TM835

APROL ST-SFC Configuration

PERFECTION IN AUTOMATION
www.br-automation.com

Requirements	
Training modules	TM213 – Automation Runtime TM242 – Sequential Function Chart (SFC) TM246 – Structured Text SEM841.5 – APROL Process Control Training: Basic 1
Software	APROL Automation Runtime SuSE LINUX
Hardware	1 control computer, 1 controller

Table of contents

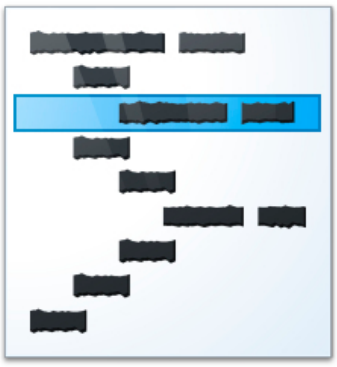
1 Introduction

You have already been introduced to the APROL process control system from B&R and know, on the one hand, how it is structured and, on the other, how the commissioning of a plant is carried out.

Before you create the logic for the plant, you will deal with the programming languages which are available in the process control system. Structured Text (ST) offers a quick and efficient way of programming in the area of automation. Structured Text is a high level programming language with conceptional elements from the Basic, Pascal, and ANSI-C programming languages.

In the best case, you have already been introduced to the simple standard structures, commands, key words, and the syntax of ST in the Automation Studio training module TM246 and seminar SEM246.1.

The Sequential Function Chart (SFC) will be explained in detail in the second part of this training module. This programming language consists of a graphic and step-oriented display and is therefore very suitable for depicting the sequential steps of a plant.



Apart from its simplicity, SFC offers many other possibilities to analyze graphically - which we will see later. We recommend that you get familiar with the Automation Studio training module TM242, in order to learn the elements and function of a Sequential Function Chart.

Have you already worked through the Automation Studio training modules for ST and SFC, or are you already with that type of programming?
You can then learn how the ST and SFC project parts are created in APROL in this training module. Apart from the creation of programs and function blocks, the different ways to debug and find errors will also be explained. The knowledge which you acquire will be consolidated with repeated examples and exercises.

1.1 Objectives

You will learn how to work with Structured Texts and Sequential Function Charts in APROL with this training module. On the one hand in APROL, ST and SFC programs are created in the context of a project, on the other, ST and SFC function blocks are created in the context of a library.

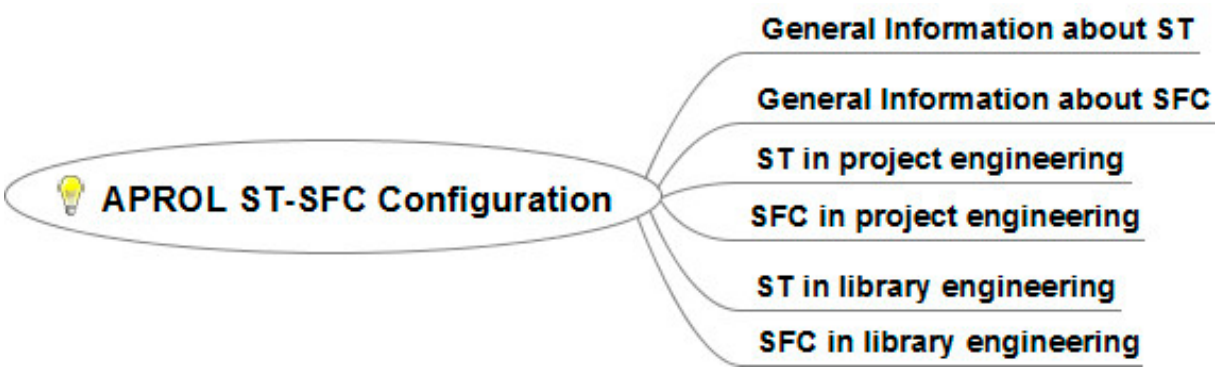
The most important and basic functions of ST and SFC will be repeated at the beginning.

Afterwards, the module is divided into two parts: The first part deals with Structured Text. There will be an explanation of how ST project parts are created, the declaration and usage of variables, and the other St editor functions.

The Sequential Function Chart will be dealt with in the second part. You will then learn how to create an SFC project part and declare and use variables. The SFC editor also provides many other functions, which we will also learn, for creating step sequences.

You will also learn how you can diagnose the ST and SFC programs. The system offers the monitor mode, the SFCViewer, and different SFC reports for this purpose.

This training documentation includes numerous screenshots which should allow you to get a much better idea of your own system. The functionality of the system can also be tested by placing tasks.



2 General Information about ST

2.1 Structured Text Features

General information

ST is a text-based high-level language for programming automation systems. Simple standard constructs enable fast and efficient programming. ST uses many traditional features of high-level languages, including variables, operators, functions and elements for controlling program flow.

ST is a programming language according to IEC[1] standardized.

