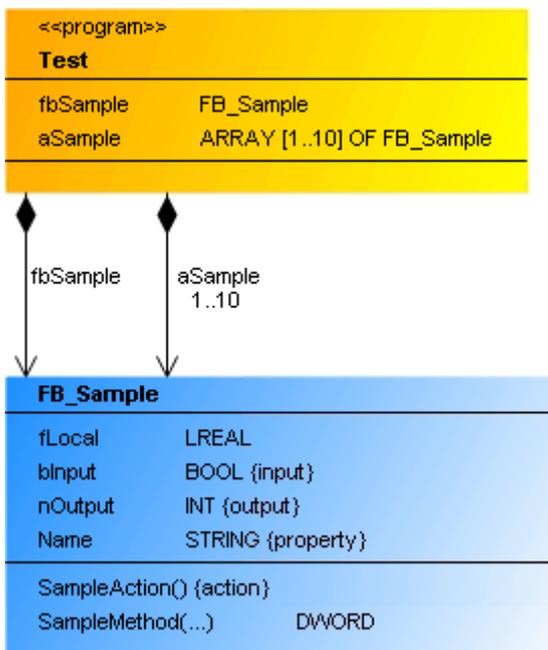
| User input in the class diagram    | Response in the class diagram  | Description  |
|------------------------------------|--|--|
| Double-click on the class element. | The corresponding object editor opens, and the declaration and implementation are displayed. | You can change the declaration or the implementation. After closing of the object you are returned to class diagram. The changes are automatically applied to the class diagram. |



Keep in mind that the default settings in the dialog 'Add POU' or 'Add interface' originate from the last application of this dialog.

**Samples**

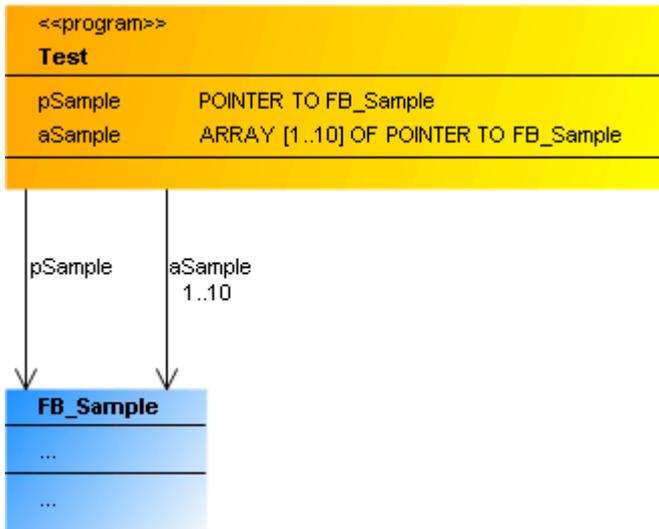
- Composition



```

PROGRAM Test
VAR
fbSample      : FB_Sample;
aSample       : ARRAY[1..10] OF FB_Sample;
END_VAR
    
```

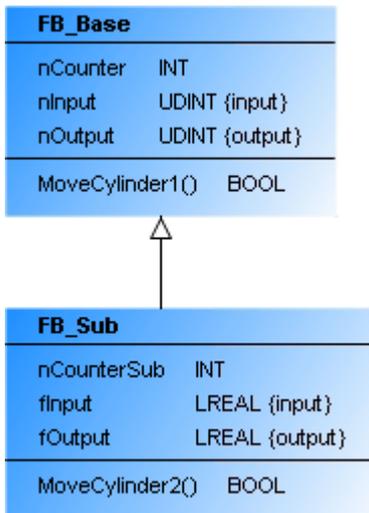
- Association



```

PROGRAM Test
VAR
pSample      : POINTER TO FB_Sample;
aSample      : ARRAY[1..10] OF POINTER TO FB_Sample;
END_VAR
    
```

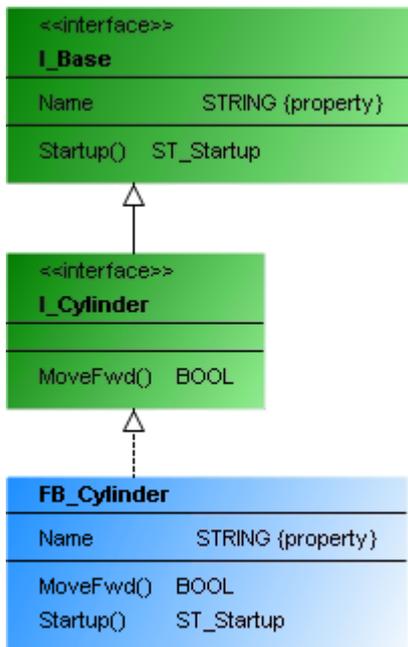
- Generalization



```

FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
    
```

- Implementation



```

INTERFACE I_Cylinder EXTENDS I_Base
FUNCTION_BLOCK FB_Cylinder IMPLEMENTS I_Cylinder
    
```

### 6.4.2 Interface

An interface defines methods and property declarations that describe an externally visible behavior. It contains no variables and no implementation, but only the definition of methods and/or properties. An interface is a variable type that can be instantiated.

An interface can have the following relationship type:

- Generalization: An interface can inherit from another interface.



Like classes, interfaces are shown divided into three parts. The rectangle is green and overwritten with <<interface>>. The interface name follows below in bold. The interface properties are displayed after the first dividing line, based on the following syntax:

<property name>: <data type> {property}

All interface methods follow after the second dividing line. After the method name there may be a reference to the variable transfer in parentheses. If a return type is declared for a method, this follows in the right-hand column.

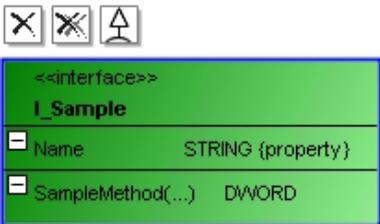
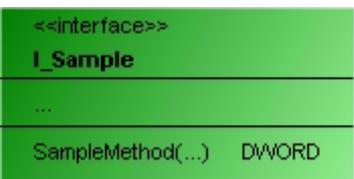
<method name>(…): <return type>

#### Properties

| “Property”   | Description   |
|--------------|---|
| “Identifier” | Insert here an unique name for the selected element. The name can be modified here, however also within the class diagram by selecting the name and then opening an inline editor by a further mouse click. |

**Edit interface**

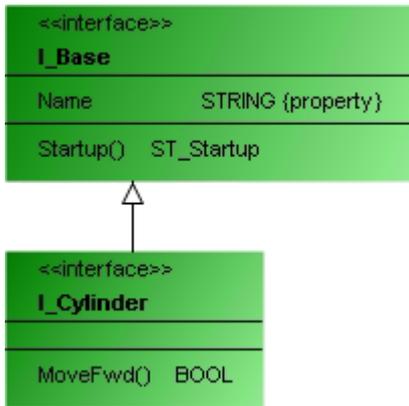
The following user inputs are available if "Selection" is enabled in "Toolbox" (default).

| User input in the class diagram   | Response in the class diagram  | Description   |
|---|--|---|
| <p>Select the tool "Interface":</p>  <p>Click in an empty area of the diagram. The dialog "Add interface" opens. Enter a name for the new object, adjust the settings and close the dialog with "Add".</p> | <p>An interface is created.</p>  | <p>The object exists in the diagram and in the project. The view in the project tree is updated automatically.</p>  |
| <p>Click on an interface icon.</p>  |   | <p>To the left above the interface, icons are now visible, which make it possible to add relationship elements. The example on the left has expanded property and method lists after .</p> |
| <p>Click on .</p>  | <p>The interface is only removed from the diagram.</p>   | <p>Use "flat" removal of an interface to remove it from the class diagram view only. The object still exists and is visible in the project tree.</p>  |
| <p>Click on .</p>   | <p>The interface is removed from the diagram and the project.</p>  | <p>The object is removed. It then no longer exists.</p>   |
| <p>Click on  and then in an empty area of the diagram. The dialog "Add interface" opens. Enter a name and exit the dialog with "Add".</p>  | <p>A generalization points from the existing interface outward to the new interface. The existing interface inherits from the new interface.</p> | <p>The existing interface contains the declaration. Example:<br/> <pre>INTERFACE I_Sample EXTENDS I_New</pre> <p>Keep in mind that the default settings in the dialog 'Add interface' originate from the last application of this dialog.</p> </p>                          |
| <p>Click on , then on an existing interface.</p>   | <p>A generalization points from the second interface to the first interface.</p>   | <p>The first interface contains the declaration. Example:<br/> <pre>INTERFACE I_Sample EXTENDS I_Existing</pre> </p>  |
| <p>Click on .</p>  |   | <p>The property or method list is minimized.</p>  |
| <p>Click on .</p>  |  | <p>The property or method list is expanded.</p>   |
| <p>Click on the name. Once it is selected, click on it once again.</p>  | <p>After the first click, the name has a blue border. After the second click the line editor opens.</p>  | <p>Change the interface name in the line editor. The change is applied synchronously and automatically to the project. That is, the object name in the project tree and in the declaration section of the POU is adjusted immediately.</p>                                  |

| User input in the class diagram | Response in the class diagram                                      | Description  |
|---------------------------------|--|--|
| Double-click on an interface.   | The corresponding object editor opens with the declaration editor. | Edit the declaration. After closing of the object you are returned to class diagram. The changes are automatically applied to the class diagram. |

**Example**

- Generalization



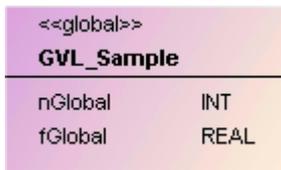
INTERFACE I\_Cylinder EXTENDS I\_Base

**6.4.3 Global Variable List**

A global variable list (GVL) is used to declare global variables. These are available project-wide.

A GVL can have the following relationship types:

- Composition: a class can contain other program elements.
- Association: a class can know other program elements.



A global variable list is represented by a two-part rectangle in **pale pink** and headed with <<global>>. All attributes are shown after the first dividing line:

<attribute name>: <data type>

**Properties**

| “Property”   | Description  |
|--------------|--|
| “Identifier” | Insert or change an unique name of the selected element. |

**Edit GVL**

The following user inputs are available if “Selection” is enabled in the toolbox (default).

| User input in the class diagram   | Response in the class diagram   | Description  |
|---|---|--|
| <p>Select the tool "Global Variable List (GVL)":</p>  <p>Click in an empty area of the diagram. The dialog "Add Global Variables List" opens. Enter a name for the new object and close the dialog with "Add".</p> | <p>A GVL is created.</p>  | <p>The object exists in the diagram and in the project. The view in the project tree is updated automatically.</p>   |
| <p>Click on a GVL object.</p>   |   | <p>Command icons are visible to the left above the object.</p>   |
| <p>Click on .</p>  | <p>GVL is only removed from the diagram.</p>  | <p>Use "flat" removal to remove it from the class diagram view only. The object still exists and is visible in the project tree.</p>   |
| <p>Click on .</p>   | <p>GVL is removed from the diagram and the project.</p>   | <p>The object is removed. It then no longer exists.</p>  |
| <p>Click on , then on an existing class or a DUT.</p>  | <p>A composition points from the GVL to the selected class or DUT.</p>  | <p>The declaration of GVL contains the instantiation based on the selected element. Example:<br/>fbExistent : FB_Existent;</p>   |
| <p>Click on  and then in an empty area of the diagram. Dialog "Add POU" opens. Enter a name and exit the dialog with "Add".</p>  | <p>A composition points from the GVL to the new class.</p>  | <p>The GVL contains the declaration. Example:<br/>fbNew: FB_New;</p>   |
| <p>Click on , then on an existing class or a DUT.</p>  | <p>An association points from the GVL to the selected class or DUT.</p>   | <p>The GVL contains the declaration for the selected element. Example:<br/>pExistent: POINTER TO FB_Existent;</p>  |
| <p>Click on  and then in an empty area of the diagram. The dialog "Add POU" opens. Enter a name and exit the dialog with "Add".</p>  | <p>An association points from the GVL to the new class.</p>   | <p>The GVL contains the declaration for the new class. Example:<br/>pNew: POINTER TO FB_New;</p>   |
| <p>Click on .</p>  |   | <p>The attribute or operation list is minimized.</p>   |
| <p>Click on .</p>  |   | <p>The attribute or operation list expands.</p>  |
| <p>Click on an identifier. Once it is selected, click again.</p>  | <p>After the first click, the name has a blue border. After the second click the line editor opens.</p>   | <p>Change the class name in the line editor. The change is applied synchronously to the project. That is, the object name in the project tree and in the declaration section of the POU is adjusted immediately.</p> |

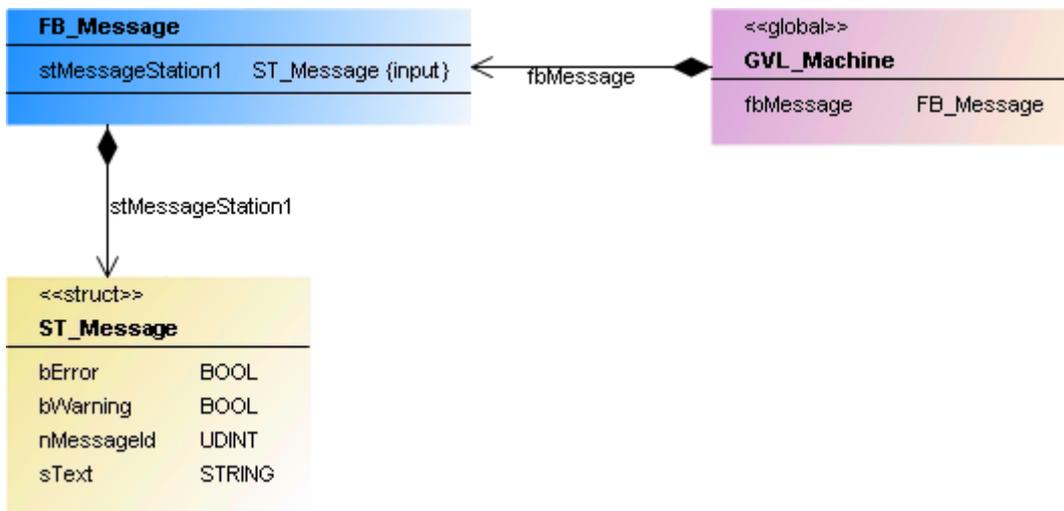
| User input in the class diagram | Response in the class diagram  | Description  |
|---------------------------------|--|--|
| Double-click an element.        | The corresponding object editor opens, and the declaration and implementation are displayed. | Edit the declaration or implementation. After closing of the object you are returned to class diagram. The changes are automatically applied to the class diagram. |



Keep in mind that the default settings in the dialog 'Add POU' originate from the last application of this dialog.

**Sample**

- Composition



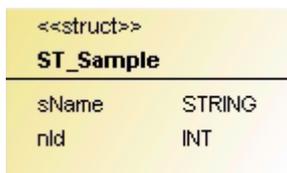
```
VAR_GLOBAL
    fbMessage      : FB_Message;
END_VAR
```

**6.4.4 User-defined data type**

Via a user-defined data type ("Data Unit Type" = DUT) the user can define data types, e.g. in the form of structures or enumerations.

A user-defined structure data type can have the following relationship type:

- Generalization: a structure can inherit from another structure.



A user-defined data type is represented by a two-part rectangle in **pale yellow** and overwritten with `<<struct>>`, if it is a structure or a union. `<<enum>>` indicates an enumeration type. The identifier follows in bold. All attributes are shown after the first dividing line:

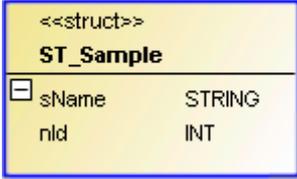
<attribute name>: <data type>

**Properties**

| "Property"   | Description  |
|--------------|--|
| "Identifier" | Insert or change an unique name of the selected element. |

**Edit DUT**

The following user inputs are available if "Selection" is enabled in the toolbox (default).

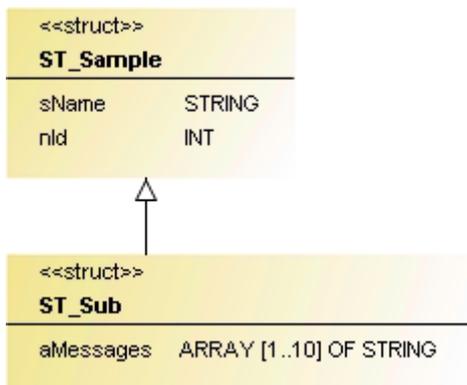
| User input in the class diagram  | Response in the class diagram   | Description   |
|--|---|---|
| <p>Select the tool "Data Unit Type (DUT)":</p>  <p>Click in an empty area of the diagram. The dialog "Add DUT" opens. Enter a name for the new object, adjust the settings and close the dialog with "Add".</p> | <p>A data type object is created.</p>   | <p>The object exists in the diagram and in the project. The view in the project tree is updated automatically.</p>  |
| <p>Click on a DUT.</p>   |   | <p>Icons are visible to the left above.</p>   |
| <p>Click on .</p>   | <p>DUT is only removed from the diagram.</p>  | <p>Use "flat" removal to remove it from the class diagram view only. The object still exists and is visible in the project tree.</p>                                      |
| <p>Click on .</p>  | <p>DUT is removed from the diagram and the project.</p>   | <p>The object is removed. It then no longer exists.</p>   |
| <p>Click on  and then in an empty area of the diagram. The dialog "Add DUT" opens. Enter a name for the parent object and exit the dialog with "Add".</p>   | <p>A generalization points from the existing DUT to the new DUT. The existing DUT inherits from the new one.</p>  | <p>The existing DUT contains the declaration. Example:<br/>TYPE ST_Sample EXTENDS ST_New</p>  |
| <p>Click on , then on an existing DUT.</p>  | <p>A generalization arrow points from the second DUT to the first DUT.</p>  | <p>The first DUT inherits from the second DUT. The first DUT contains the declaration. Example:<br/>TYPE ST_Sample EXTENDS ST_Existing</p>                                |
| <p>Click on .</p>   |   | <p>The attribute or operation list is minimized.</p>  |
| <p>Click on .</p>   |   | <p>The attribute or operation list expands.</p>   |
| <p>Double-click on the identifier. Then change the name in the line editor.</p>  | <p>After the first click, the name has a blue border. After the second click the line editor opens.</p>   | <p>The change is applied synchronously and automatically to the project.</p>  |
| <p>Double-click the element.</p>   | <p>The corresponding object editor opens.</p>   | <p>Edit the declaration or implementation. After closing of the editor you are returned to class diagram. The changes are automatically applied to the class diagram.</p> |



The default settings in the dialog "Add DUT" originate from the last application of this dialog.