Properties

Structured Text is characterized by the following features:

- High-level text language:
- Structured programming
- Easy to use standard constructs
- Fast and efficient programming
- Self-explanatory and flexible use
- Similar to PASCAL
- Conforms to the IEC 61131-3 standard

Possibilities

The following functions are supported in APROL:

- Using connectors and digital and analog inputs and outputs
- Logical operation
- Logical comparison expressions
- Arithmetic operations
- Decisions
- Step sequencers
- Loops
- Create local function blocks
- Calling functions and function blocks from the libraries
- Variables Live-Value (Debugging)

[1] The IEC 61131-3 norm is a worldwide valid norm for programming languages for programmable logic controllers.Apart from the Structured Text, the Sequential Function Chart, Ladder diagram, Instruction list, and Continuous Function Chart programming languages are also specified.

2.2 Basic Elements and Command Groups

In order to avoid repetitions, you are asked to read the exact syntax and usage of the basic elements and command groups of the Structured Texts in the Automation Studio training module TM246.

At this point, only a part of the basic data and important issues will be explained and repeated.

The assignment in ST consist of a variable on the left side which is assigned the result of a calculation on the right side by the assignment operator "**:=**". All assignments must be closed with a semicolon "**;**".

```
Result := ProcessValue * 2;   (*Result  ← (ProcessValue * 2) *)
```

Table 1: Assignment takes place from left to right

When the code line has been processed, the value of "Result" variable is twice as large as the value of "ProcessValue" variable.

Furthermore, the following command groups are distinctive in Structured Text:

- **Boolean operations**

Boolean operations can be combined in any way. The result of the expression can only be TRUE (logical 1) or FALSE (logical 0).

```
Variable_1 := Variable_2 AND NOT Variable_3;
Variable_1 := Variable_2 OR Variable_3;
Variable_1 := Variable_2 XOR Variable_3;
```

- **Arithmetic operations**

A distinct benefit of high-level programming languages is the easy management of arithmetic operations such as addition, subtraction, multiplication, etc.

```
Variable_1 := Variable_2 + Variable_3;
Variable_1 := Variable_2 - Variable_3;
Variable_1 := (Variable_2 * Variable_3)/(Variable_4 + 1);
```

- **Comparison operators and decisions**

Simple constructions are available to compare several variables. These comparison operations return the value TRUE or FALSE and are mainly used as the condition in decisions such as IF, ELSIF, WHILE, and UNTIL.

```
IF a > b THEN
    Result := 1;
ELSIF (a > b) AND (a < c) THEN
    Result := 2;
ELSE
    Result := 3;
END_IF
```

- **State machines - Case statement**

If one and the same variable is checked with an IF statement, a CASE statement is better used instead. The CASE statement compares a step variable with multiple values. If one of these comparisons is a match, the steps that compare to that step are executed. The program code in the ELSE branch will be executed if nothing applies.

```
CASE StepVariable OF
1,5:        Result := 100;
2:         Result := 500;
3,4,6..10:     Result := 1000;
ELSE
        Result := 0;
END_CASE
```

- **Loops**

Loops are attractive if parts of the code must be repeated in the same cycle. The loop code block will be repeated until a defined termination condition is met.

We differentiate between head and foot-controlled loops: The head-controlled loop (FOR, WHILE) checks its termination condition before the loop is run, whilst the foot-controlled loop (REPEAT) checks at the end of the loop and therefore runs through the code at least one time.

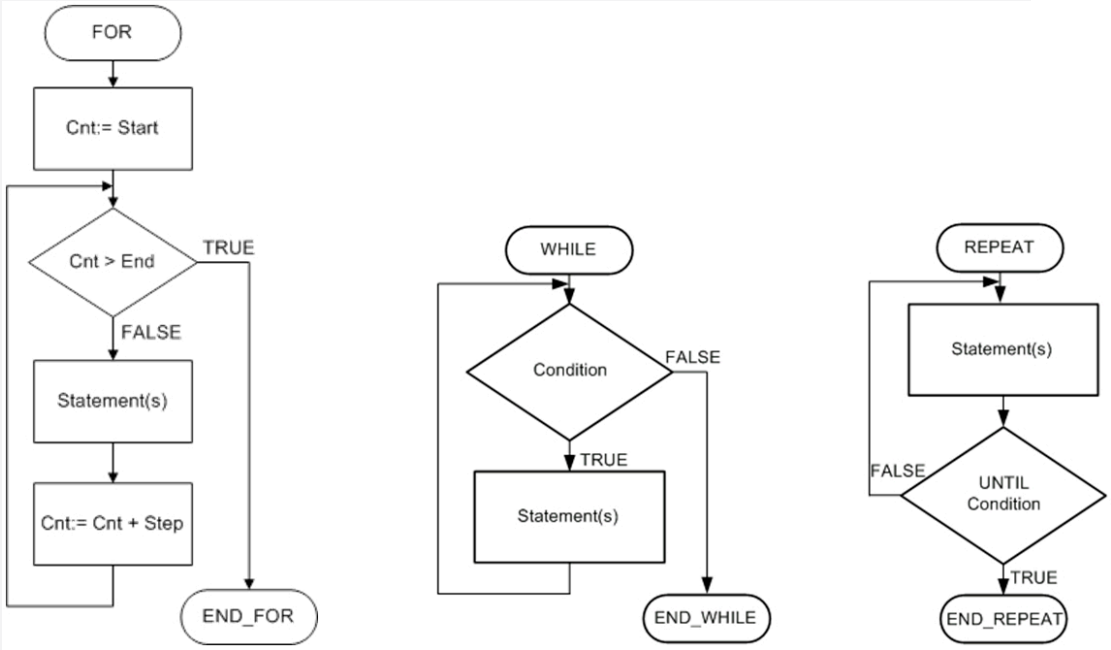


Figure 1: Different types of loops

```
//Beispiel für eine FOR-Schleife:
FOR i:= StartVal TO StopVal BY StepVal DO
    Result := Result + 1;
END_FOR
```

3 General Information about SFC

Sequential Function Chart is a visual programming language that makes it possible to coordinate the sequential execution of different program sections and steps. This programming language is used when program steps and defined transitions to the individual steps is required. It is even possible to execute parallel sequences within a single program.

Sequential Function Chart is a programming language according to IEC[2] standardized. The steps can call actions which are programmed in Structured Text.

In addition to the actions, the SFC steps and transitions are programmed in Structured Text.

Sequential Function Chart is extremely suitable for use with state machines. The possibility of simultaneous execution (parallel branches) allows complex concepts to be realized in a simple manner. The visual structure of the program is also advantageous for documentation and troubleshooting sequences.

When are SFCs used?

The programming language SFC can be implemented directly everywhere where sequences can be mirrored directly with SFC. State machines can be "redrafted" and implemented in SFC; the opposite is usually not possible.

Creating large applications

The software concept must be thought out thoroughly before large applications are put into practice, in order to get the full benefits of what SFC was conceived for - controlling sequences.

It is recommended to realize the actual sequences in the SFC program. The underlying functions of the devices, such as evaluating I/Os, scaling values, or controlling actions, should be realized in additional program modules.

The underlying functions of the devices in SFC programs are called using defined interfaces and contain the status of the function in the same interface.

Such concepts of the benefit of a change in the underlying functions of the devices (new sensor, other drive), but an unchanged sequence. The sequence chain and the underlying functions of the devices can be well tested and expanded, because of their division.

The rest of the document deals with the Sequential Function Chart (abbreviated SFC).

[2] The IEC 61131-3 norm is the only worldwide valid norm for programming languages for programmable logic controllers.Apart from the Sequential Function Chart, the Structured Text, Ladder diagram, Instruction list, and Continuous Function Chart programming languages are also specified.

3.1 Basic functions of the SFC

The basic components of SFC are called steps and transitions. Steps are shown as rectangles with different types of borders. The transitions are displayed as bars, the steps as crosses.

All steps and transitions are connected to each other in alternating order. In other words, a transition always follows a step. A step and its transition form a unit. It is not possible, for example, for two steps or two transitions to occur consecutively.

The example shown here illustrates a program with three steps, the associated transitions and a jump.

The "FIRST" step is run when an SFC program is called for the first time. This is marked with a double frame and is always called when the program is initialized. The associated transition has the value TRUE, which means that the initialization step will only be called for one cycle.

The programs stays in the "HEAT" step until the "tooHot" transition is true. The "HEAT" step then becomes inactive, and the program moves on to the "COOL" step. As soon as the "tooCool" transition becomes TRUE, then the jump is executed and the "HEAT" step becomes active once more.

Only one step is executed per program cycle.

Step names are symbolic and don't need to be declared. The transitions are variables of the type BOOL or logical expressions which can be composed in various programming languages.

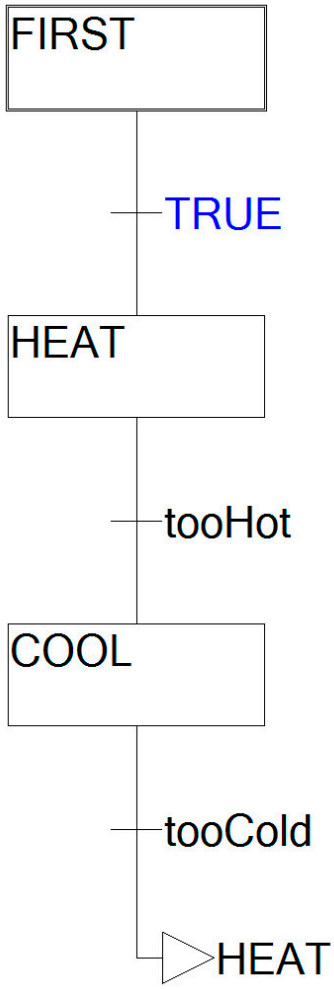


Figure 2: Sequence with two steps, one jump and an initialization step

A step and a transition from a single unit. A step is followed by a transition. It is not possible for two or more transitions or steps to appear consecutively.

If there is a transition from one step to the next, the new step will be executed in the next program cycle.

SFC steps

A step is either active or inactive. It represents a situation in which commands or actions are processed as long as it is active. When the step is switched to inactive, the following transition makes a decision by evaluating the transition condition.

Steps are represented as rectangles. If a step includes cyclic program code, then a black triangle is shown in its upper right corner. Each step can have an optional entry and exit action. An entry action (E) is called once when the step becomes active. An exit action (X) is called once after the step becomes inactive.
A time monitoring can also be configured for the steps.

The step content is composed of actions which are programmed in ST.

SFC transitions

Each step is activated or deactivated via transitions. If a transition is active (TRUE, logical "1"), the corresponding step will be deactivated and the following step will be activated in the next cycle. If the transition has the value FALSE, logical "0", it is then inactive and there will be no further switching in the course of the program.