**Sample**

- Generalization



```

TYPE ST_Sub EXTENDS ST_Sample :
STRUCT
aMessages : ARRAY[1..10] OF STRING;
END_STRUCT
END_TYPE
  
```

**6.4.5 Variable declaration**

A "Variable declaration" is used for adding a variable to a class (program, function block, function) or data structure (DUT) in the class diagram.

**Add Variable**

In the "Toolbox" click on "Variable declaration": 

Then select an element in the diagram for extending by a variable. If you try to select a non-reasonable position, the cursor will look like a prohibition sign. At reasonable position it appears as a blue cross. If you click at such a position, then the dialog "Auto declare" opens for adding a variable to the focused element. Insert the required settings as usual. After having closed the dialog, the focused element gets extended by the new variable. An appropriate update will be done synchronously in the class diagram and in the object editor.

Look of the cursor

-  : At this position, no adding is possible.
-  : At this position, you can add a variable to the focused element.

**6.4.6 Property**

The "Property" element is used to add a property to a class (program or function block) or an interface.

**Add property**

Select the "Property" tool: 

Then you can select the element in the diagram that should be extended by a property. At points that are not meaningful, the cursor has the form of a prohibition sign. At meaningful points the cursor has the form of a blue or black cross. If you click on such a point, a further property is added to the element in focus. The "Add property" dialog opens. Enter the required settings as usual. Closing the dialog expands the program element with this property, which is immediately updated in the class diagram and in the project tree.

Cursor shape:

-  : at these points in the diagram expansion is not possible.
-  : at these points in the diagram you can add a property to the element in focus.

### Access modifier



Available from TC3.1 Build 4026

The access modifier of a method or a property is displayed in the class diagram by means of a symbol. The following table shows which symbol stands for which access modifier.

| Symbol | Access modifier |
|--------|-----------------|
| +      | PUBLIC          |
| #      | PROTECTED       |
| -      | PRIVATE         |
| ~      | INTERNAL        |

## 6.4.7 Method

The element "Method" is used to extend a class (program or function block) or an interface with a method.

### Add method

Select the tool "Method": 

In the diagram you can then select the element that is to be extended with a method. At points that are not meaningful, the cursor has the form of a prohibition sign. At meaningful points the cursor has the form of a blue or black cross. If you click on such a point, a further method is added to the element in focus. The dialog "Add method" opens. Enter the required settings as usual. Closing the dialog expands the program element with this method, which is immediately updated in the class diagram and in the project tree.

Cursor

-  : at these points in the diagram expansion is not possible.
-  : at these points in the diagram you can add a method to the element in focus.

### Access modifier



Available from TC3.1 Build 4026

The access modifier of a method or a property is displayed in the class diagram by means of a symbol. The following table shows which symbol stands for which access modifier.

| Symbol | Access modifier |
|--------|-----------------|
| +      | PUBLIC          |
| #      | PROTECTED       |
| -      | PRIVATE         |
| ~      | INTERNAL        |

### 6.4.8 Action

The "Action" element is used to extend a program or a function block by an action.

#### Add Action

Select the "Action" tool: 

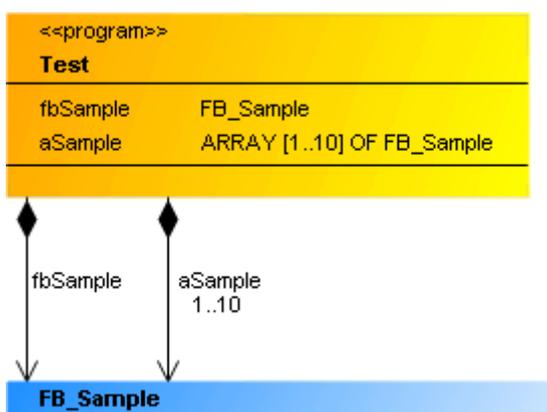
Subsequently, the element to be extended by an action can be selected in the diagram. At points that are not meaningful, the cursor has the form of a prohibition sign. At meaningful points the cursor has the form of a blue or black cross. If you click such a point, a further action will be added to the focused element. The "Add action" dialog opens. Enter the required settings as usual. Closing the dialog expands the program element with this action, which is immediately updated in the class diagram and in the project tree.

Cursor

-  : at these points in the diagram expansion is not possible.
-  : at these points in the diagram you can extend the focused element by an action.

### 6.4.9 Composition

A composition is a UML relationship that expresses a "contained" property: an element contains another element. In IEC code, this corresponds to the instantiation of an element: fbSample: FB\_Sample. The cardinality indicates how often the relationship exists. In IEC code this corresponds to an ARRAY [...]. If a cardinality greater than 1 is specified, the declaration is as follow: aSample: ARRAY[1..10] OF FB\_Sample.



A composition is represented as an arrow with a solid black diamond, pointing from a class or a global variable list to a class of type FUNCTION\_BLOCK or a DUT.

**Property**

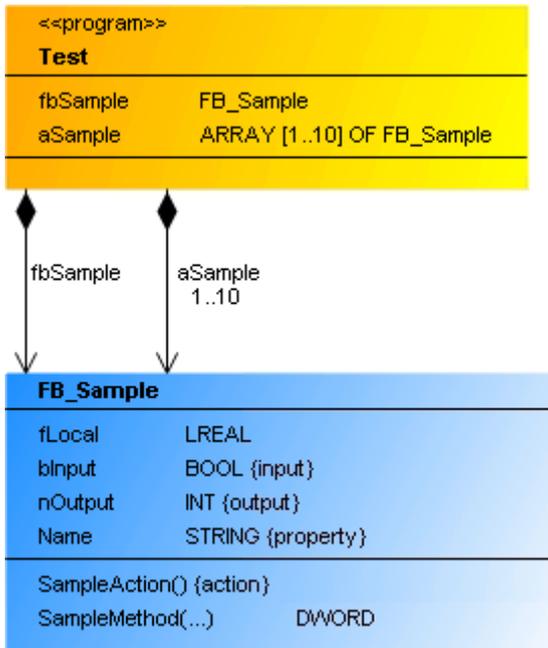
| “Property”       | Description   |
|------------------|---|
| “Relationship”   | Composition (not editable)  |
| “Optimize route” | If the option is enabled, the route of the relationship arrow is optimized automatically. The starting point at the start element and the end point at the target element are fixed. If, for sample, the target element is moved, the point to which the arrow points on the target element is retained. If the option is disabled, the route is retained. This option is deselected when a relationship element is positioned manually in the class diagram.<br>Activate the option if automatic optimization is required. |
| “Start element”  | The name of the element at which the relationship element starts is displayed.  |
| “Target element” | The name of the element to which the relationship element points is displayed.  |
| “Identifier”     | The name of the relationship element is displayed.  |

**Edit composition**

| User input in the class diagram  | Response in the class diagram   | Description   |
|--|---|---|
| Select the tool "Composition":<br><br>Select a class or a GVL, then click on the element to include.  | A composition between the elements is drawn.                              | The IEC code is automatically adapted by extending the declaration section of the existing element. Example:<br>fbExistent: FB_Existent;                          |
| Select the tool "Composition":<br><br>Select a class or a GVL, then click in an empty area of the diagram. The dialog "Add POU" opens. Enter a name, adjust the settings and close the dialog with "Add". | A composition pointing from the class or GVL to the new class is created. | The IEC code is automatically adapted by extending the declaration section of the existing element. Example:<br>fbNew: FB_New;                                    |
| Select the tool "Pointer".<br>Click on a composition and move the line with the mouse.   |   | The selected (and therefore blue) composition runs at the new position. The property "Optimize routing" is automatically disabled.                                |
| Select the tool "Pointer".<br>Click on a composition and use the [Del] key or click on "Delete" in the context menu.   |   | The association is removed from diagram and the IEC code. The instantiation of the class or the data type is removed from the declaration section of the element. |

**Samples**

- Composition of a class



- Single composition

```

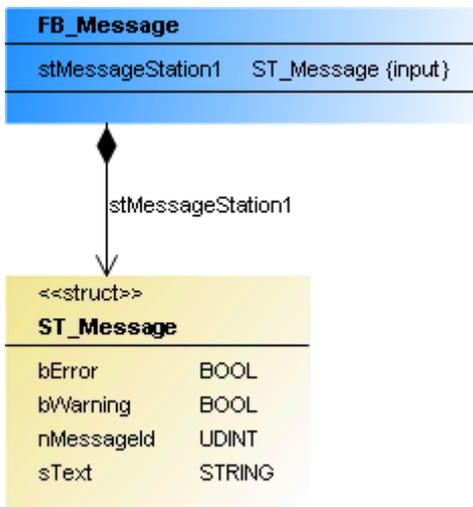
PROGRAM Test
VAR
fbSample      : FB_Sample;
END_VAR
    
```

- Multiple composition

```

PROGRAM Test
VAR
aSample      : ARRAY[1..10] OF FB_Sample;
END_VAR
    
```

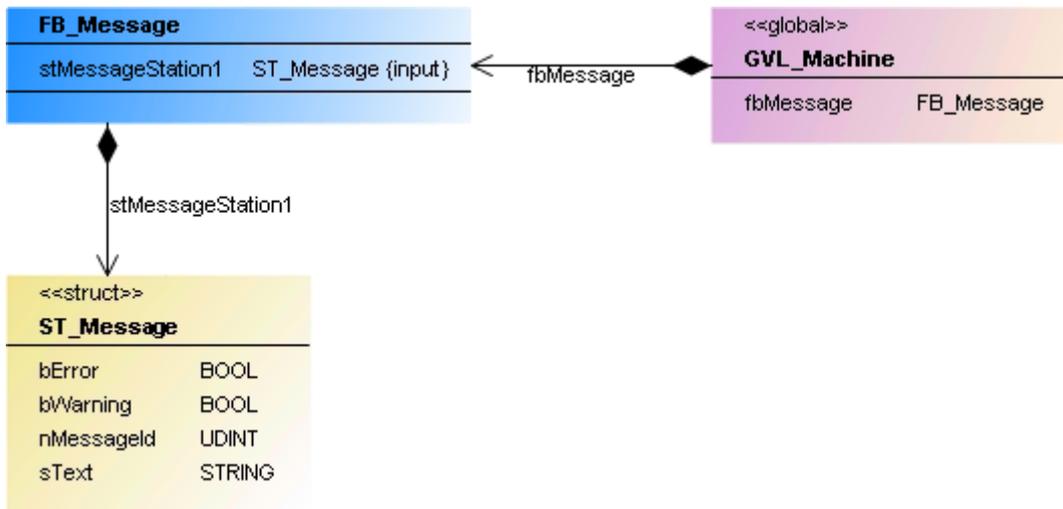
- Composition of a data object



```

FUNCTION_BLOCK FB_Message
VAR_INPUT
stMessageStation1 : ST_Message;
END_VAR
    
```

- Composition in GVL



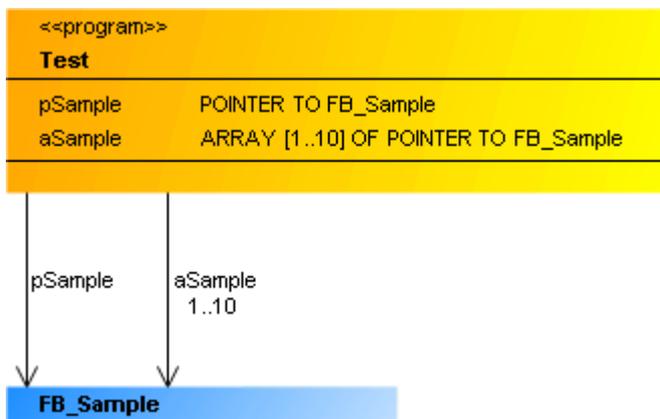
```

VAR_GLOBAL
    fbMessage      : FB_Message;
END_VAR
    
```

### 6.4.10 Association

An association is a UML relationship that expresses "knowing". The knowing element points as a pointer or reference to another element. In IEC code this corresponds to a POINTER TO instruction (pSample : POINTER TO FB\_Sample) or a REFERENCE TO instruction (refSample : REFERENCE TO FB\_Sample).

In the case of pointers, the cardinality specifies how often the relationship exists. In IEC code this corresponds to an ARRAY [...]. If a cardinality greater than 1 is specified, the declaration is as follow: aSample : ARRAY[1..10] OF POINTER TO FB\_Sample.



An association is represented as an arrow pointing from a class or a global variable list to a class of type FUNCTION\_BLOCK or DUT.

**Property**

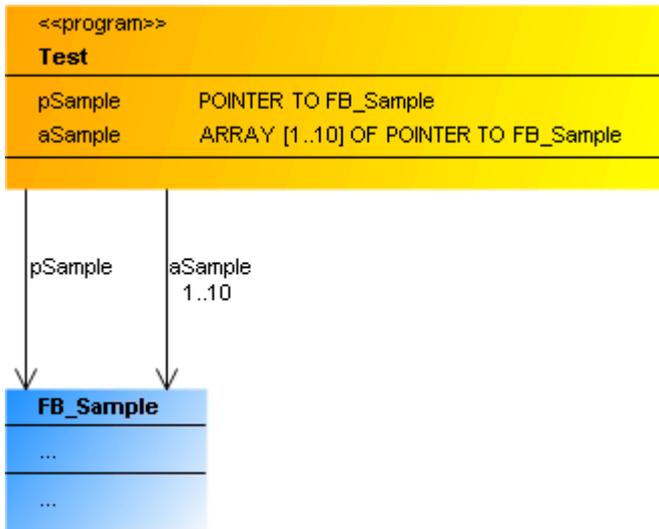
| “Property”       | Description   |
|------------------|---|
| “Relationship”   | Association (cannot be edited)  |
| “Optimize route” | If the option is enabled, the route of the relationship arrow is optimized automatically. The starting point at the start element and the end point at the target element are fixed. If, for sample, the target element is moved, the point to which the arrow points on the target element is retained. If the option is disabled, the route is retained. This option is deselected when a relationship element is positioned manually in the class diagram.<br>Activate the option if automatic optimization is required. |
| “Start element”  | The name of the element at which the relationship element starts is displayed.  |
| “Target element” | The name of the element to which the relationship element points is displayed.  |
| “Identifier”     | The name of the relationship element is displayed.  |

**Edit association**

| User input in the class diagram   | Response in the class diagram  | Description   |
|---|--|---|
| Select the tool "Association":<br><br>Select a class or a GVL, then click on the object to be used for the association.  | An association between the elements is drawn.                              | The IEC code is automatically adapted by extending the declaration part of the existing element. Example:<br><br>pExistent : POINTER TO FB_Existent;      |
| Select the tool "Association":<br><br>Select a class or a GVL, then click in an empty area of the diagram.<br>The dialog "Add POU" opens. Enter a name, adjust the settings and close the dialog with "Add". | An association pointing from the class or GVL to the new class is created. | The IEC code is automatically adapted by extending the declaration part of the existing element. Example:<br><br>pNew : POINTER TO FB_New;                |
| Select the tool "Pointer".<br>Click on an association and move the line with the mouse.   |  | The selected (and therefore blue) association runs at the new position. The property "Optimize routing" is automatically disabled.                        |
| Select the tool "Pointer".<br>Click on an association and use the [Del] key or click on "Delete" in the context menu.   |  | The association is removed from diagram and the IEC code. The POINTER TO or REFERENCE TO instruction is removed from the declaration part of the element. |

**Samples**

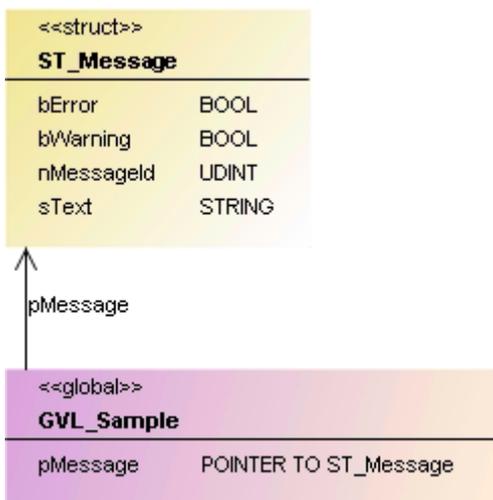
- Association with a class (of a program) – single and multiple



```

PROGRAM Test
VAR
pSample      : POINTER TO FB_Sample;
aSample      : ARRAY[1..10] OF POINTER TO FB_Sample;
END_VAR
    
```

- Association with a data structure (of a GVL) – single

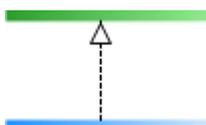


```

VAR_GLOBAL
pMessage      : POINTER TO ST_Message;
END_VAR
    
```

### 6.4.11 Implementation

A realization is a UML relationship that expresses an interface implementation. The realizing or implementing class object (function block) implements the properties and methods of the interface. In IEC coding this relationship corresponds to the keyword IMPLEMENTS.



A realization is indicated by a dashed arrow, pointing from a class of type FUNCTION\_BLOCK to an interface.

Property

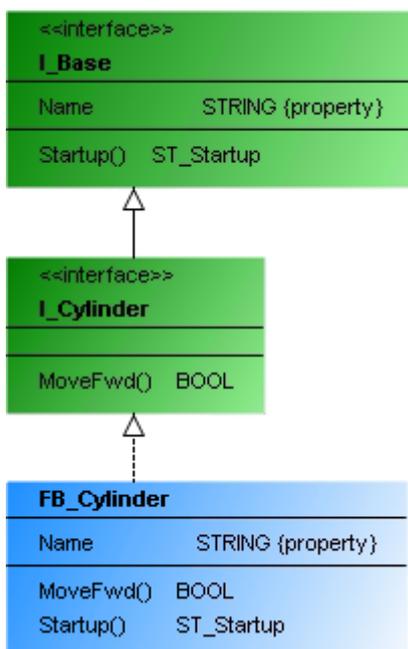
| “Property”     | Description                |
|----------------|----------------------------|
| “Relationship” | Realization (not editable) |

Edit realization

| User input in the class diagram   | Response in the class diagram   | Description  |
|---|---|--|
| Select the tool "Realization":<br><br>Select a class and then an interface.  | A realization is created, pointing from the class to the interface.     | The IEC code is automatically adapted to indicate the interface in the declaration section of the class. Example:<br><br>FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Existent                            |
| Select the tool "Realization":<br><br>Select a class, then click in an empty area of the diagram. The dialog “Add interface” opens. Enter a name, adjust the settings and close the dialog with "Add". | A realization is created, pointing from the class to the new interface. | The IEC code is automatically adapted by creating a new interface and indicating this interface in the declaration section of the class. Example:<br><br>FUNCTION_BLOCK FB_Sample IMPLEMENTS I_New |
| Select the tool "Pointer".<br>Click on a realization and move the line with the mouse.  |   | The selected (and therefore blue) realization runs at the new position.  |
| Select the tool "Pointer".<br>Click on a realization and use the [Del] key or click on "Delete" in the context menu.  |   | The realization is removed from diagram and the IEC code. The IMPLEMENTS statement is removed from the declaration section of the class.   |

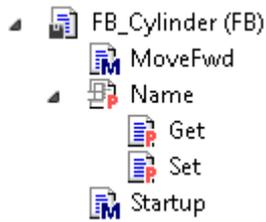
Example

- Realization



```

INTERFACE I_Cylinder EXTENDS I_Base
FUNCTION_BLOCK FB_Cylinder IMPLEMENTS I_Cylinder
    
```



## 6.4.12 Generalization

A generalization is a UML relationship that expresses an inheritance or specialization. The inheriting element has the attributes and operations of the element from which it inherits. In IEC coding this relationship corresponds to the keyword EXTENDS.



A generalization is indicated by an arrow with a solid tip, pointing from the inheriting class to the base class from which it inherits. The direction of the arrow indicates which component inherits from which.

Inheritance is possible between classes, interfaces and user-defined data types:

- A function block can inherit from another function block.
- An interface can inherit from another interface.
- A DUT can inherit from another DUT.
- Programs and functions cannot inherit or pass on.

### Property

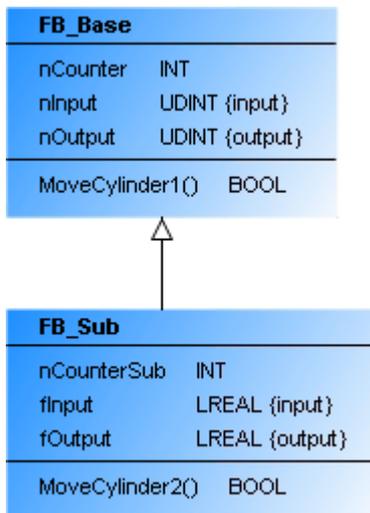
| “Property”     | Description                   |
|----------------|-------------------------------|
| “Relationship” | Generalization (not editable) |

**Edit generalization**

| User input in the class diagram   | Response in the class diagram  | Description   |
|---|--|---|
| Select the tool "Generalization":<br><br>Select the object that is to inherit, then click on the parent object.  | A generalization is created, pointing from the inheriting element to the parent element.     | The IEC code is automatically adapted to indicate the parent element in the declaration section of the inheriting class.<br>Example:<br>FUNCTION_BLOCK FB_Sample<br>EXTENDS FB_BaseExistent                     |
| Select the tool "Generalization":<br><br>Select the object that is to inherit, and then click on an empty area in the diagram.<br>A dialog for creating a new object opens. Enter a name, adjust the settings and close the dialog with "Add". | A generalization is created, pointing from the inheriting element to the new parent element. | The IEC code is automatically adapted by creating a new object and indicating the parent object in the declaration section of the inheriting object. Example:<br>FUNCTION_BLOCK FB_Sample<br>EXTENDS FB_BaseNew |
| Select the tool "Pointer".<br>Click on a generalization and move the line with the mouse.   |  | The selected (and therefore blue) generalization runs at the new position.  |
| Select the tool "Pointer".<br>Click on a realization and use the [Del] key or click on "Delete" in the context menu.  |  | The generalization is removed from the diagram and the IEC code. The EXTENDS statement is removed from the declaration section of the inheriting object   |

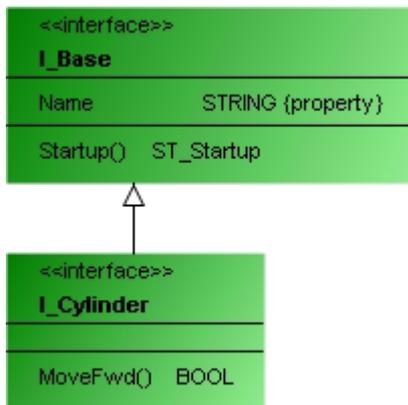
**Samples**

- Function block



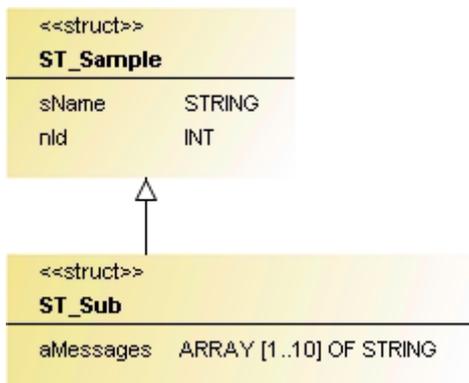
```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
```

- Interface



```
INTERFACE I_Cylinder EXTENDS I_Base
```

- DUT



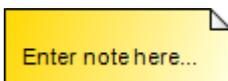
```

TYPE ST_Sub EXTENDS ST_Sample :
STRUCT
aMessages : ARRAY[1..10] OF STRING;
END_STRUCT
END_TYPE
  
```

## 6.4.13 Note

You can insert a comment in the editor of a class diagram or statechart with the "Note" element.

At present only one-line notes are possible.



### Adding a note

Select the "Note" tool: 

In the editor, click the desired insertion position. Then double-click the text in the element and replace it with the desired text.

Cursor

-  : The cursor has the form of a prohibition sign at points where insertion is not allowed.
-  : The cursor is a blue cross at points where it is allowed.

## 7 UML state diagram

In addition to the following information, please refer to the [samples \[► 90\]](#), which provide an introduction to the tool and to the call behavior of the UML state diagram.

### 7.1 Basic principles

The UML state diagram is a graphical formalism to specify and design the sequence of a system with discrete events or to program a time behavior. Integrated in the PLC area of the TwinCAT 3 development environment, a state diagram has an editor with state and transition elements. When the application is compiled, executable code is generated. By default, the switching behavior of a state diagram is clocked depending on the task cycle when it is executed, but depending on how the states are configured, it can also switch independently. In this case, these are cycle-internal states. Powerful functionalities such as syntax monitoring or debugging in online mode support the development process.

#### Application case

A Statechart is used for programming time-discrete systems. Programs, function blocks, functions, methods or actions can be created as a graph from states and transitions with the **Statechart** implementation language.

Functions/methods and their data are normally temporary. If you implement a function/method with the UML Statechart implementation language, the chart contains cycle-internal states and re-initializes the data at the start of the task cycle.

With methods, conversely, there is a possibility to declare the method as VAR\_INST. The data are then retained as with a function block and are not temporary, therefore the Statechart will be run through in several task cycles as usual. Refer also to [Option UseVarInst \[► 84\]](#) regarding this. This option is not available with functions.

#### Syntax

The syntax is under compiler control:

- a transition between states lying in different regions are not allowed
- orthogonal states cannot be nested in other states
- exactly one exception transition or completion transition is allowed to exit a composite state
- a conditional transition going into an orthogonal state is not allowed, then a fork/join is required

#### Commands

The following commands or action options are available for the Statechart:

- [Creating a new Statechart \[► 54\]](#)
- [Editing Statechart \[► 55\]](#)
- [Go to Definition \[► 58\]](#)

Also note the commands, which are available for all UML diagrams: [Common commands for all UML diagrams. \[► 15\]](#)

#### Implicit variables of a state diagram

The implicit variables are located in the program unit that maps the state diagram (e.g. in the instance of the function block or in a program). The internal variables are located inside the program unit under the element name "UML\_SC\_<POU name>" with the data type "UML\_SC\_<id>".

For the function block "FB\_UML", the element name of the implicit variables is "UML\_SC\_FB\_UML", for example. They are located in the function block instance "fbUml".

| Expression    | Type   |
|---------------|--|
| fbUml         | FB_UML   |
| UML_SC_FB_UML | _UML_SC_9cf66aa46976417893eebd57c6deeeeb       |
| InFinalState  | BOOL   |
| ReInit        | BOOL   |
| Abort         | BOOL   |
| AutoReInit    | BOOL   |
| States        | ARRAY [1..4] OF _UML_SC_State                  |
| Names         | _UML_SC_9cf66aa46976417893eebd57c6deeeeb_Names |

#### "UML\_SC\_<POU name>" of the data type \_UML\_SC\_<id>:

Some implicit variables reflect the state of the object during the runtime (*InFinalState*, *States*), while others serve to control the behavior at runtime (*Reinit*, *Abort*, *AutoReInit*).

| Name         | Data type                                     | Meaning   |
|--------------|---|---|
| InFinalState | BOOL  | This variable has the value TRUE if the state diagram is in the end state.  |
| ReInit       | BOOL  | Setting this variable to TRUE will cause the state diagram to be re-initialized, i.e. the start state of the diagram is activated.                          |
| Abort        | BOOL  | Setting this variable to TRUE will cause the current operation of the diagram to be aborted and the end state of the diagram to be activated.               |
| AutoReInit   | BOOL  | If the value of this variable is TRUE, the start state of the diagram is automatically restored as soon as the end state is reached.<br>Default value: TRUE |
| States       | ARRAY[<number of states>]<br>OF _UML_SC_State | see below   |
| Names        | _UML_SC_<id>_Names                            | see below   |

#### "UML\_SC\_<POU name>.States" of the data type ARRAY[<number of states>] OF \_UML\_SC\_State:

The variables of the structure \_UML\_SC\_State serve to describe a state. This data type is the base type of an array with the name "States", which is declared in the implicit variables and describes the individual states of the state diagram.

The individual variables of the structure \_UML\_SC\_State are described in the following table.

| Name               | Data type | Meaning   |
|--------------------|-----------|---|
| Active             | BOOL      | Flag for determining whether the state is currently active.   |
| FastExecutionFault | BOOL      | Relevant for IntraCycle states: if an IntraCycle state is active for longer than a cycle, this flag is set by the system. The flag is reset on exiting from the IntraCycle state. |
| ID                 | INT       | ID of the state<br>The ID of the state corresponds to the index at which the state is described in the "States" array.  |
| ActivationTime     | TIME      | Time stamp of the last activation of the state  |
| Name               | STRING    | Name of the state   |

**"UML\_SC\_<POU name>.Names" of the data type \_UML\_SC\_<id>\_Names:**

The data type \_UML\_SC\_<id>\_Names contains an INT variable for each state of the state diagram. This variable is declared with the name of the state and its value indicates the ID of the state. The ID of the state corresponds to the index at which the state is described in the "States" array.

| Name   | Data type | Meaning  |
|--|-----------|--|
| <Name of the state><br><b>Example:</b> StartState1 | INT       | ID of the state (= index at which the state is described in the "States" array)<br><b>Example:</b> ID of the state "StartState1", e.g. 4 |
| <Name of the state><br><b>Example:</b> State1      | INT       | ID of the state (= index at which the state is described in the "States" array)<br><b>Example:</b> ID of the state "State1", e.g. 5      |
| <Name of the state><br><b>Example:</b> State2      | INT       | ID of the state (= index at which the state is described in the "States" array)<br><b>Example:</b> ID of the state "State2", e.g. 6      |
| ...  | ...       | ...  |

**Example:**

A function block is created with the implementation language UML SC and the name "FB\_UML". The statechart diagram has a state named "MyStateName". The FB also contains the "MyMethod" method in the ST language. In the following sample, it is queried from inside and outside the FB whether the statechart diagram is currently in the "MyStateName" state.

Access from inside the FB / in FB\_UML.MyMethod:

```
FUNCTION_BLOCK FB_UML
VAR
    bInSpecificState : BOOL;
    ...
END_VAR

METHOD MyMethod : BOOL
bInSpecificState := UML_SC_FB_UML.States[UML_SC_FB_UML.Names.MyStateName].Active;
```

Access from outside the FB / in MAIN:

```
PROGRAM MAIN
VAR
    fbInstance : FB_UML;
    bInSpecificState : BOOL;
END_VAR
```

```
fbInstance();
fbInstance.MyMethod();

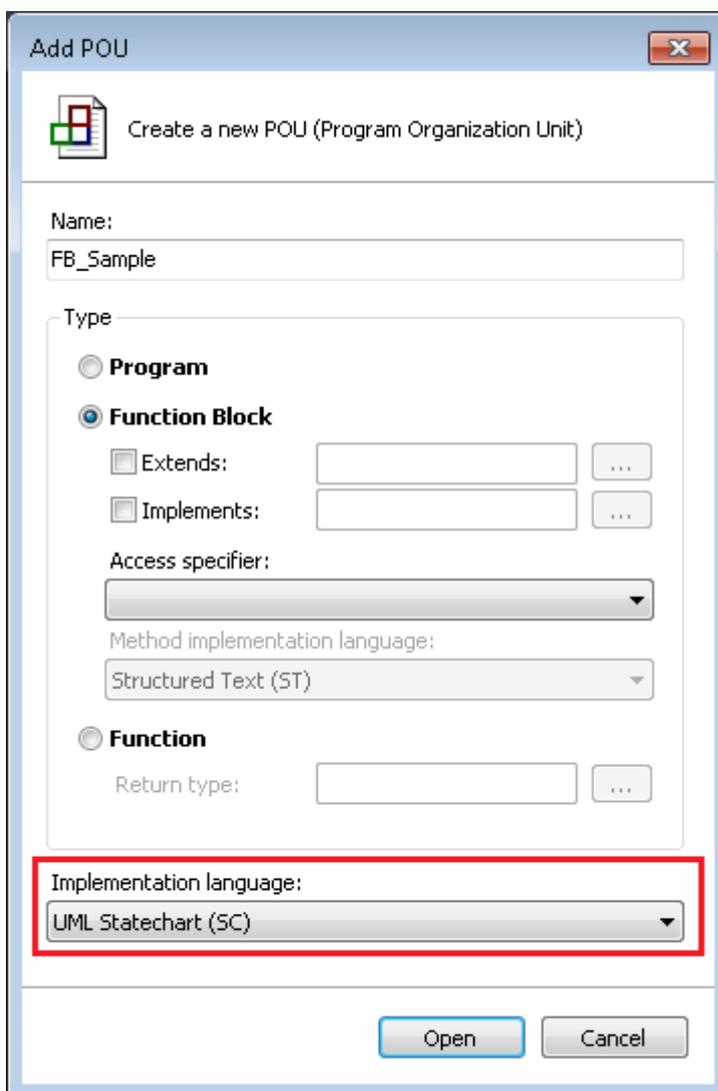
bInSpecificState := fbInstance.UML_SC_FB_UML.States[fbInstance.UML_SC_FB_UML.Names.MyStateName].Active;
```

## 7.2 Commands

### 7.2.1 Creating a new Statechart

Programs, function blocks, functions, methods or actions can be created as a graph from states and transitions with the Statechart implementation language.

1. In the context menu of the project tree select the command for adding the required program element (e.g. function block, method).
2. Configure the opening dialog as usual (for a POU element, e.g. name and type) and select **UML Statechart (SC)** as the implementation language.



3. Confirm the inputs and configurations with the **Open** button.
  - ⇒ The new statechart object is added in the project tree, and the editor for the new diagrams opens.

## 7.2.2 Editing Statechart

The actions available for editing a Statechart include the following. Further editing options can be found under [e \[▶ 58\]ditor](#).

### Adding a new element

1. Open the **Toolbox** window via the **View** menu.
  2. Select an [element \[▶ 59\]](#) in the **Toolbox** view and drag & drop it onto the opened Statechart. Drop it in a suitable location to visualize it there.
- ⇒ The new element is shown in the diagram.

### Deselect “Toolbox” view

- ✓ An element is selected in the **Toolbox** window. In the editor the cursor has the form of the selected element.
1. Press the right mouse button.
- ⇒ The element is deselected, and the standard **Pointer** element is selected.

### Editing identifiers

1. Open the line editor of an identifier with two single clicks.
2. Enter a new name and confirm it via the [Enter] key or by clicking in an empty area of the diagram.



⇒ The identifier has the new name.

### Adding a new action object

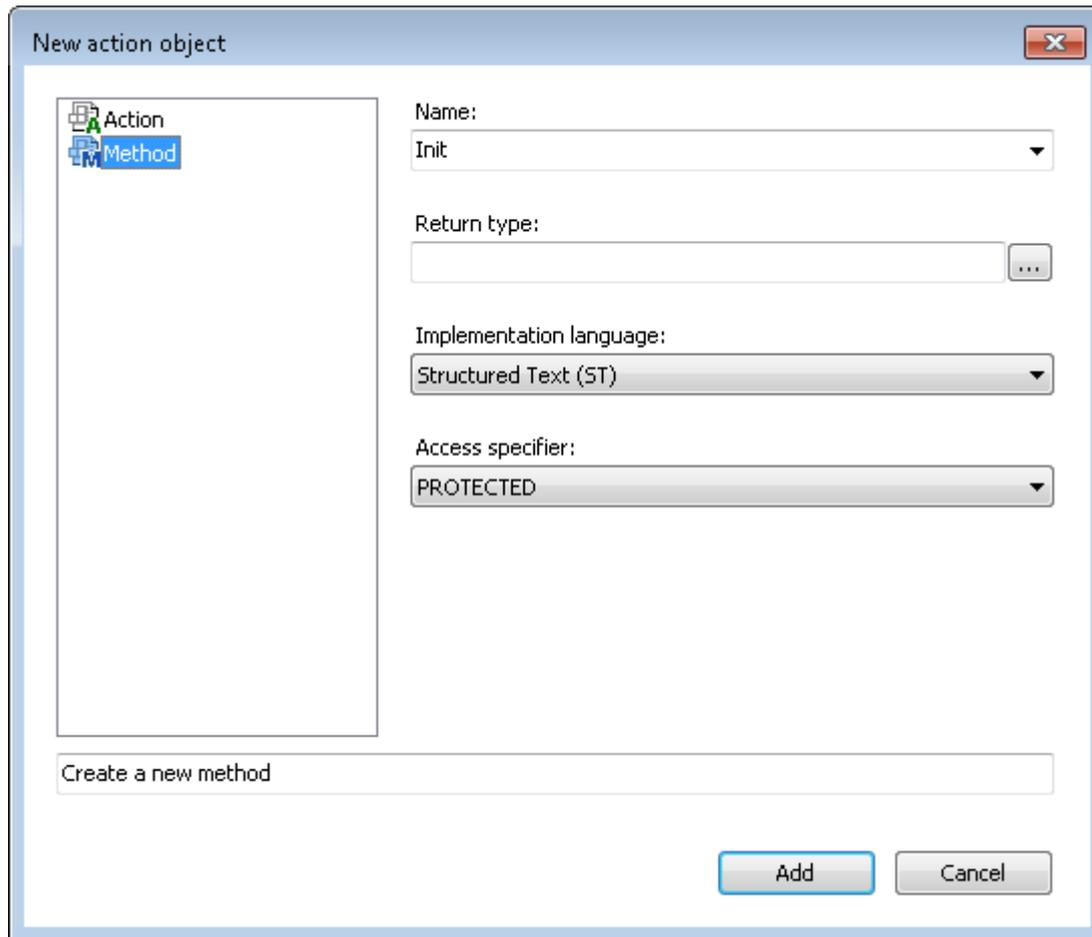
1. Click on the symbol **New <ENTRY/DO/EXIT> action/method**, which appears at the end of an action line for a state and at the start of an action line for a composite state.
- Requirement: So far the action line has no declaration or implementation.