Each transition has a transition condition allocated to it, i.e. a calculation rule that delivers a Boolean value. In APROL, a transition condition can only be programmed in the IEC language 'ST'.

SFC actions

Actions are assigned to a step. As soon as the step is active, the corresponding actions become active and behave according to their configured action qualifiers.

The content of an action is also programmed in the IEC language 'ST'.

Important action qualifiers in overview:

- Cyclic action call, as long as the step is active (N).
- Time limitation of an action (L).
- Time delay of an action (D).
- Conditional setting of an action, over and above several steps (S).
- Resetting a previously set action (R).

In SFC, individual steps don't always have to be executed one right after the other. Parallel and alternative branches can be inserted for more flexibility.

Parallel branch (simultaneous chain)

Several steps which are arranged in parallel and connected with a double line can follow after a transition. The parallel steps are connected together again after the last transition with double horizontal lines.

All of the simultaneous paths in a parallel branch (simultaneous chain) are processed at the same time (i.e. parallel) after the previous transition has switched. The transition after a simultaneous chain is only evaluated when the last step of all parallel paths are active at the same time.

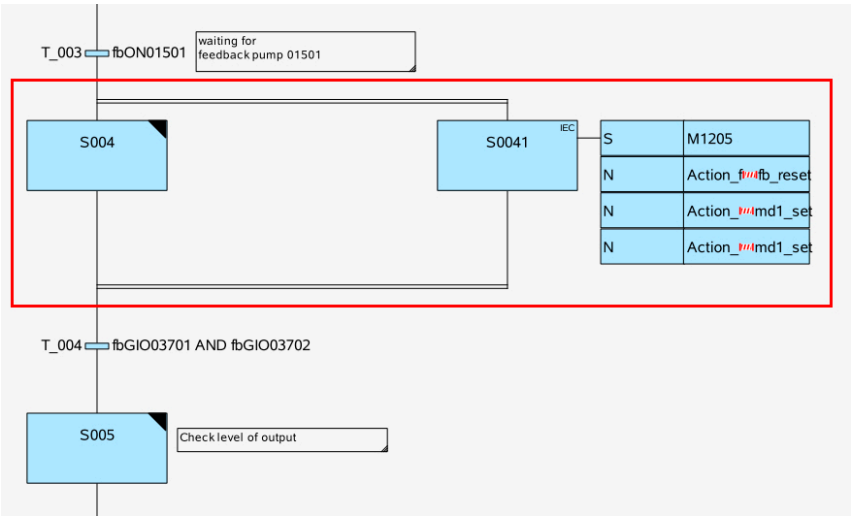


Figure 3: beginning of simultaneous sequences

Alternative branch

One or more alternative branches are necessary in SFC if a sequence has to be divided up into several different paths. In the alternative branch, each step becomes active with its associated transition. The path is thus divided from the rest/alternatives. If several transitions are active at the same time, the evaluation order of the transitions takes place from left to right, i.e. the more to the left, the higher the priority.

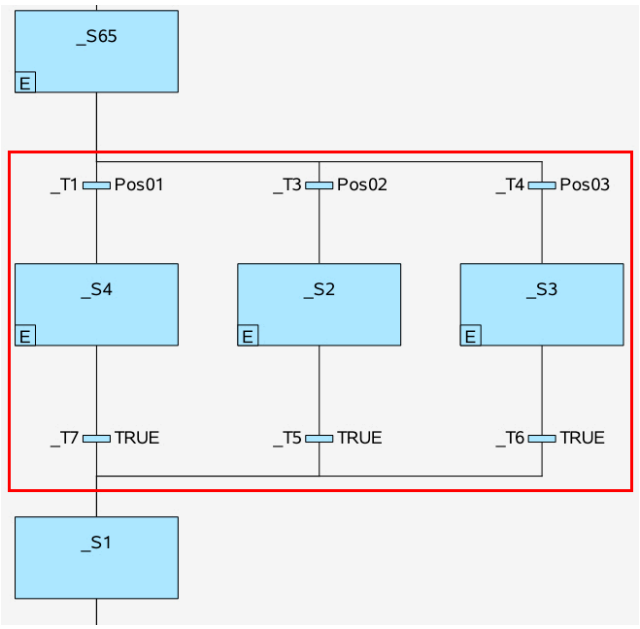


Figure 4: Alternative branch

The alternative branch can return to one mutual path at its end or end in a jump.

Jumps

Step chains obtain a flexible construction by using jumps. A jump is also activated via a transition. Any step in the SFC network can be jumped to. It is only possible to jump to a step - It is not possible to jump to transitions.

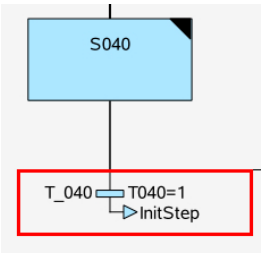


Figure 5: Jump

3.2 Difference between simple steps and IEC steps

We already know that a step can contain executable program code. This code is then executed once when the step becomes active. There are two types of steps; the "simple" and "IEC" steps. The differences are explained in the following table.