⇒ The dialog **New action object** opens.

2. Specify the object type in the left part of the dialog by selecting **Action** or **Method**.
3. If the object type **Action** is selected: Configure the name and the implementation language for the new action.  
If the object type **Method** is selected: Configure the name, the return type, the implementation language

and the access modifier for the new method.



4. Confirm the inputs and configurations with the **Add** button.

⇒ The new action or method object is added to the project tree and displayed as linked object in the edited state. In addition, the editor for the new action or the new method opens.



### Creating a new transition

Transitions can be created based on various source elements: start state, state, composite state, choice, fork/join.

The possible target elements, for which an incoming transition is created, depend on the source element type and the transition type (transition, completion transition, exception transition).

The general procedure for generating a transition is explained below.

Option 1 – via icon:

1. Select an element in the opened Statechart.
2. Click on a transition icon, which appears above the element (the selected element acts as source element).
3. Click on an existing element to configure it as target element for the selected transition. Alternatively, click in a blank area of the editor to generate a new state.

- Configure the action and the guard condition for the transition as required by opening the respective line editors with two single clicks. If the guard condition depends on more than one variable, you can insert a new line via [Ctrl+Enter] during the configuration of the transition condition, in order to show the whole transition more clearly.
- ⇒ You have created a transition from source to target element, which determines the switching characteristics of the Statechart.

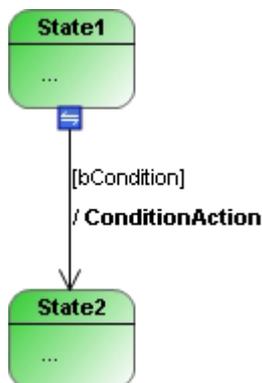
Option 2 – via **Toolbox**:

- Click on a transition element from **Toolbox**.
  - In the open Statechart first click on the source element, then on an existing element in order to configure it as target element for the selected transition. Alternatively, click in a blank area of the editor to generate a new state.
  - Configure the action and the guard condition for the transition as required by opening the respective line editors with two single clicks. If the guard condition depends on more than one variable, you can insert a new line via [Ctrl+Enter] during the configuration of the transition condition, in order to show the whole transition more clearly.
- ⇒ You have created a transition from source to target element, which determines the switching characteristics of the Statechart.

### Changing the end points of an existing transition

The end points of an existing transition can be linked to other source or target elements using the Reconnect functionality. As a result, existing transition configurations (action, guard condition) are retained, while the transition connections change.

- Click on the start or the end of a transition, depending on whether you want to change the source element or the target element.
- ⇒ The reconnect symbol appears at the transition connection.



- Drag & drop the symbol onto another source or target element to connect the transition there.
- ⇒ The transition was linked to another source/target element, without changing the other transition configurations.

### As project navigator

- Open a Statechart and double-click on an action object of a (composite) state.
- ⇒ The editor for the action object opens.
- If required, you can change the declaration and implementation of the action object in the open editor.
- ⇒ Use the project navigator functionality to view or adjust the behavior of the selected action object.

### Multiple selections

- If **Pointer** is enabled in **Toolbox** (default), you can drag a rectangle over several elements in the Statechart while pressing the left mouse button. All elements covered by the rectangle are then selected.

- Multiple selections are also possible by successively selecting the required elements while pressing the [Ctrl] key.
- Use [Ctrl+A] or **Select all** to select all elements of the diagram.

### 7.2.3 Go to definition

For transitions, the command **Go To Definition** is available in the corresponding context menu.

The command is available in offline and online mode.

1. Right-click on a transition to open the corresponding context menu.
  2. Select the command **Go to Definition**.
- ⇒ The editor, in which the first variable of the transition or the transition expression is declared, opens and the variable is marked in the declaration editor.

### 7.2.4 Find All References

For transitions, the command **Find All References** is available in the corresponding context menu.

The command is available in offline and online mode.

1. Right-click on a transition to open the corresponding context menu.
  2. Select the command **Find All References**.
- ⇒ The **Cross Reference List** view opens and shows the locations where the first variables of the transition or transition expression are used.

### 7.2.5 Add Watch

For transitions, the command **Add Watch** is available in the corresponding context menu.

The command is only available in online mode and provides a further debugging option – in addition to the integrated online view available with the transition – for transferring the variables of a transition to the watch list via a command. This command is particularly useful if a transition expression consists of a combination of several variables (e.g. "bCondition1 AND bCondition2"). If you execute the command **Add Watch** for this transition, both variables of the transition expression are added to the watch list, i.e. *bCondition1* and *bCondition2*. This gives you a simple overview of which sub-signal of the overall expression does not yet have the required value for switching to the next state.

1. Right-click on a transition to open the corresponding context menu. The transition expression can consist of one or more variables.
  2. Select the **Add Watch** command.
- ⇒ The transition expression variable(s) is/are inserted in the currently open watch list. If no watch list is currently open, the command inserts the variable(s) into Watch List 1 and opens its view.

## 7.3 Editor

The Statechart editor is used for designing and programming procedures based on a Statechart. The **state** is the main element of the Statechart. A Statechart is usually clocked via the task cycle, although cycle-internal state machines are also possible (see configuration of a [state](#) [► 60]).

The Statechart can be edited through various actions. The editing options are documented [here](#) [► 55].

In addition, there are further, element-specific user inputs:

- [Edit state](#) [► 62]
- [Edit transition](#) [► 79]
- [Edit completion transition](#) [► 80]
- [Edit exception transition](#) [► 83]

- [Edit start state \[▶ 60\]](#)
- [Edit choice \[▶ 77\]](#)
- [Edit composite state \[▶ 67\]](#)
- [Edit fork/join \[▶ 75\]](#)

## 7.4 Elements

The **Tools** window shows the elements of the statechart. They can be added to the State diagram window via drag & drop. States and pseudo states can be positioned as required.

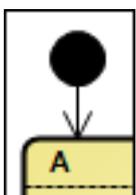
### Insert element

1. Open **View > Tools** to access the elements.
2. Drag an element into the State diagram window and drop it at a suitable location.

|   |  |
|---|--|
|    | <a href="#">Start state [▶ 59]</a>           |
|    | <a href="#">End state [▶ 60]</a>             |
|    | <a href="#">State [▶ 60]</a>                 |
|   | <a href="#">Composite state [▶ 64]</a>       |
|  | <a href="#">Fork/Join [▶ 74]</a>             |
|  | <a href="#">Choice [▶ 76]</a>                |
|  | <a href="#">Transition [▶ 77]</a>            |
|  | <a href="#">Completion transition [▶ 79]</a> |
|  | <a href="#">Exception transition [▶ 80]</a>  |
|  | <a href="#">Note [▶ 83]</a>                  |

### 7.4.1 Start State

A start state is a pseudo state that indicates the initial state. If it is at top-level, the sequence of the state chart starts there. If it is in a composite states or in region of an orthogonal state, the sequence of the region starts.



The start state is displayed as a black, filled circle.

**Properties**

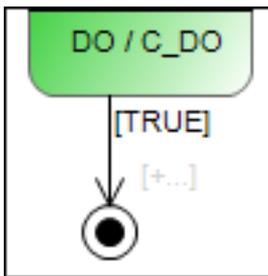
| "Property"   | Description  |
|--------------|--|
| "Identifier" | Here you can enter a name. It is not displayed in the state diagram. |

**Edit start state**

| User input in the Statechart  | Response in the State-chart  | Description  |
|---|--|--|
| Focus on a start state.   | <br> | The start state is editable.<br><ul style="list-style-type: none"> <li>A completion transition can be added via the command icon that is displayed above the start state.</li> </ul> |
| Click on the symbol  |  | A completion transition is added. If you click on an existing state, it becomes the target state of the transition. Click in an empty area to create a new state.                    |

**7.4.2 End State**

An **End state** is a pseudo state that indicates that the execution of the region containing this end state has been completed. End states can be located in statecharts at top level, in composite states, or in regions of orthogonal states.



An end state is displayed as a little black, filled circle within a second empty circle.

**Properties**

| "Property"   | Description  |
|--------------|--|
| "Identifier" | Here you can enter a name. It is not displayed in the state diagram. |

**7.4.3 State**

A **state** is the main element of a state diagram. A state machine (or state diagram) passes through various states and executes their actions during its runtime. A state can have ENTRY, DO or EXIT actions, which are executed at specified times during the runtime of the state.



A state is displayed as a rectangle with rounded corners.

**Normal state**

A normal state is clocked by the task in which it is called. This means that the **transition** into the next state is not called before the next task cycle switches.

**Cycle-internal state**

If a state machine consists of cycle-internal states, the switching characteristics are independent of the task cycle. This means that completion of the actions of an internal state is followed immediately by the transition. The transition condition is checked immediately, and the transition action is executed when the condition is met. Also, the system immediately switches to the target state when the transition condition is met.

Whether a state switches cycle-internally is set in the property "Internal state".

**ENTRY, DO, EXIT actions/methods**

A state can have an ENTRY, DO and/or EXIT action or method. For these actions or methods you can choose any implementation language.

- The ENTRY action can initialize the state. It is executed once when all incoming transitions switch, so that the state becomes active.
- The DO action is executed as long as the state is active.
- The EXIT action is intended to ensure that the state is exited in a valid state. The EXIT action is executed once when all outgoing transitions switch, so that the state is exited.

**Call behavior**

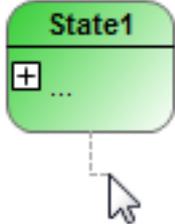
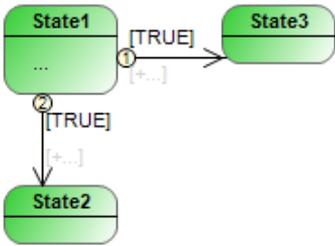
Make sure you refer to the [samples \[▶ 90\]](#), which describe and illustrate the call behavior of the UML state diagram.

**Properties**

| "Property"             | Description  |
|------------------------|--|
| "Identifier"           | Name of the state<br>Example: Enable, Production, ErrorHandler   |
| "Color"                | Color of the state<br>Click on the set color to change the color of the state via the drop-down menu that opens.<br><br>Default:  50, 205, 50   |
| "Implicit start state" | <ul style="list-style-type: none"> <li>• No: [Default] set the property to No if you require a normal state.</li> <li>• Diagram: set the property to diagram if you want that the state to be also a start state. A state machine requires a start state at the top level and also each region in a composite state.</li> </ul>  |
| "Cycle-internal"       | <ul style="list-style-type: none"> <li>• <input checked="" type="checkbox"/> : tick the checkbox to set the state cycle to internal. In "Cycle-internal" the transition is immediately switched to the next state.</li> <li>• <input type="checkbox"/> : [Default] untick the checkbox to set the state cycle to normal. The state then changes with the task cycle. A transition to the next state is executed with a cycle change of the task.</li> </ul>  |
| "Max. DO cycle calls"  | Requirement: this property is available if the property "Cycle-internal" is activated.<br>Define the maximum number of calls of the DO action per cycle. Enter a number between 1 and 32767. This setting is relevant for the cycle-internal states if the outgoing transition of the cycle-internal state is not fulfilled during the processing of the cycle-internal state and will also not be fulfilled during the DO calls. Setting the maximum number of calls ensures that calling the cycle-internal state does not lead to an infinite loop. |
| "ENTRY action"         | Assign an action to the selected state by specifying its action name.  |
| "DO action"            |  |
| "EXIT action"          |  |

**Edit state**

The following user inputs are available, provided "Pointer" or "Choice" is enabled in "Toolbox" (standard mode).

| User input in the state diagram  | Response in the state diagram   | Description   |
|--|---|---|
| Focus on a state.  |    | <p>The state is editable.</p> <ul style="list-style-type: none"> <li>You can edit the name through two single clicks on the name.</li> <li>You can add outgoing transitions via the command icon above the state.</li> <li>Via the three icons in the center of the state you can extend the state with an ENTRY, DO and/or EXIT action.</li> <li>The state can be removed with the [Del] key.</li> </ul> |
| Click on the green symbol   |    | <p>An outgoing transition is created. If you click on an existing state, it becomes the target state of the transition. Click in an empty area to create a new state.</p>   |
| Generate several transitions on the same state.  |   | <p>If a state has more than one outgoing/incoming transition, the priorities of the transitions define their execution order. The priority of the transitions is shown in a small circle.</p>   |
| Click on  :   |  | <p>The state is extended by an ENTRY action/method.</p>   |
| Click on  :   |  | <p>The state is extended by a DO action/method.</p>   |
| Click on  :   |  | <p>The state is extended by an EXIT action/method.</p>  |
| Focus on a state, which was extended by an action, and click on the symbol  . |  | <p>For action lines the symbol appears at the end of the line, if no action has been assigned yet.</p> <p>The dialog "New action object" opens for generating a new action. The name of the action is shown after a slash.</p> <p>See also: "Add new action object" <a href="#">▶ 55</a></p>  |
| Click twice (two single clicks) on the ENTRY, DO or EXIT expression (irrespective of whether an action was already assigned or not).                             |   | <p>The line editor opens with IntelliSense support, so that a new action can be assigned. If the required action is selected in IntelliSense, the action can be selected by double-clicking or single-clicking plus [Enter].</p>  |

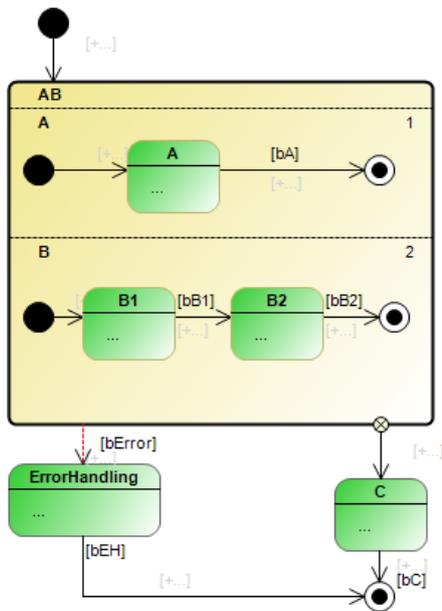
| User input in the state diagram   | Response in the state diagram   | Description   |
|---|---|---|
| Focus on a state that was assigned an action. Click on the symbol  or double-click on the assigned action. |  | The assigned action, "SampleMethod" in the sample, opens in the editor. The action object can be edited in the POU editor that opens. |
| Click on  :  |  | The action list for the state collapses.  |
| Click on  :  |  | The action list for the state expands.  |

## 7.4.4 Composite State

A **composite state** is used for grouping the states it covers. It can be used for two different use cases:

- Grouping/Nesting:
  - If a composite state consists of precisely one region, the inner states are passed through sequentially at runtime. An inner state can thereby be a composite state again, so that a nesting of composite states is created. In addition, such composite states can call ENTRY/DO/EXIT actions.
  - Composite states, each with a particular region, are used to group states, so that shared forwarding or shared error handling can be implemented, e.g. with the aid of (pseudo) exception transitions. In addition or alternatively, the composite state can be provided with its own ENTRY/DO/EXIT actions, which can be topically assigned to the inner states. This is interesting, for example, if a higher-level DO action is to be called in addition to the DO actions of the inner states. In this case the higher-level DO action could be placed in the composite state.
  - If multiple composite states are nested, only the innermost composite state may have more than one region.
- Parallel sub-state machines:
  - If a composite state consists of several regions, the inner states are grouped orthogonally. The regions of the composite state are assigned priorities, which control the processing order at runtime. The states of the regions are passed through pseudo-parallel, according to their internal sequencing.
  - Composite states with several regions or so-called orthogonal states are used for programming parallel states.

Samples of the implementation of various use cases can be found at the foot of this page.



A composite state is shown as a yellow, filled rectangle with rounded corners. Its name is displayed in the upper left corner of the rectangle. Regions are separated by black dashed lines.

In the case of a composite state with several regions, the priority of each region is displayed in the top right-hand corner and the name of each region in top left-hand corner of the region. The lines, names and priorities are editable.

**Syntax rules**

**General syntax rules for composite states**

- A composite state can have one or several outgoing exception transitions. An error handling, for example, can be implemented with the aid of an exception transition.
- Completion transition
  - A composite state can only have one outgoing completion transition.
  - If a composite state has an outgoing completion transition, each region must contain a start and end state.
  - The regions do not require an end state if the composite state has no outgoing completion transition.
  - However, the composite state must have an outgoing completion transition if all regions of the composite state contain an end state.
  - The composite state reaches its end state when all regions have reached their respective end states.

**Syntax rules for a composite state with precisely one region**

- Activate start state/composite state
  - If the composite state has an outgoing completion transition, the region must have a start and end state (refer also to the general syntax rules regarding the end state) and the incoming transition must be connected to the composite state to activate it.
  - If the composite state does not have an outgoing completion transition, the region can optionally contain a start state.  
 If a start state exists, to activate the composite state, the incoming transition must be connected to the composite state. A direct connection via a conditional transition between a state inside the composite state and a state outside the composite state is not allowed - neither from outside to inside nor from inside to outside.  
 On the other hand, if there is no start state, the incoming transitions are directly connected to the inner states. This could be one or more incoming transitions. Furthermore, the other direction is also possible in this case: starting from the states inside the composite state, outgoing conditional

transitions can be connected directly to states outside of the composite state. This could be one or more outgoing transitions. Please note that states inside and outside of a composite state may only be directly connected to one another (regardless of the direction) if the inner and outer state are separated only by one level of a composite state. This means that a state located inside a composite state that is in turn located inside another composite state (nesting) cannot be directly connected to a state located outside this nesting of composite states. This connection would cross two borders of composite states, but it is only allowed to cross one border. The named state can only be directly connected with an outside state if this target state is located in the outer composite state (only one level crossed).

- Nesting
  - A composite state with precisely one region can contain another composite state. In this case the states are nested.
  - The nesting of composite states may be arbitrarily deep, with only the innermost composite state having more than one region.

### Syntax rules for a composite state with multiple regions

- Activate start state/composite state
  - The incoming transition must be linked with the composite state to activate it.
  - Each region must contain a start state.
- Transitions between states in different regions are not permitted.
- A direct connection via a conditional transition between a state inside the composite state and a state outside the composite state is not allowed - neither from outside to inside nor from inside to outside. Exception: using a fork that is outside the composite state, you can create transitions that go to states inside the composite state.

### ENTRY, DO, EXIT actions/methods

A composite state that has precisely one region can be assigned ENTRY/DO/EXIT actions/methods.

In order to enable this function, the option "Allow ENTRY/DO/EXIT actions" must be activated. This option is located in the properties of the composite state. For these actions or methods you can choose any implementation language.

- The ENTRY action can initialize the composite state. It is executed once when all incoming transitions switch or when an internal state becomes active, so that the composite state is activated.
- Note the description of the property "Execute DO actions, even if inner composite states are active", which determines the behavior of the DO action.
- The EXIT action is intended to ensure that the composite state is exited in a valid state. The EXIT action is executed once when all outgoing transitions switch or when an outer state becomes active, so that the composite state is exited.