Simple step	IEC steps
Needs less physical memory on the controller	Conforms to the IEC Norm
3 actions possible per step (Entry, Cyclic, and Exit action)	Up to 32 actions possible per step (plus entry and exit action)
Actions behave like actions assigned to IEC steps, with the "N" action qualifier	Actions can have different action qualifiers assigned to them
Actions can only be used in one step	Existing actions can also be assigned several times to different steps.

Table 2: Different types of SFC steps

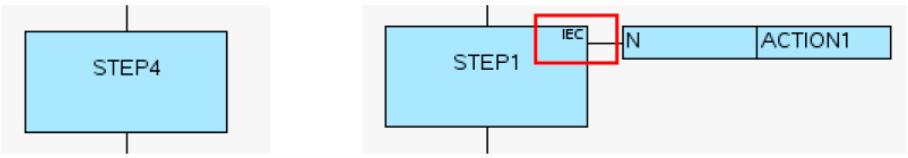


Figure 6: Comparison of simple steps (left) and IEC steps (right)

The SFC editor shows if a new step is a simple or IEC step.

Simple steps which have already been placed can also be turned into IEC steps, and vice versa.

3.3 What are the special features of SFC in APROL?

The 'ST' and 'SFC' project parts are supported by the APROL library concept for structured programming.

The following schematic overview explains the possibilities of using the library blocks (CAE library) in the 'CFC', 'ST', and 'SFC' project parts (CAE project).

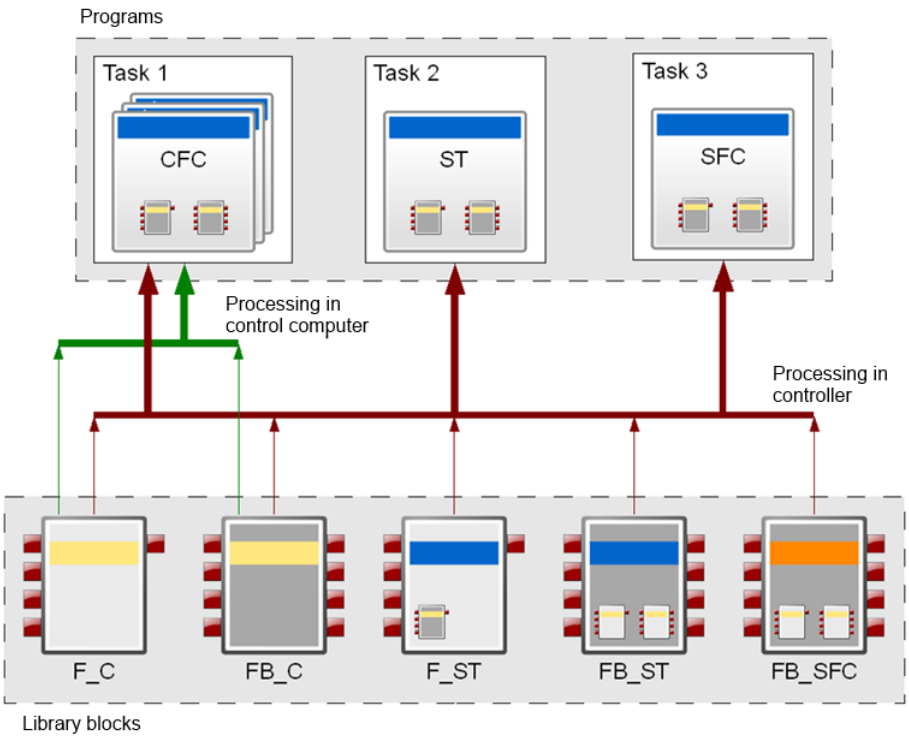


Figure 7: Schematic display of the possibilities of usage

Key: 'F' - Function, 'FB' - Function block, 'C' - Programming language 'C'

- SFC and ST programs can basically only be used on a controller.
- A controller task is only allowed to have one SFC or ST program allocated to it. This is **not allowed to be the default task of the controller**.
- Several CFC programs are allowed to be allocated to a controller task.
- Any amount of function blocks (ST, SFC, or C) can be used in an ST, SFC, or CFC program.

Decision: 'SFC program' or 'SFC function block'

An SFC program is used when a sequential control that intervenes in several parts of the production system is required **only one time**.

An SFC function block is used when a sequential control is required several times. Changes in the SFC function block are carried out centrally in the CAE library and are transferred automatically to all instances.

3.4 How does SFC differ to the other IEC languages?

The IEC programming languages (EN 61131-3:2003)

Programming language	Description
Function block language	Named 'Continuous Function Chart' (CFC) in APROL. Contains logic circuit diagrams
Structured Text	Named 'Structured Text' (ST) in APROL. Can be compared with high level programming languages.
Instruction List	The 'Instruction List' (IL) programming language can be compared to assembler.
Ladder Diagram	The 'Ladder diagram' (LD) programming language can be compared to circuit diagram.
Sequential Function Chart (SFC)	In comparison to the other IEC languages, the 'Sequential Function Chart' (SFC) programming language in APROL is **especially** suited for modeling and programming **complex, sequential processes** and is similar to a state diagram.