### Call behavior

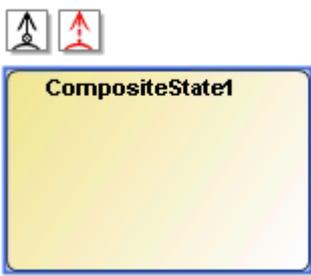
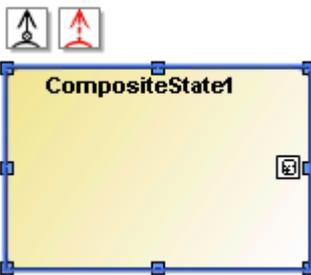
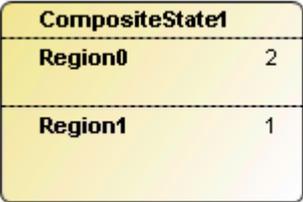
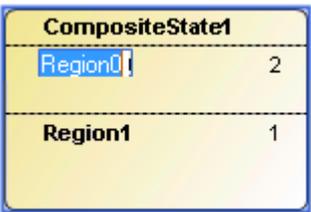
Make sure you refer to the [samples \[► 90\]](#), which describe and illustrate the call behavior of the UML state diagram.

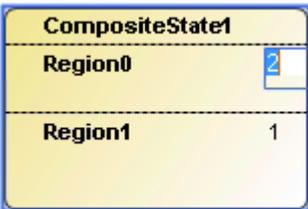
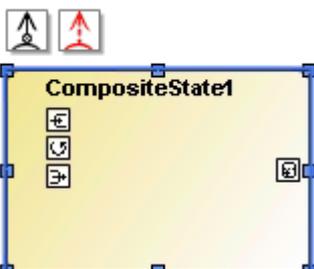
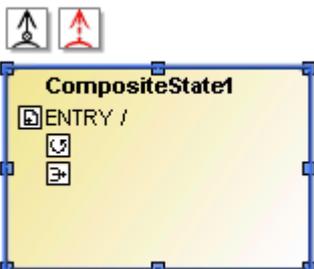
**Properties**

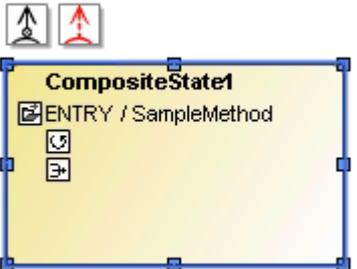
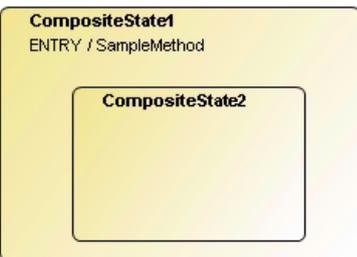
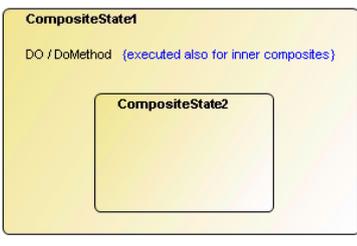
| "Property"  | Description   |
|---|---|
| "Identifier"  | Name of the composite state<br>Example: DoorAutomation  |
| "Color"   | Color of the composite state<br>Click on the set color to change the color of the composite state via the drop-down menu that opens.<br><br>Default:  240, 230, 140  |
| "Allow ENTRY/DO/EXIT actions"                                   | Requirement: this property is available when the selected composite state has exactly one region. In this case you can also assign actions to a composite state if it is part of a nesting of composite states.<br><br><input checked="" type="checkbox"/> : you can assign an ENTRY, DO or EXIT action to the selected composite state.<br><br><input type="checkbox"/> : you cannot assign ENTRY, DO or EXIT actions to the selected composite state.   |
| "Execute DO actions, even if inner composite states are active" | Requirement: several composite states are graphically nested within each other. The option is only available for the outer composite state and is passed on to the inner composite states.<br><br><input checked="" type="checkbox"/> : at runtime the DO action of the outer composite state is executed continuously, even if an inner composite state is active. In the editor the note "{is also executed for inner composite states}" appears next to the DO action, in order to highlight this action behavior.<br><br><input type="checkbox"/> : when one of the inner composite states becomes active, the DO action of the outer composite state pauses. |
| "ENTRY action"  | Requirement: the selected composite state has precisely one region and the property "Allow ENTRY/DO/EXIT actions" is activated.   |
| "DO action"   |   |
| "EXIT action"   |   |

**Edit composite state**

The user inputs in the state diagram editor can be summarized as follows:

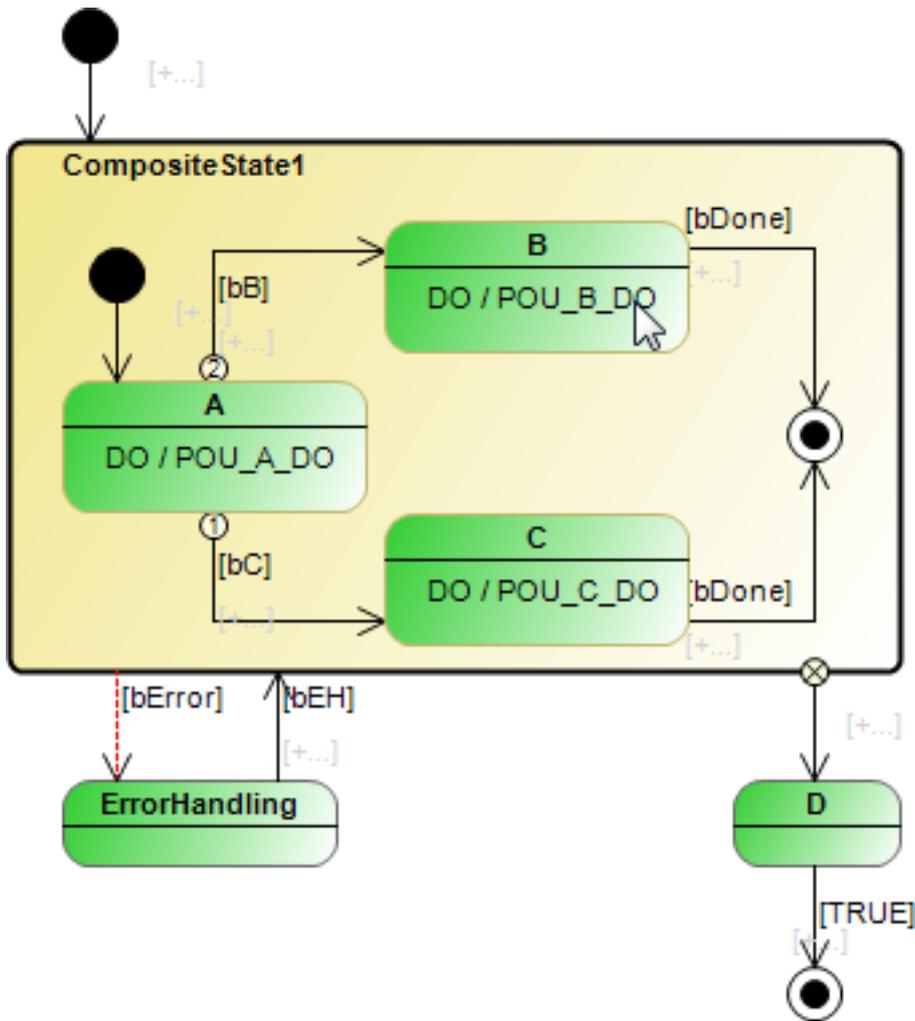
| User input in the state diagram editor  | Response in the state diagram   | Description   |
|---|---|---|
| Focus on a composite state.   |    | <p>The composite state is editable.</p> <ul style="list-style-type: none"> <li>You can edit the name through two single clicks on the name.</li> <li>You can add outgoing transitions via the command icons that are shown above the state.</li> <li>The size of the composite state is adjustable.</li> <li>The state can be removed with the [Del] key.</li> </ul> <p>Conditional editing options:</p> <ul style="list-style-type: none"> <li>The composite state can be extended with an ENTRY, DO and/or EXIT action, if the composite state has a region and the property "Allow ENTRY / DO / EXIT actions" is enabled.</li> </ul> |
| Click on the green symbol<br>                        |   | A completion transition is added. If you click on an existing state, it becomes the target state of the transition. Click in an empty area to create a new state.   |
| Click on the green symbol<br>                       |   | An exception transition is added. If you click on an existing state, it becomes the target state of the transition. Click in an empty area to create a new state.   |
| Focus on a composite state and hold the mouse pointer over the state.   |  |   |
| Pull one of the blue  squares to another position. |   | The size of the composite state was adjusted.   |
| Click on the green symbol                          |  | The state is subdivided, and a further region is added. The name and priority are displayed for each region.  |
| Click twice (two single clicks) on the name of a region.  |  | The name of the region is editable.   |

| User input in the state diagram editor   | Response in the state diagram   | Description  |
|--|---|--|
| Click twice (two single clicks) on the priority of a region.   |    | The priority of the region is editable. Enter a number to define the priority of the region. The priorities of the other regions are automatically adjusted.   |
| Click on the dividing line and move it.  |   | The dividing line is moved to adjust the size of the regions.  |
| Click on the dividing line and press the [Del] key.  |   | The dividing line is removed, so that the regions that were separated by it become one region.   |
| Focus on a composite state and hold the mouse pointer over the state.  |    | Requirements: <ul style="list-style-type: none"> <li>• The composite state has one region.</li> <li>• The property "Allow ENTRY / DO / EXIT actions" is enabled</li> </ul>   |
| Click on one of the three symbols<br>   |   | These command icons extend a composite state with an ENTRY, DO or EXIT action.<br>Clicking on one of the rectangles brings up the following: <ul style="list-style-type: none"> <li>•  <b>ENTRY /</b> : extension by ENTRY action</li> <li>•  <b>DO /</b> : extension by DO action</li> <li>•  <b>EXIT /</b> : extension by EXIT action</li> </ul> |
| Focus on a state, which was extended by an action, and click on the symbol  . |  | For action lines the symbol appears at the end of the line, if no action has been assigned yet.<br>The dialog "New action object" opens for generating a new action. The name of the action is shown after a slash.<br>See also: <a href="#">"Add new action object" [► 55]</a>  |
| Click twice (two single clicks) on the ENTRY, DO or EXIT expression (irrespective of whether an action was already assigned or not).                             |   | The line editor opens with IntelliSense support, so that a new action can be assigned. If the required action is selected in IntelliSense, the action can be selected by double-clicking or single-clicking plus [Enter].  |

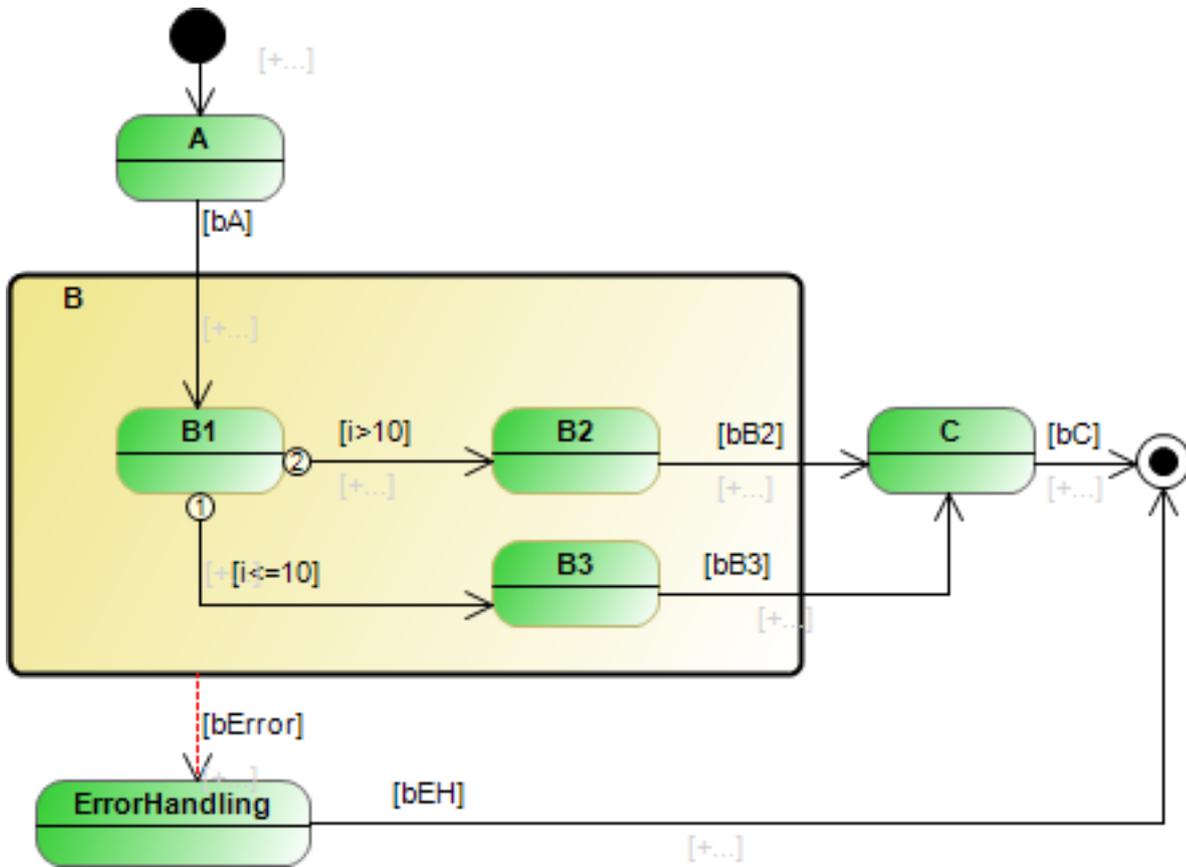
| User input in the state diagram editor   | Response in the state diagram   | Description  |
|--|---|--|
| <p>Focus on a state that was assigned an action. Click on the symbol  or double-click on the assigned action.</p> |    | <p>The assigned action, "SampleMethod" in the sample, opens in the editor. The action object can be edited in the POU editor that opens.</p>   |
| <p>Pull a composite state from the Toolbox view to a composite state with precisely one region.</p>  |    | <p>The composite states are nested. If the property "Allow ENTRY / DO / EXIT actions" is enabled for the outermost state, you can assign each composite state its own actions (ENTRY/ DO / EXIT).</p>  |
| <p>Add a DO action to the outermost composite state and activate the option "Execute DO actions, even if inner composite states are active" under Properties.</p>                                  |   | <p>At runtime the DO action, e.g. "DoMethod", is executed continuously, even if one of the inner composite states is active. With deeper nesting, this option and its value are passed on to inner states. The note "{is also executed for inner composite states}" appears in the editor.</p> |
| <p>Pull a state from the "Toolbox" window to a region of the composite state.</p>  |  | <p>The state is assigned to this region. If the  symbol appears, the insert position to which the mouse points is not allowed.</p>  |

**Samples of composite states**

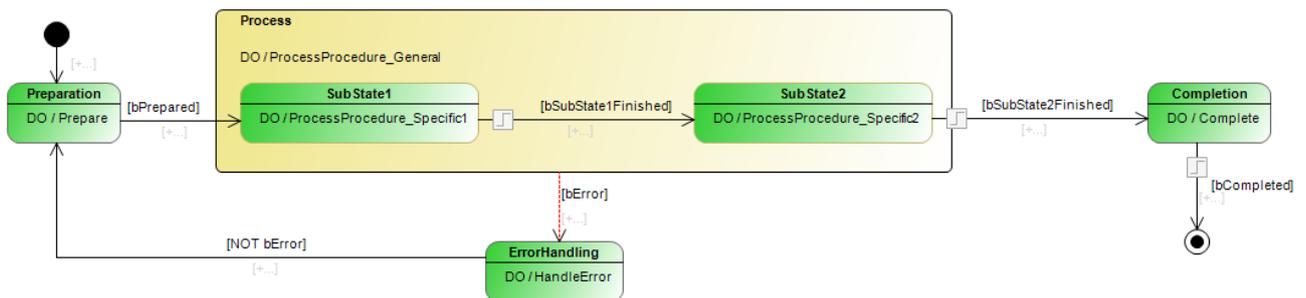
Composite state with exception and completion transition and with a region with start and end state:



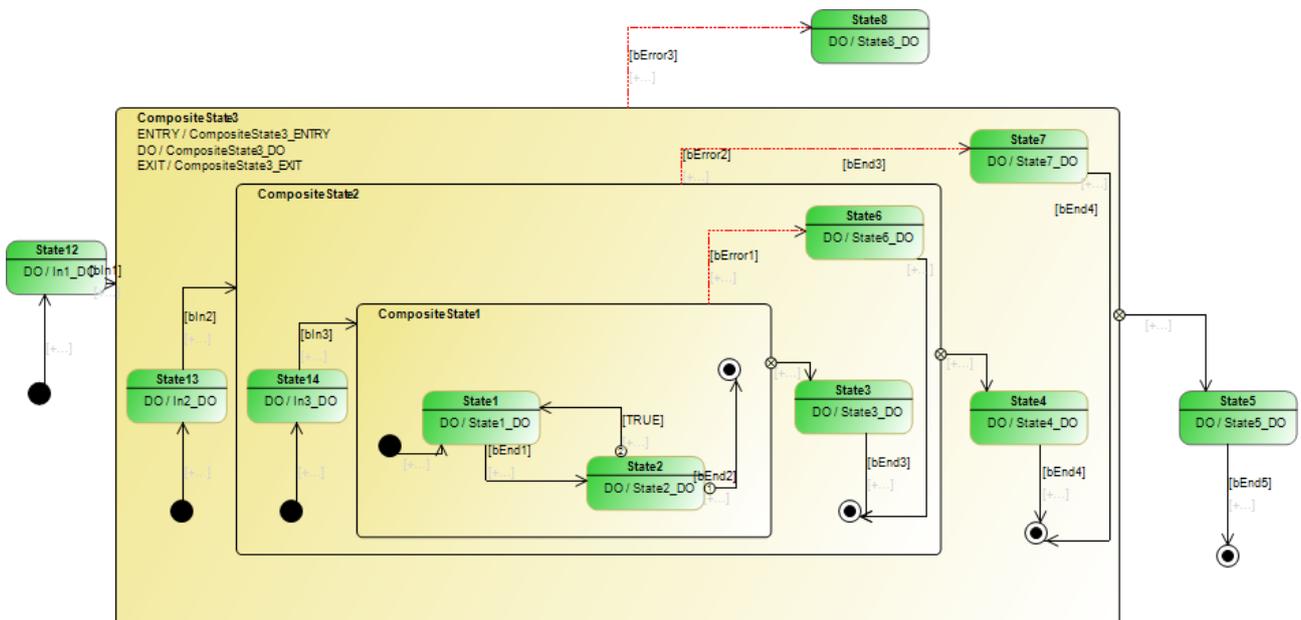
Composite state without completion transition and with a region without start and end state:



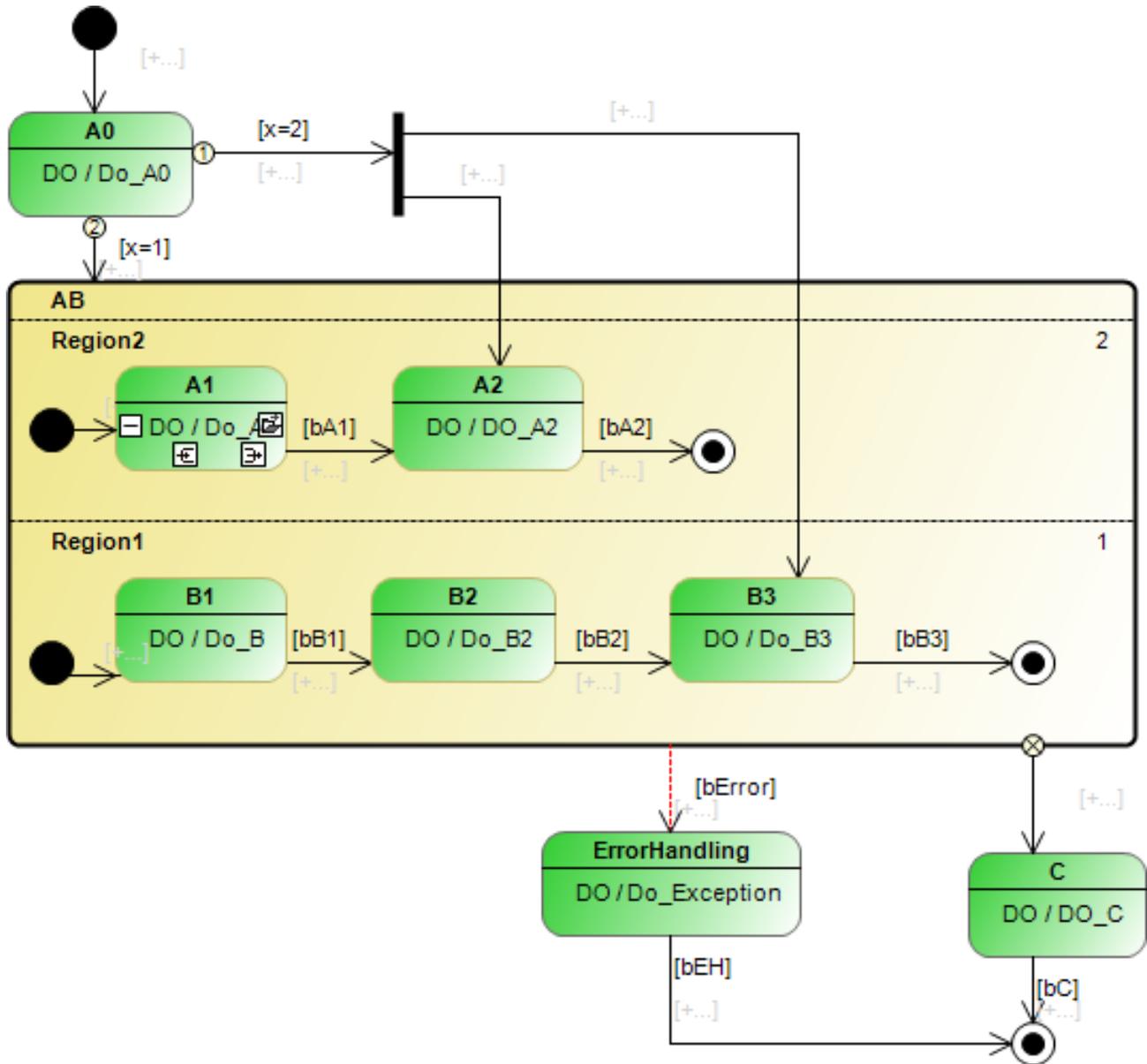
Composite state with one region and its own DO action:



Nested composite states, some with their own ENTRY/DO/EXIT action:



Composite state with several regions/orthogonal state with fork:



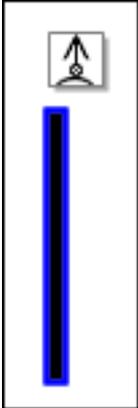
Sample "Lift": orthogonal state with completion and exception transition:



**Properties**

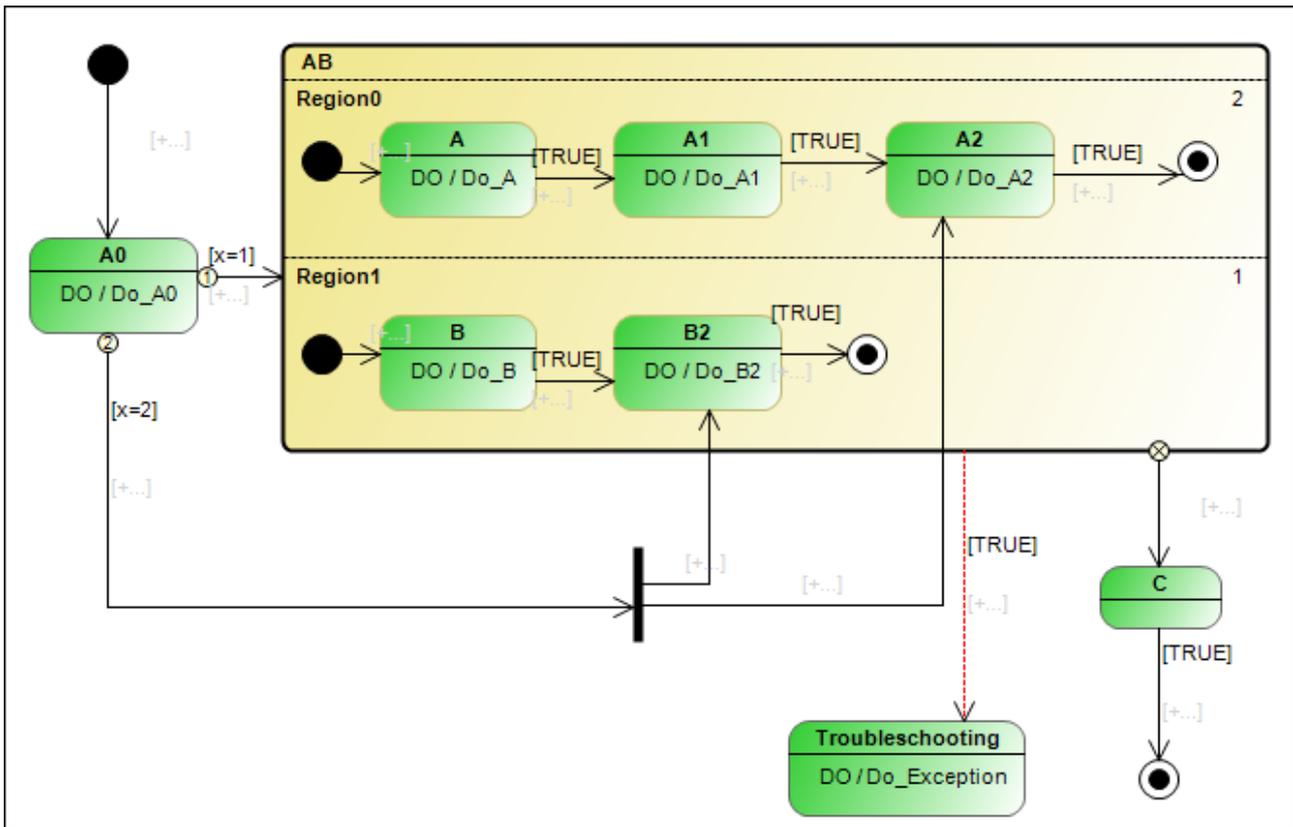
| "Property"           | Description   |
|----------------------|---|
| "Identifier"         | You can enter a name here. It is not displayed in the state diagram.  |
| "Vertical direction" | <ul style="list-style-type: none"> <li><input checked="" type="checkbox"/> : [Default] tick the checkbox to align the bar vertically.</li> <li><input type="checkbox"/> : untick the checkbox to align the bar horizontally.</li> </ul> |

**Edit fork**

| User input in the state diagram   | Response in the state diagram   | Description   |
|---|---|---|
| Focus on a fork.  |  | <p>The fork is editable.</p> <ul style="list-style-type: none"> <li>You can adjust the size of the fork.</li> <li>You can add a completion transition via the command icon displayed above the fork.</li> </ul> |
| Click on the green symbol                          |   | A completion transition is added. If you click on an existing state, it becomes the target state for the transition. Click on an empty area to create a new state.  |
| Pull one of the blue  squares to another position. |   | The size of the fork has been adjusted.   |

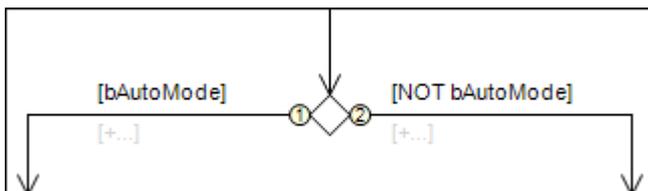
**Example**

Fork with orthogonal states



### 7.4.6 Choice

A **Choice** is a node in a state machine; at the node, an in-cycle evaluation of the subsequent guard conditions is done. It has at least one incoming and one outgoing transition. Choice nodes can be positioned top-level in statecharts, in composite states, and in regions. If several choices are connected by transitions, the connection must be designed in a way that no circular connection exists, such as recirculation to a choice that is already the source element of a transition in this chain.

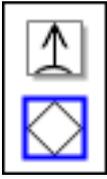


A choice is displayed as a diamond. The circle with the flow-control number indicates the order of processing.

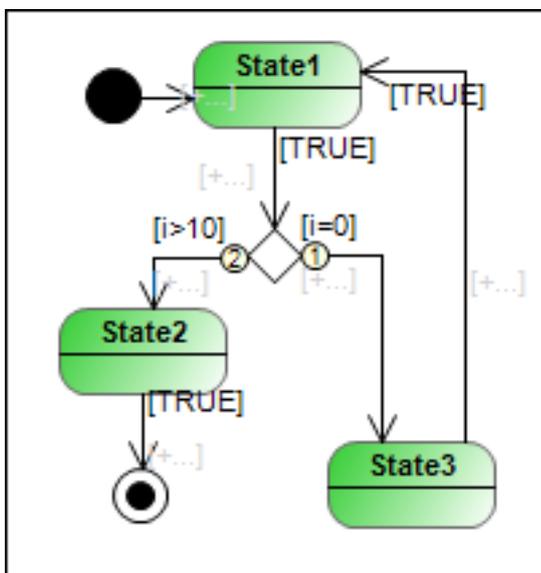
#### Properties

| "Property"   | Description  |
|--------------|--|
| "Identifier" | Here you can enter a name. It is not displayed in the state diagram. |

**Edit choice**

| User input in the state diagram   | Response in the state diagram   | Description  |
|---|---|--|
| Focus on a choice.  |  | The choice is editable.<br><ul style="list-style-type: none"> <li>You can add outgoing transitions via the command icon above the choice.</li> </ul>               |
| Click on the green symbol  |   | An outgoing transition is created. If you click on an existing state, it becomes the target state of the transition. Click in an empty area to create a new state. |

**Example of a choice**



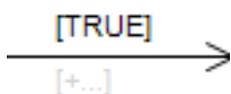
**7.4.7 Transition**

A **transition** controls the transition behavior between states. A transition is possible if one of the following events occurs:

- conditional event or change event
- Termination event (when the actions of the source state are complete)
- Time event

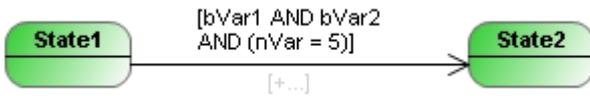
A transition has a guard condition and an optional action. The state transition is usually executed when the evaluation of the guard condition detects a rising edge. Alternatively, the transition can be configured to detect whether the guard condition is TRUE. In this case, advancing is independent of the edge behavior of the guard condition. When the state transition is executed with the next clock cycle, first the transition action is executed, then the target state is assumed.

If a source state has several outgoing transitions, each transition is assigned a priority. These can be changed in the element properties (View → “Properties” → “Priority”). The priority determines the order in which the guard conditions are checked and therefore in which order the transitions are switched.



A transition is displayed as a thin arrow pointing to the next state.

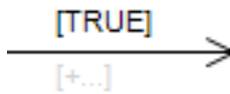
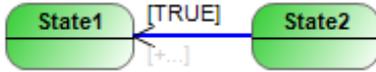
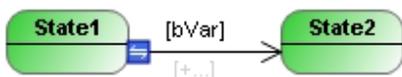
The guard condition of a transition can consist of several Boolean variables or expressions. In addition, transition elements, which can be added to a POU object as separate objects, can be used as transition condition. A new line can be added in the line editor for the transition condition via [Ctrl+Enter], in order to display the individual conditions more clearly.



**Properties**

| "Property"          | Description  |
|---------------------|--|
| "Relationship type" | Transition (not editable)  |
| "Priority"          | Priority, which determines the processing order<br>Sample: 1<br>Note: if the state has further transitions and you change the priority, all transitions are affected by the change and are automatically adjusted.   |
| "Rising edge"       | <ul style="list-style-type: none"> <li><input checked="" type="checkbox"/> : if the guard condition returns a rising edge (from 0 to 1), the transition is passed through and the state transition is executed.<br/>Tip: in the editor, a transition that behaves in this way is marked with the symbol .</li> <li><input type="checkbox"/> : if the guard condition is TRUE, the transition is executed.<br/>Tip: if the guard condition is always TRUE, the transition is executed once.</li> </ul> |

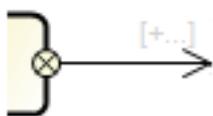
Edit transition

| User input in the state diagram   | Response in the state diagram   | Description  |
|---|---|--|
| Selecting "Transition" in "Toolbox".<br>Click on a state (source state) in the state diagram, then click on another state (target state). |    | A transition with TRUE condition and an action icon is created. Use this method to generate a transition between existing states.  |
| Click on a state (source state) in the state diagram.   |    | The icon through which an outgoing transition can be added to the selected state is available. If you use it, the state is extended with an outgoing transition. The target state is determined with a further click.  |
| Double-click on the symbol<br>                           |    | An input field with IntelliSense functionality opens. Select a Boolean variable or a Boolean expression. When the required option is selected in IntelliSense, it can be activated by double-clicking or by focusing plus [Enter].   |
| Two single clicks on  :                                  |    | An input field with IntelliSense functionality opens. Select a method or action. When the required option is selected in IntelliSense, it can be activated by double-clicking or by focusing plus [Enter].   |
| Click once on a transition, the action icon or the guard icon:  |  | The transition is selected (indicated by a blue arrow). A selected transition can be moved via drag and drop. The position of the linked states remains unchanged.   |
| Click on the start or end of a transition.  |  | The symbol  enables reconnecting of the transition, while retaining the existing transition configuration. You can pull the symbol to another source or target state in order to connect the transition there. The corresponding condition and action are retained. |

### 7.4.8 Completion Transition

A **completion transition** is unconditional. That is, it has **no** guard condition that triggers a switching operation. Instead, it switches without an additional condition, once the source state has been fully processed. During the next task cycle any action that may have been assigned is executed.

All outgoing transitions of a start state and a fork/join are completion transitions. A composite state can also have a completion transition (depending on the application case of the composite state).



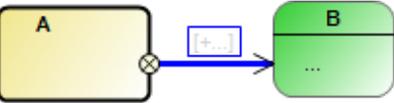
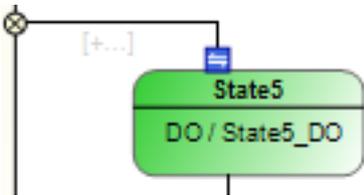
A completion transition is displayed as a thin arrow pointing to the target state. The small circle with a cross at the start of the arrow indicates that it is a completion transition.

**Properties**

| "Property"          | Description                          |
|---------------------|--------------------------------------|
| "Relationship type" | Completion transition (not editable) |
| "Priority"          | 1 (not editable)                     |

**Edit completion transition**

The user inputs in the state diagram editor can be summarized as follows:

| User input in the state diagram   | Response in the state diagram   | Description   |
|---|---|---|
| Select "Completion transition" in the toolbox. Click on a state and then on another state (target state). |    | An unconditional transition with an action icon is created. This method for generating a transition is used if the transition is to link existing states.   |
| Click on  .              |   | The icon is available when the corresponding state is selected. If you use it, the state is extended with an outgoing completion transition. The target state is determined with a further click.   |
| Click on the transition itself or on the action icon.   |    | The transition is selected (indicated by a blue arrow). A selected transition can be moved via drag and drop. The position of the linked states remains unchanged.  |
| Click  twice.          |  | An inline editor opens. When you start typing, the corresponding components appear under IntelliSense choices. A program/method/action can be selected by double-clicking.  |
| Click on the start or end of a completion transition.   |  | The symbol  enables reconnecting of the transition, while retaining the existing transition configuration. You can pull the symbol to another source or target state in order to connect the transition there. The corresponding action is maintained. |

**7.4.9 Exception Transition**

An **exception transition** controls the switching to the next state or pseudo state, if an error or an exception occurs. It has a guard condition and an optional action.

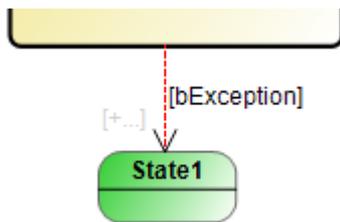
The source state is usually a composite or orthogonal state. Exception transitions cannot be used for "normal" states. However, a "normal" state is active as a rule within the composite or orthogonal state, which has an exception transition.

An exception transition ends the currently running execution and switches to the state that is foreseen for the reaction to this event. In this state the error handling and exception behavior are defined. Triggered by an exception, the exception transition ends the process of the currently active state or substate. The execution of the DO action(s) currently being executed will be completed.

**i** The exception transition is used to exit a composite state from every substate. Its state is evaluated **after** the DO action for the active state has been executed. Thus, even if the condition for the exception transition is already TRUE on the first occurrence of a composite state, the DO action of the first state is executed. Since ENTRY and EXIT actions are not tied to conditions, they are always executed independent of the exception transition.

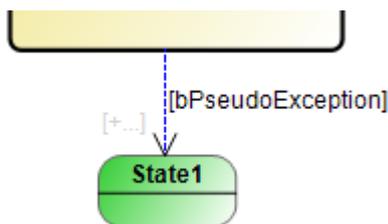
An exception transition can be interpreted such that it does not originate from a composite state, but directly from each substate. The conditions for the exception transitions are evaluated **after** their DO actions. Since ENTRY and EXIT actions are not tied to conditions, they are always executed independent of the exception transition.