Additional benefits of the 'SFC' programming language are:

- simple structuring of processes which are based on a sequence of steps
- Suitability for time and event orientated operations
- graphic configuration of the SFC network
- Modularization / reusability by use of function blocks
- easy to learn
- easy to maintain
- easy to expand

4 ST in project engineering

4.1 Introduction and editor functions

As explained in the previous chapter, on the one side, ST programs are created in the project, and on the other side, ST function blocks and ST functions are created in the library.

We will start with the creation of an ST program in the context of the project.

The ST editor is a text editor which has many additional functions. Commands and keywords are shown in color.

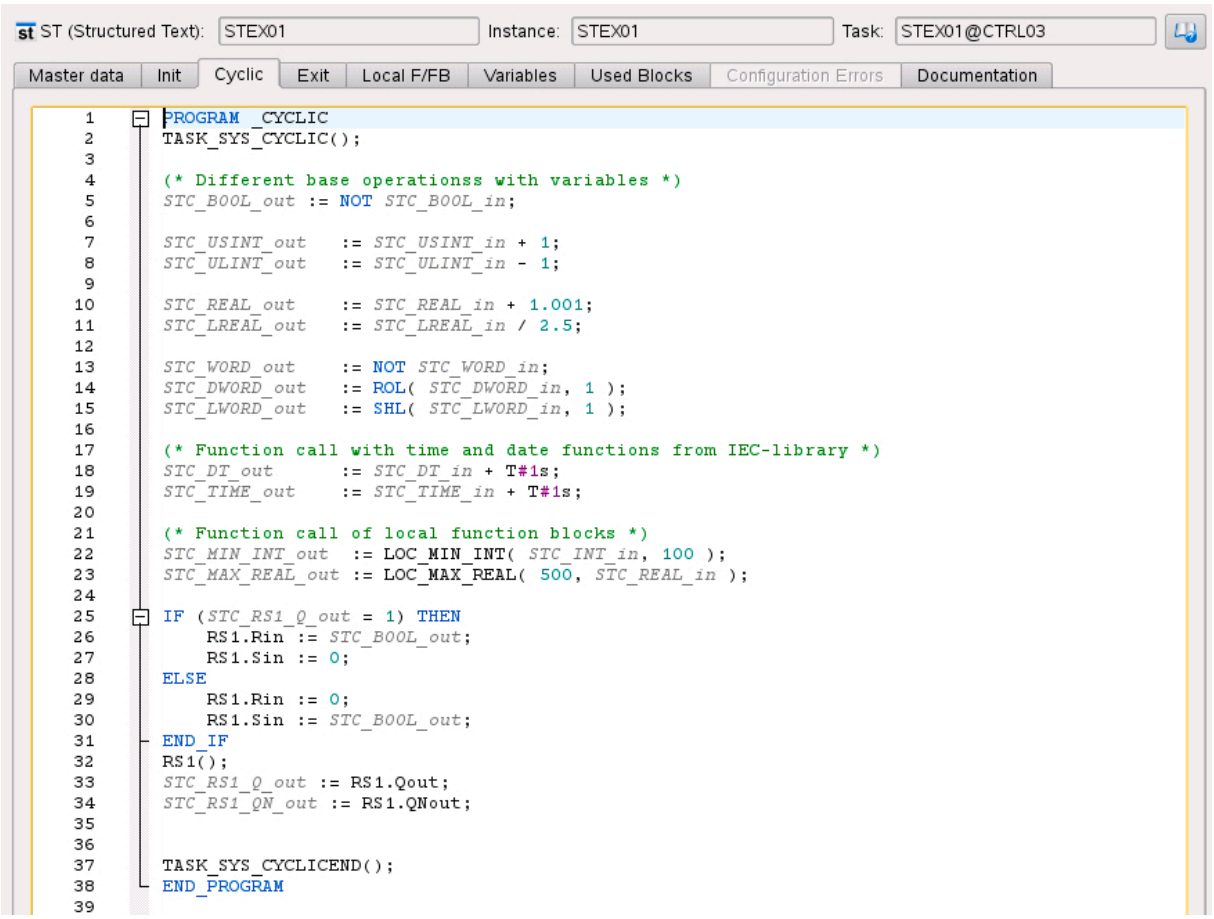


Figure 8: The APROL ST editor

The editor offers the following functions and properties

- Distinction between uppercase and lowercase letters (case sensitive)
- Possibility of defining local functions and function blocks
- Overview of all variables used in the code
- Automatic completion of variables
- Find / Replace
- Undo / Redo
- Import of ST code
- Integrated syntax check in the context menu

- Possibility to use functions and function blocks from other APROL libraries
- Identification of corresponding pairs of parentheses
- Expanding and collapsing constructs (outlining)
- Line modified marker

4.2 Creation of ST programs

We will begin with the creation of an ST program in the project.

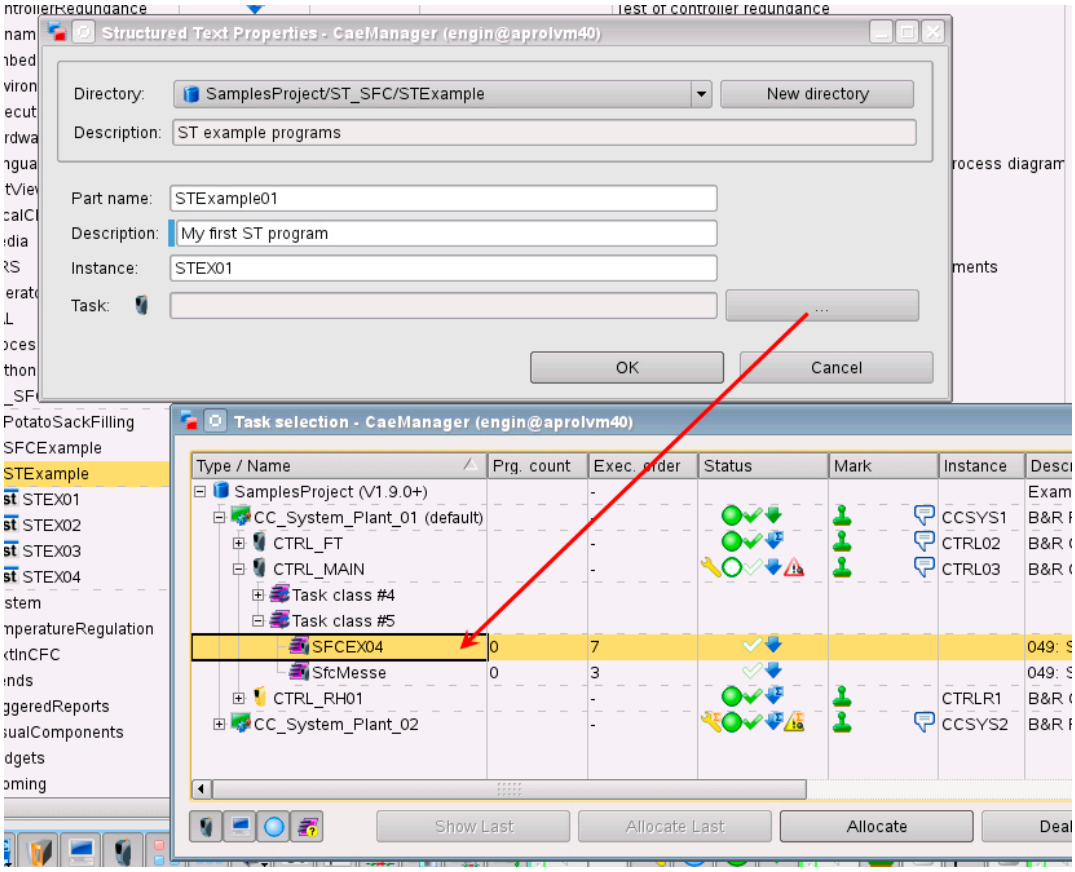


Figure 9: Creation and task allocation in the ST program

Navigation: CaeManager / CAE project / File / New / ST (Structured Text)	
Name	<Program name>
Instance	<Instance name>
task	Select task with [...]
Confirm with [**OK**]	

A task must not be assigned exactly at this very moment. This can also be done later. However, the ST program cannot be activated without a task allocation.

Apart from that, a separate controller task must be created for each ST program. This task is neither allowed to be the default task nor contain any other project parts!

The project part can be compiled after the ST program has been created, as well as a build of the task and a download to the hardware.

With this, the St program configuration is complete.

4.3 Working with the ST editor

4.3.1 The most important tabs in the ST editor

'Master data' tab

The general SFC program settings can be seen in this tab. The project part's instance can also be adjusted here.

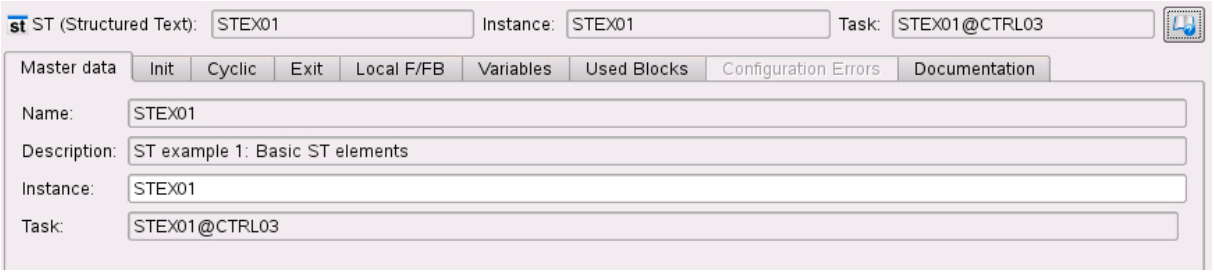


Figure 10: Master data in the ST editor

Init and exit procedures

The **'Init' tab** contains a text editor. This is for editing the initialization code using the 'Structured text' programming language. The code is executed in the Init-UP of the task and can be used to initialize control functions and variables.

The **'Exit' tab** also contains a text editor. This is for editing the de-initialization code - and also written in ST. The exit code is executed in the Exit-UP of the task and can be used, for example, to de-initialize control functions and variables.

The program code in the init and exits tabs is only processed/executed once.