An exception transition is indicated by a red dashed arrow pointing to the next state.



**Pseudo exception transitions**

A pseudo exception transition replaces the end state in a composite state. It is indicated by a blue dashed arrow.



In some cases it is clearer to draw a transition at the edge of the composite state whose condition is used as a basis for exiting the whole composite state, instead of an end state in a composite state that is linked to many transitions. This is similar to an exception transition, but without the interpretation that this represents a malfunction. It is therefore referred to as a pseudo exception transition. A pseudo exception transition has no influence on the cyclic execution behavior.

**Properties**

| "Property"          | Description  |
|---------------------|--|
| "Relationship type" | Exception transition (not editable)  |
| "Priority"          | <p>Priority, which determines the processing order</p> <p>Sample: 3</p> <p>Note: if the state has further transitions and you change the priority, all transitions are affected by the change and are automatically adjusted.</p>  |
| "Pseudo exception"  | <ul style="list-style-type: none"> <li><input type="checkbox"/> : exception transition. The arrow is shown with a dashed red line in the editor. [Default setting]</li> <li><input checked="" type="checkbox"/> : pseudo exception transitions. The arrow is shown with a dashed blue line in the editor. If the condition is met, the state is exited. However, it is not a fault signal that triggers advancing.</li> </ul>  |
| "Rising edge"       | <ul style="list-style-type: none"> <li><input checked="" type="checkbox"/> : if the guard condition returns a rising edge (from 0 to 1), the transition is passed through and the state transition is executed.<br/>Tip: in the editor, a transition that behaves in this way is marked with the symbol  .</li> <li><input type="checkbox"/> : if the guard condition is TRUE, the transition is passed through.</li> </ul> |

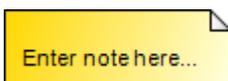
**Edit exception transition**

| User input in the state diagram   | Response in the state diagram | Description   |
|---|-------------------------------|---|
| Select "Exception transition" in the toolbox. Click on a composite state (source state) and then on another state (target state). |                               | An exception transition with TRUE condition and an action icon is created.  |
| Click on a composite state (source state) in the state diagram.   |                               | The icon through which an outgoing exception transition can be added to the selected state is available. If you use it, the state is extended with an outgoing exception transition. The target state is determined with a further click.                           |
| Double-click on the symbol<br><br>[TRUE]  |                               | An input field with IntelliSense functionality opens. Select a Boolean variable or a Boolean expression. When the required option is selected in IntelliSense, it can be activated by double-clicking or by focusing plus [Enter].                                  |
| Click the [+...] symbol twice   |                               | An input field with IntelliSense functionality opens. Select a method or action. When the required option is selected in IntelliSense, it can be activated by double-clicking or by focusing plus [Enter].  |
| Click on the transition, the action icon or the guard icon.   |                               | The transition is selected (indicated by a blue arrow). A selected transition can be moved via drag and drop. The position of the linked states remains unchanged.  |
| Click on the start or end of an exception transition.   |                               | The symbol  enables reconnecting of the transition, while retaining the existing transition configuration. You can pull the symbol to another source or target state in order to connect the transition there. The corresponding condition and action are retained. |

**7.4.10 Note**

You can insert a comment in the editor of a class diagram or statechart with the "Note" element.

At present only one-line notes are possible.



**Adding a note**

Select the "Note" tool:

In the editor, click the desired insertion position. Then double-click the text in the element and replace it with the desired text.

## Cursor

-  : The cursor has the form of a prohibition sign at points where insertion is not allowed.
-  : The cursor is a blue cross at points where it is allowed.

## 7.5 Object Properties

1. Select an UML Statechart in the project tree.
2. Open the context menu and select **Properties** or click on **View > Properties**.

The tabs that are relevant for UML are described below.

### UML

- **Abortable:**
  - Option available for: all Statecharts
  - Option enabled (default): Processing of the Statechart can be aborted. Such a Statechart has an additional internal variable: `_UML_SC_<name>.Abort`. If this variable is set with IEC code, processing of the Statechart can be stopped immediately, regardless of the internal state.
  - Option disabled: Processing of the Statechart cannot be aborted.
- **UseVarInst:**
  - Option available for: Statecharts, which were created as "Method" program elements and belong to a function block
  - Option enabled (default): The data of the Statechart methods are not temporary; they are held as instance variables for the corresponding function block. If you declare the variables of the method as `VAR_INST`, the data are held in the same way as for a function block and are no longer temporary, so that the Statechart is passed through in several task cycles as usual.
  - Option disabled: The data of the Statechart methods are held on a temporary basis. Methods and their data are usually temporary. If you implement a method with the implementation language UML Statechart and disable the option "UseVarInst", the method behaves like a cycle-internal state and reinitializes the data at the start of the task cycle. In this case only cycle-internal states can be used.
  - Note that this option is only available for Statechart methods that belong to a function block, not for methods that belong to a program, since programs cannot contain `VAR_INST` declarations.

## 7.6 Online Mode

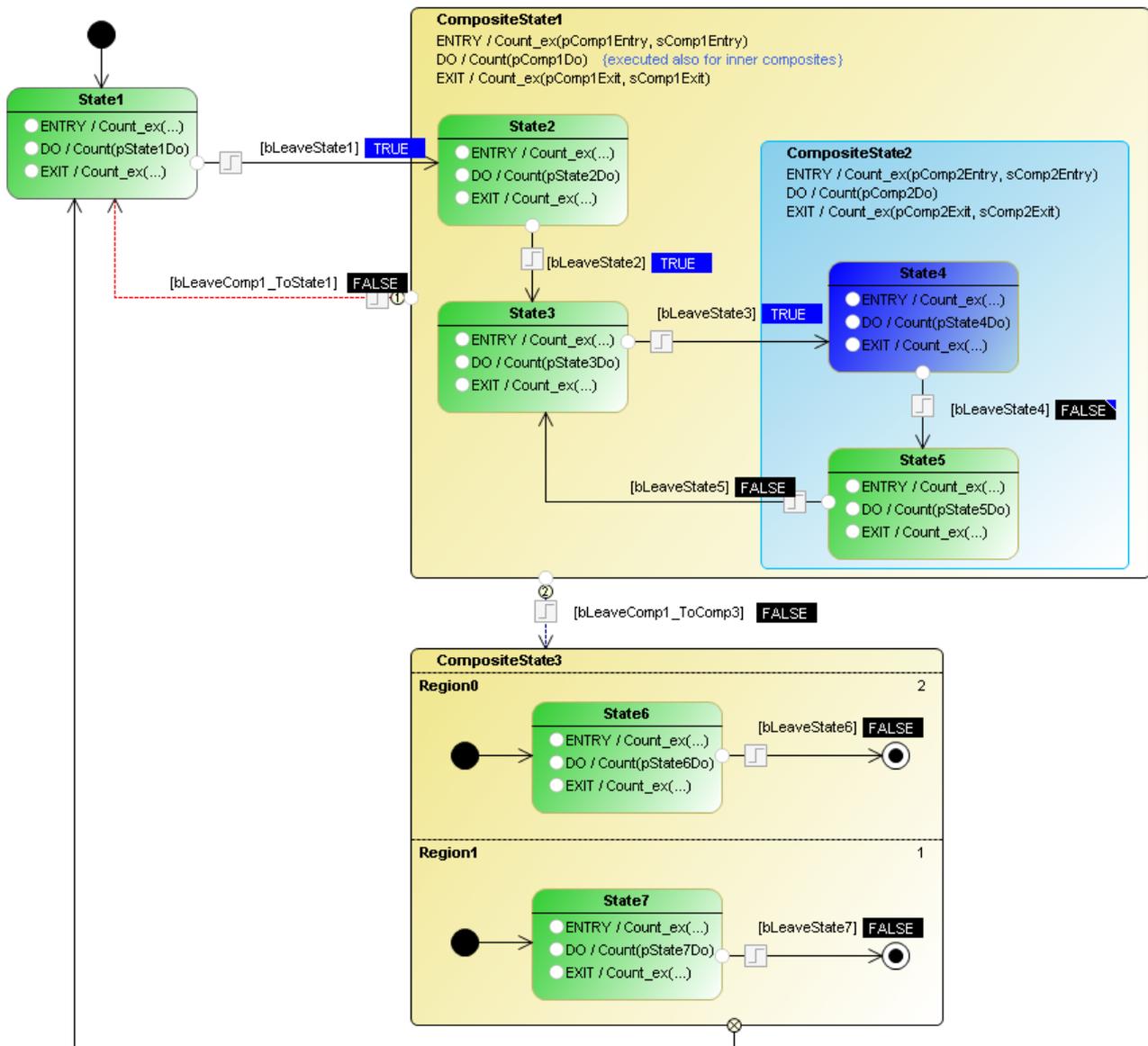
The behavior of Statechart objects in online mode corresponds to the normal online behavior in TwinCAT 3. As usual it is possible to monitor a loaded application, write or force variables, set breakpoints or process a program in single cycles.

In online mode an active state is shown in blue, an active composite state in light blue. In online mode the value of a transition expression is displayed next to the transition. If the transition consists of only one transition variable, the value of the variable can be changed for writing or forcing by double-clicking on the monitoring value. If a value has been prepared for writing or forcing, this is indicated in the top right corner of the monitoring area (see the monitoring field of the transition variable `bLeaveState4` in the center of right-hand edge in the image below: the blue corner indicates that the variable value TRUE is ready for writing or forcing).

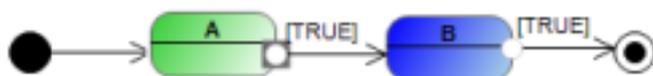
An additional debugging option is provided by transferring the variables of a transition to the watch list via a command. To this end select [Add Watch \[► 58\]](#) from the context menu of the transition. This command is particularly useful if a transition expression consists of a combination of several variables (e.g. "bCondition1 AND bCondition2"). If you execute the command **Add Watch** for this transition, both variables of the

transition expression are added to the watch list, i.e. *bCondition1* and *bCondition2*. This gives you a simple overview of which sub-signal of the overall expression does not yet have the required value for switching to the next state.

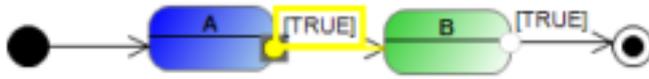
Furthermore, the commands [Go To Definition \[► 58\]](#) and [Find All References \[► 58\]](#) are supported for transitions.



All possible positions for breakpoints are shown as white circles with grey edge in the Statechart. To set a breakpoint, select a circle. The circle is highlighted in grey: .



Select **Debug > Toggle Breakpoint** or press [F9]. The program stops at the breakpoint, and the point is highlighted in yellow in the Statechart.



### Online Change

Offline edits to an existing Statechart can usually be loaded to the controller with an online change without re-initializing the chart if you change transitions, conditions, priorities, actions or names. The state that was active before the online change is thus still active after the online change.

However, in the case of an online change the Statechart is always restarted –i.e. the start state is activated – if a state or a region is added, removed or replaced. This also applies to implicit states such as the "Selection" element, or to the property for making a Statechart "abortable".

## 8 FAQ

### OOP and UML

#### Does object-oriented programming (OOP) and UML always have to be used together?

- The combined use of OOP and UML offers many benefits (see next question), although it is not compulsory. Object-oriented programming of applications is also possible without using UML. Likewise, UML can be used in PLC projects, which are not based on object-oriented programming.

#### What are the benefits of using OOP and UML together?

- In order to make the most of OOP, the structure of an object-oriented software should be designed and created before the implementation (e.g. What classes are available, what is their relationship, what functionalities do they offer, etc.). Before, during and after programming, documentation helps to understand, analyze and maintain the software.
- As an analysis, design and documentation tool for software, UML offers options for planning, creating and documenting the application. UML is particularly suitable for object-oriented implementations, since modular software is particularly suitable for representation with the aid of a graphical language.
- For example, the class diagram is used for analyzing and creating the program structure. The more modular the software structure, the easier and more efficient the class diagram can be used (e.g. Graphical representation of separate function blocks with individual methods for providing the functionalities etc.).
- The state diagram can be used to specify the sequence of a system with discrete events. The more consistent the object- and event-orientation of the software structure, the more transparent and effective the state machines can be designed (e.g. The behavior of modules/systems is based on a state model with states (such as startup, production, pause); within the states corresponding functionalities are called, which are encapsulated in methods (such as startup, execute, pause) etc.).

#### UML state diagram methods with deactivated "UseVarInst" option

The following three queries refer to methods with the implementation language **UML state diagram**, which in addition meet one of the following conditions:

- The state diagram method belongs to a program.
- The state diagram method belongs to a function block, and the option **UseVarInst** of the method is disabled.

State diagram methods that meet these conditions are referred to as "the methods mentioned" in the questions below.

Further information on the option **UseVarInst** can be found under [UML state diagram > Object properties](#).

[▶ 84]

#### Why can these methods only contain cycle-internal states, and why is there no online view?

- As a general rule, all data of a method are temporary and only valid while the method is executed (stack variables). If, however, a method variable is declared as instance variable with `VAR_INST`, it is not stored on the methods stack, but on the stack for the function block instance. It therefore behaves like other variables of the function block instance and is not reinitialized with each call of the method.
- Since the data of the state diagram method mentioned are not declared as instance variables, they are cycle-internal state machines. The states contained in the diagrams are therefore switched cycle-internally, not by the task cycle, since the method execution "restarts" with each task cycle, due to the temporary nature of the data management. Online view is therefore not available for these state diagram methods, since the variables of the state machine are temporary, and the active state can change repeatedly within a cycle.
- This is in contrast to state diagrams without temporary data management (e.g. programs or function blocks as state diagram, function block method as state diagram with **UseVarInst** option enabled), in which the states are switched by the task cycle as standard and can optionally be configured as "cycle-internal".

#### Why does the tool list for the methods mentioned not contain composite states, forks and joins?

- Since the data of these methods are temporary and only valid while the method is executed, the UML state diagram methods can only have cycle-internal states (see previous question).
- A composite state would therefore have to be completed within a cycle. This means the element would always be processed sequentially, even if execution of a region is not completed.
- This behavior would therefore be fundamentally different than for "normal" composite states, which can remember their internal state.
- So as not to offer different behaviors for a composite state, depending on the POU type with which the element is used, the composite states are not allowed in the methods mentioned.
- Since forks and joins are only permitted in conjunction with composite states, these elements are not available in den state diagram methods mentioned.

**What should be considered when programming the methods mentioned? What could be the reason for high system load when these methods are executed?**

- As explained under the previous questions, the UML state diagram methods are cycle-internal state machines. Therefore, they are operated cycle-internally, not based on the task cycle.
- This means that completion of the actions of an internal state is followed immediately by the transition. The transition condition is checked immediately, and the transition action is executed when the condition is met. Also, the system immediately switches to the target state when the transition condition is met.
- If the transition condition is not met, the system does not switch to the target state, and the current state remains active. Due to the cycle-internal states, the DO action for the current state is called again in the same cycle, if the maximum number of DO-cycle calls was not yet reached.
  - **Max. DO-cycle calls:** The maximum number of DO-cycle calls for a state can be configured, whereby a value between 1 and 32767 can be set. This number indicates the maximum number of DO-action calls.
- Note the following: If the transition condition is not met during the whole task cycle, the DO-action of the current active state is executed in this cycle until the maximum number of DO-cycle calls in this state is reached.

Sample: If a value of 32767 is set for "max. DO-cycle calls" and the transition condition is not met during the task cycle, the DO-action of the corresponding state is executed 32767 times in a cycle! Since this could lead to high system load, depending on the scope of the DO-action, the state machine and in particular the transition condition and the maximum number of DO-cycle calls should be verified for compliance with the required application behavior.

## 9 Samples

### 9.1 UML class diagram

#### 9.1.1 1 Basics

This UML class diagram example illustrates the basic functionality of the [UML class diagram \[▶ 17\]](#).

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716013451/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716013451/.zip)

##### ClassDiagram\_1

This class diagram illustrates the structure of the PLC project. It contains various PLC elements (such as function block, interface, global variable list etc.).

The class diagram also illustrates the relationship between the elements (inheritance between function blocks, implementation of an interface, instantiation of a function block).

##### ClassDiagram\_2

This class diagram explains and explores the following basic functionalities:

- Visualize existing element from the cross-references on the diagram
  - Display the "Toolbox" window
  - Select FB\_InfoData on the class diagram
  - Toolbox: elements that are related to the selected element but are not included in the class diagram are displayed under the headings "Incoming cross references" and "Outgoing cross references".
  - You can drag and drop the element onto the diagram, so that the element is shown in the class diagram.
  - For more information see: [Adding existing elements to a diagram \[▶ 20\]](#).
- Visualize existing element from the project tree on the diagram
  - Select an element of type POU, INTERFACE, GVL or DUT in the project tree and drag and drop it onto the opened class diagram.
  - Drop it in a suitable location to visualize it there.
  - The corresponding element is then displayed in the diagram. If relationships with already shown elements exist, these are displayed automatically.
  - For more information see: [Adding existing elements to a diagram \[▶ 20\]](#).
- Adding new elements or creating new relationships between elements
  - Use the elements of the "Toolbox" window.
  - For more information see: [Editing a class diagram \[▶ 22\]](#) and [Elements \[▶ 25\]](#).

#### 9.1.2 2 Simple machine

This "UML class diagram" sample illustrates the basic functionality of the [UML class diagram \[▶ 17\]](#) based on a machine with object-oriented programming.



Note that the modules have not been functionally implemented. The sample is designed to describe the functionalities of the UML class diagram and to demonstrate them using an exemplary program structure.

---

With the help of the class diagram, the structure of a PLC program can be created, extended and modified. In addition, as in this sample, the structure of a PLC program can be documented with the help of the class diagram and easily understood on the basis of this graphic.

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716015115/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716015115/.zip)

## Machine modules

The application has the following levels and modules.

Machine level: machine (FB\_Machine)

Subsystem level: ejection (FB\_Ejector)

Submodule level: cylinder with or without hardware feedback signal (FB\_Cylinder, FB\_CylinderFeedback)

## Inheritance

Since all these function blocks have common features (common data and functionalities), a base class is provided where these common features are implemented once for all function blocks. These implementations are passed on to the subclasses via the inheritance mechanism offered by object-oriented programming.

- The four machine modules extend FB\_ModuleRoot.

Furthermore, the cylinder with feedback functionality represents an extension of the cylinder without this functionality. Therefore, inheritance is also used here.

- FB\_CylinderFeedback extends FB\_Cylinder.

## Interface

To define the requirements for different cylinder types, the basic methods and properties that a cylinder must provide are defined in an interface.

- I\_Cylinder defines the cylinder requirements.
- FB\_Cylinder implements I\_Cylinder.

## Instantiation

- MAIN instantiates FB\_Machine.
- FB\_Machine instantiates FB\_Ejector twice, since the machine has two ejector modules.
- FB\_Ejector instantiates I\_Cylinder, FB\_Cylinder and FB\_CylinderFeedback. As an example, only one cylinder is active at a time, either the cylinder with feedback functionality or the cylinder without feedback functionality. The interface variable iCylinder is assigned the currently active cylinder instance. This allows the active cylinder to be controlled in a generalized manner via the interface variable.
- FB\_ModuleRoot instantiates ST\_Error.

## 9.2 UML state diagram

In the following UML SC samples, the level of difficulty and the scope increase with ascending number of the sample.

### 9.2.1 1 Lamp

This "UML state diagram" sample illustrates the basic functionality of the [UML state diagram \[► 51\]](#) and contains the following UML SC elements:

- [Start State \[► 59\]](#)
- [State \[► 60\]](#)
- [Transition \[► 77\]](#)

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716016779/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716016779/.zip)

## Overview

UML SC is used to program the behavior of a lamp, which can be switched on and off via a switch on the visualization. The lamp has two states, "On" and "Off".

Each state contains one ENTRY action and one DO action.

- The lamp is switched on or off in the ENTRY actions. The ENTRY action is called once whenever the corresponding state is activated.
- An ON or OFF counter is incremented in the DO actions. This indicates that a DO action is called permanently as long as the corresponding state is active.
- The value of the switch is requested as a transition between the states.

## Visualization

In addition to the convenient [online mode](#) [▶ 84] of the UML SC diagram, the behavior of the lamp and the values of the counters can be followed via the visualization.

## 9.2.2 2 Pedestrian traffic light

This "UML state diagram" sample illustrates the basic functionality of the [UML state diagram](#) [▶ 51] and contains the following UML SC elements:

- [Start State](#) [▶ 59]
- [State](#) [▶ 60]
- [Transition](#) [▶ 77]

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716018443/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716018443/.zip)

## Overview

UML SC is used to program the behavior of a pedestrian traffic light. A green-phase request can be sent via a push button on the visualization. The pedestrian traffic light has the two states, "Red" and "Green".

If a green phase is requested, the traffic light switches to green after the time "cTimeWaitForGreen" has elapsed. The traffic light switches back to red once the time "cTimeGreenPhase" has elapsed.

The two states "Red" and "Green" each contain an ENTRY action and a DO action.

- The ENTRY actions are used to switch the traffic light to red or green. In addition, the respective timer function block (of type TON) is reset. The ENTRY action is called once whenever the corresponding state is activated.
- In the DO actions a Red or Green counter is incremented. This indicates that a DO action is called permanently as long as the corresponding state is active. The respective timer function block is also called.
- The Q output of the respective timer function block is used as a transition between the states, so that the state is changed after the corresponding time has elapsed.

## Visualization

In addition to the convenient [online mode](#) [▶ 84] of the UML SC diagram, the behavior of the pedestrian lights, the values of the counters and the waiting time that has already elapsed can be tracked via the visualization.

## 9.2.3 3 SaveText simulation

This "UML state diagram" sample illustrates the basic functionality of the [UML state diagram](#) [▶ 51] and contains the following UML SC elements:

- [Start State](#) [▶ 59]
- [State](#) [▶ 60]

- [Transition \[▶ 77\]](#)
- [Choice \[▶ 76\]](#)
- [Composite State \[▶ 64\]](#)
- [Completion Transition \[▶ 79\]](#)

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716020107/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716020107/.zip)

## Overview

UML SC is used to simulate the behavior of an application in which text is to be saved in XML or text format. The information is then to be sent to a client and a master.

This behavior has not actually been implemented in the sample project. Instead, the outer shell or basic structure of the state machine is implemented using UML SC as an example.

## States

The UML SC diagram has the following states:

- InitState
- XmlFormat
- TextFormat
- PublishToMaster
- PublishToClient

## Counter/call frequency

The states each contain a DO action and an ENTRY and/or EXIT action. Each action increments a corresponding counter. This indicates that:

- the ENTRY action is called once each time the corresponding state is activated;
- the EXIT action is called once each time the corresponding state is exited;
- the DO action is called permanently as long as the associated state is active.

## Selection element

If the "XmlFormat" or "TextFormat" state is activated, the selection element is used as a decision basis. The element has two outgoing transitions. The transition with the higher priority (i.e. with the lower priority number) is checked first. If the associated transition condition is met, the target state is activated.

## Composite state

The "PublishToMaster" and "PublishToClient" states are located in different regions of a composite state, so that they are processed in pseudo-parallel mode. This means that both states are active simultaneously, but the region with the higher priority (i.e. with the lower priority number) is executed first. Colored highlighting on the visualization indicates which region was processed first. Furthermore, during the execution of the sample project it becomes apparent that the composite state is only exited when both substate machines have reached their end state.

Project modification possible: the priorities of the regions can be changed by adjusting the number in the upper right corner of the region. If you configure Region1 with priority 1 and download this change via an online change, next time you activate the composite state you can see that "Client first" is now highlighted instead of "Master first". Consequently, Region1 was executed first.

## Visualization

In addition to the convenient [online mode \[▶ 84\]](#) of the UML SC diagram, the behavior of the state machine and the values of the counters can be tracked via the visualization. Switches on the visualization facilitate switching back and forth between the states.



The values of the transition conditions can be changed not only via the visualization or the declaration editor of the FB, but also directly via the UML SC diagram.

If the transition consists of only one transition variable, the value of the variable can be changed for writing or forcing by double-clicking on the monitoring value. If a value was prepared for writing or forcing, this is indicated in the upper right-hand corner of the monitoring area (see also: [Online Mode \[▶ 84\]](#)).

## 9.2.4 4 Call Behavior - Basis

This "UML state diagram" sample illustrates the basic calling behavior of the [UML state diagram \[▶ 51\]](#) and includes the following UML SC elements:

- [Start State \[▶ 59\]](#)
- [State \[▶ 60\]](#)
- [Transition \[▶ 77\]](#)
- [Composite State \[▶ 64\]](#)
- [Completion Transition \[▶ 79\]](#)
- [Exception Transition \[▶ 80\]](#)

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716021771/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716021771/.zip)

### Overview

The following sub-aspects illustrate the call behavior of states and composite states. With the help of different programming means the different calling aspects are illustrated (programming tools => calling aspect).

- Counter => Call frequency
- Entering the called state action in an array => Call sequence
- Task cycle IDs => Assignment of the call to the cycle ID

The respective assignment of the programming tools to the call aspect is explained below.

Both use cases are applied for the composite state:

- Grouping/Nesting, in which case the composite state has its own ENTRY/DO/EXIT actions
- Parallel sub-state machines

### Counter => Call frequency

Each ENTRY/DO/EXIT action, that is called in a state or a composite state, increments a corresponding counter. This illustrates the call frequency of an action and indicates that:

- the ENTRY action is called once each time the associated (composite) state is activated;
- the EXIT action is called once each time the associated (composite) state is exited;
- the DO action is called permanently as long as the associated (composite) state is active;
- a composite state with a region (use case Grouping/Nesting) is activated if none of the inner states was previously active and as soon as one of the inner states is activated;
- a composite state with a region (use case Grouping/Nesting) remains active as long as one of the inner states is active;
- a composite state with a region (use case Grouping/Nesting) is exited if one of the inner states was previously active and this state is exited so that none of the inner states is now active;
- it depends on the option "Execute DO actions even if inner composite states are active" whether the DO action of an outer composite state (in the sample "CompositeState1") is still called if an inner composite state is active (in the sample project: "CompositeState2").

## Entering the called state action in an array => Call sequence

Each ENTRY/DO/EXIT action that is called in a state or in a composite state enters its name (e.g. "State1\_\_Entry") into an array. This illustrates the call sequence of the different actions and indicates that:

- the basic call sequence for a state is: first ENTRY, then DO, then EXIT. In the sample project:
  - State1\_\_Entry
  - State1\_\_Do
  - State1\_\_Exit
- if an inner state of a composite state is activated with a region (use case grouping/nesting) and if this also activates the composite state, the call sequence "from outside to inside" is: outer state EXIT, composite state ENTRY, inner state ENTRY, composite state DO, inner state DO. In the sample project:
  - State1\_\_Exit
  - Comp1\_\_Entry
  - State2\_\_Entry
  - Comp1\_\_Do
  - State2\_\_Do
- an inner composite state can be exited via a (pseudo) exception transition of the outer composite state (in the sample project via the transitions: eLeaveComp1\_ToState1, eLeaveComp1\_ToComp3).
- if a composite state is exited e.g. via a (pseudo) exception transition, the calling sequence "from inside to outside" is: inner state EXIT, if necessary inner composite state EXIT, outer composite state EXIT, outer state ENTRY. In the sample project:
  - State4\_\_Exit
  - Comp2\_\_Exit
  - Comp1\_\_Exit
  - State1\_\_Entry / State7\_\_Entry
- in a composite state with several regions, the region with the higher priority, i.e. with the lower priority number, is called first. In the sample project:
  - State7\_\_Entry
  - State7\_\_Do
  - State6\_\_Entry
  - State6\_\_Do

## Task cycle IDs => Assignment of the call to the cycle ID

Each ENTRY/DO/EXIT action that is called in a state or a composite state stores the cycle ID of the first and last call. This illustrates which call takes place in which cycle and indicates that:

- if a state is activated, the ENTRY and DO actions are called within the same cycle.  
In the screenshot "Visu\_B\_CallingOrder\_CycleNr" of the ZIP:
  - State1 / ENTRY-Call: 516 (ID of the cycle in which the call occurred)
  - State1 / DO-First Call: 516
- if a state is exited, the EXIT action is called in the following cycle of the last DO call.  
In the screenshot "Visu\_B\_CallingOrder\_CycleNr" of the ZIP:
  - State1 / DO-Last Call: 1415
  - State1 / EXIT-Call: 1416
- if a state is exited, the EXIT action of that state is called in the same cycle as the ENTRY and DO actions of the activated state.  
In the screenshot "Visu\_B\_CallingOrder\_CycleNr" of the ZIP:
  - State2 / DO-Last Call: 1762
  - State2 / EXIT-Call: 1763
  - State3 / ENTRY-Call: 1763

- State3 / DO-First Call: 1763
- if a state is exited and a composite state with a region and its own actions is activated, the EXIT action of the exited state is called in the same cycle as the ENTRY and DO actions of the activated (composite) state.  
In the screenshot "Visu\_B\_CallingOrder\_CycleNr" of the ZIP:
  - State1 / DO-Last Call: 1415
  - State1 / EXIT-Call: 1416
  - CompositeState1 / ENTRY-Call: 1416
  - State2 / ENTRY-Call: 1416
  - CompositeState1 / DO-First Call: 1416
  - State2 / DO-First Call: 1416
- if a composite state with a region and its own actions is exited via a **pseudo exception transition**, the EXIT action of the exited (composite) state is called in the same cycle as the ENTRY and DO actions of the activated state.  
In the screenshot "Visu\_B\_CallingOrder\_CycleNr" of the ZIP:
  - CompositeState1 / DO-Last Call: 2367
  - CompositeState2 / DO-Last Call: 2367
  - State4 / DO-Last Call: 2367
  - State4 / EXIT-Call: 2368
  - CompositeState2 / EXIT-Call: 2368
  - CompositeState1 / EXIT-Call: 2368
  - State7 / ENTRY-Call: 2368
  - State7 / DO-First Call: 2368
  - State6 / ENTRY-Call: 2368
  - State6 / DO-First Call: 2368
- if a composite state with a region and its own actions is exited via an **exception transition**, the EXIT action of the exited (composite) state is called in the same cycle as the last call of the DO action of the exited (composite) state and as the ENTRY and DO actions of the activated state. For example (**not** shown in the screenshot "Visu\_B\_CallingOrder\_CycleNr" of the ZIP):
  - CompositeState1 / DO-Last Call: 1050
  - CompositeState2 / DO-Last Call: 1050
  - State4 / DO-Last Call: 1050
  - State4 / EXIT-Call: 1050
  - CompositeState2 / EXIT-Call: 1050
  - CompositeState1 / EXIT-Call: 1050
  - State1 / ENTRY-Call: 1050
  - State1 / DO-First Call: 1050

## Visualization

In addition to the convenient [online mode \[► 84\]](#) of the UML SC diagram, the behavior of the state machine and the values of the counters, the table and the task cycle IDs can be tracked via the visualizations. The visualization also contains switches for switching between the states and for resetting the values.

In order to facilitate familiarization with the sample project, two visualizations with different information content are included:

- Visu\_A\_CallingOrder contains:
  - Counter => Call frequency
  - Entering the called state action in an array => Call sequence
- Visu\_B\_CallingOrder\_CycleNr extends Visu\_A by:
  - Task cycle IDs => Assignment of the call to the cycle ID



The values of the transition conditions can be changed not only via the visualization or the declaration editor of the FB, but also directly via the UML SC diagram.

If the transition consists of only one transition variable, the value of the variable can be changed for writing or forcing by double-clicking on the monitoring value. If a value was prepared for writing or forcing, this is indicated in the upper right-hand corner of the monitoring area (see also: [Online Mode](#) [▶ 84]).

## 9.2.5 5 Call behavior - transition action



### Basic sample

This "UML state diagram" sample is based on the sample [4 Call Behavior - Basis](#) [▶ 93]. All elements and calling aspects of sample 4 are also contained in this sample, so that the explanations there also apply to this sample and are necessary for understanding this sample. Only the additions to sample 4 are described below. Therefore, please read the description of sample 4 first.

Complementing sample [4 Call Behavior - Basis](#) [▶ 93], this "UML state diagram" sample illustrates another aspect of the call behavior from [UML state diagram](#) [▶ 51]. For this purpose, the following UML SC elements are additionally included:

- Transitions with transition actions ([Transition](#) [▶ 77], [Completion Transition](#) [▶ 79], [Exception Transition](#) [▶ 80])

Sample project: [https://infosys.beckhoff.com/content/1033/TF1910\\_Tc3\\_UML/Resources/11716023435/.zip](https://infosys.beckhoff.com/content/1033/TF1910_Tc3_UML/Resources/11716023435/.zip)

### Overview:

The following sub-aspects illustrate the call behavior of states, composite states and transitions. With the help of different programming means the different calling aspects are illustrated (programming tools => calling aspect).

- Counter => Call frequency
- Entering the called state action in an array => Call sequence
- Task cycle IDs => Assignment of the call to the cycle ID

The respective assignment of the programming tools to the call aspect is explained below.

### Counter => Call frequency:

A transition action is called once when the state transition is executed via the transition.

### Entering the called state action in an array => Call sequence:

Each action called in a state, composite state or transition enters its name (e.g. "State1\_\_Entry" or "Transition\_LeaveState1") into an array. This illustrates the call sequence of the different actions and indicates that:

- the basic call sequence for a state is: first ENTRY, DO, EXIT and then the transition action. Then the ENTRY action of the newly activated state is called. In the sample project:
  - State2\_\_Exit
  - Transition\_LeaveState2
  - State3\_\_Entry
- if an inner state of a composite state is activated with a region (use case grouping/nesting) and if this also activates the composite state, the call sequence "from outside to inside" is: outer state EXIT, transition action, composite state ENTRY, inner state ENTRY, composite state DO, inner state DO. In the sample project:
  - State1\_\_Exit
  - Transition\_LeaveState1
  - Comp1\_\_Entry

- State2\_\_Entry
- Comp1\_Do
- State2\_Do
- if a composite state is exited e.g. via a (pseudo) exception transition, the calling sequence "from inside to outside" is: inner state EXIT, if necessary inner composite state EXIT, outer composite state EXIT, transition action, outer state ENTRY. In the sample project:
  - State4\_\_Exit
  - Comp2\_Exit
  - Comp1\_Exit
  - Transition\_LeaveComp1\_ToComp3
  - State7\_\_Entry

### Task cycle IDs => Assignment of the call to the cycle ID:

Each action called in a state, composite state or transition stores the cycle ID of the respective first and last call. This illustrates which call takes place in which cycle and indicates that:

- the state transition is executed in the following cycle after the transition condition is met, and that when a state is exited, the transition action is called in the same cycle as the EXIT action of the exited state and as the ENTRY and DO actions of the activated state.  
In the screenshot "Visu\_CallingOrder\_CycleNr" of the ZIP:
  - State2 / DO-Last Call: 1233 (ID of the cycle in which the call occurred)
  - Transition / Rising Edge: 1233
  - State2 / EXIT-Call: 1234
  - Transition-Call: 1234
  - State3 / ENTRY-Call: 1234
  - State3 / DO-First Call: 1234
- if a state is exited and a composite state with a region and its own actions is activated, the transition action is called in the same cycle as the EXIT action of the exited state and as the ENTRY and DO actions of the activated (composite) state.  
In the screenshot "Visu\_CallingOrder\_CycleNr" of the ZIP:
  - State1 / DO-Last Call: 1055
  - Transition / Rising Edge: 1055
  - State1 / EXIT-Call: 1056
  - Transition / Call: 1056
  - CompositeState1 / ENTRY-Call: 1056
  - State2 / ENTRY-Call: 1056
  - CompositeState1 / DO-First Call: 1056
  - State2 / DO-First Call: 1056
- if a composite state with a region and its own actions is deactivated, the transition action is called in the same cycle as the EXIT action of the exited (composite) state and as the ENTRY and DO actions of the activated state.  
In the screenshot "Visu\_CallingOrder\_CycleNr" of the ZIP:
  - CompositeState1 / DO-Last Call: 1743
  - CompositeState2 / DO-Last Call: 1743
  - State4 / DO-Last Call: 1743
  - Transition / Rising Edge: 1743
  - State4 / EXIT-Call: 1744
  - CompositeState2 / EXIT-Call: 1744
  - CompositeState1 / EXIT-Call: 1744
  - Transition / Call: 1744

- State7 / ENTRY-Call: 1744
- State7 / DO-First Call: 1744
- State6 / ENTRY-Call: 1744
- State6 / DO-First Call: 1744

**Visualization:**

In addition to the convenient [online mode \[▶ 84\]](#) of the UML SC diagram, the behavior of the state machine and the values of the counters, the table and the task cycle IDs can be tracked via the visualizations. The visualization also contains switches for switching between the states and for resetting the values.



The values of the transition conditions can be changed not only via the visualization or the declaration editor of the FB, but also directly via the UML SC diagram.

---

If the transition consists of only one transition variable, the value of the variable can be changed for writing or forcing by double-clicking on the monitoring value. If a value was prepared for writing or forcing, this is indicated in the upper right-hand corner of the monitoring area (see also: [Online Mode \[▶ 84\]](#)).



More Information:  
**[www.beckhoff.com/tf1910](http://www.beckhoff.com/tf1910)**

Beckhoff Automation GmbH & Co. KG  
Hülshorstweg 20  
33415 Verl  
Germany  
Phone: +49 5246 9630  
[info@beckhoff.com](mailto:info@beckhoff.com)  
[www.beckhoff.com](http://www.beckhoff.com)