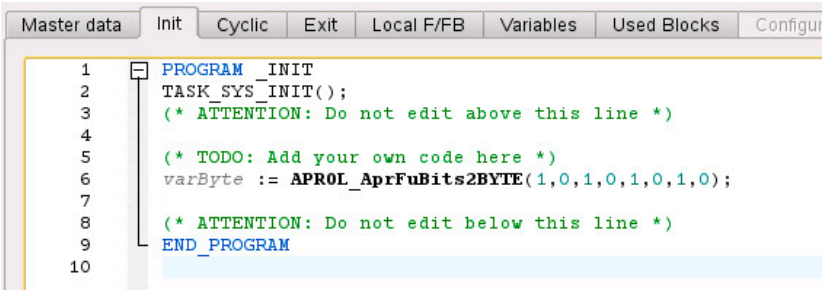


Figure 11: Program code in the init

The cyclic code part

The cyclic program code of the ST program is in the **'Cyclic' tab**. The code there is processed cyclically on the controller.

All elements, expressions, and command groups can be programmed in this tab.

Master data | Init | Cyclic | Exit | Local F/FB | Variables | Used Blocks | Configuration Errors | Documentation

```
1  PROGRAM _CYCLIC
2  TASK_SYS_CYCLIC();
3  (* ATTENTION: Do not edit above this line *)
4
5  (* C-function *)
6  varByte := APROL_AprFuBits2BYTE(1,0,1,0,1,0,1,0);
7
8  (* C-function block *)
9  Bits1_v.IN := varByte;
10 APROL_AprFbBYTE2Bits(Bits1_v, Bits1_r);
11 bool1 := Bits1_v.BIT00;
12 bool2 := Bits1_v.BIT01;
13 bool3 := Bits1_v.BIT02;
14 bool4 := Bits1_v.BIT03;
15 bool5 := Bits1_v.BIT04;
16 bool6 := Bits1_v.BIT05;
17 bool7 := Bits1_v.BIT06;
18 bool8 := Bits1_v.BIT07;
19
20 (* ST-function *)
21 varReal := 500.452;
22 varReal := DEMO_ST_LinPoly3(X:=varReal, A0:=3, A1:=2.5, A2:=0, A3:=1.45E-6);
23
24 (* ST-function block *)
25 Pid.dir := 0;
26 Pid.K := 18;
27 Pid.TT := 0.5;
28 Pid.T_N := 1;
29 Pid.T_V := 2;
30 Pid.valmin := 0;
31 Pid.valmax := 100;
32 Pid.W := 50;
33 Pid.X := varReal;
34 Pid();
35 varReal := Pid.U;
36
37 Pid(X:=varReal);
38 varReal := Pid.U;
39
40
41 (* ATTENTION: Do not edit below this line *)
42 TASK_SYS_CYCLICEND();
43 END_PROGRAM
```

Figure 12: Cyclic part of the ST program

You can check the code which you have written by using the "Check Syntax" item in the context menu in the program editor.

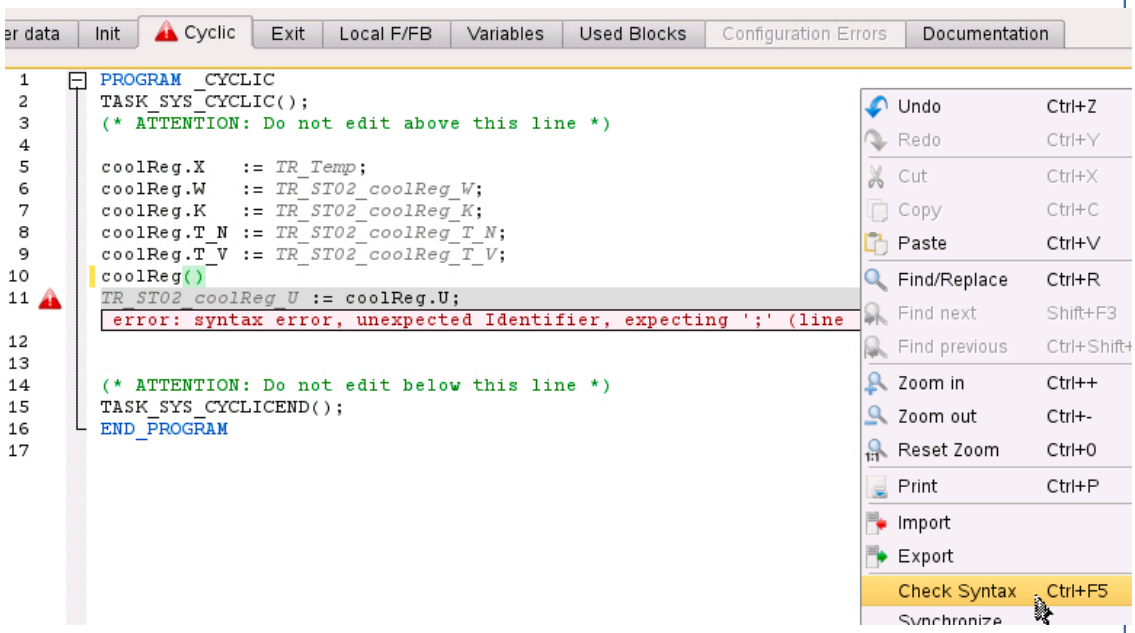


Figure 13: Syntax check in the context menu

Use of local and project variables

All variables which are used in the program code (init, cyclic, exit tabs) are created automatically during the process of saving or the manual synchronization in the **'Variables' tab**. They are listed here and are furnished with other properties.

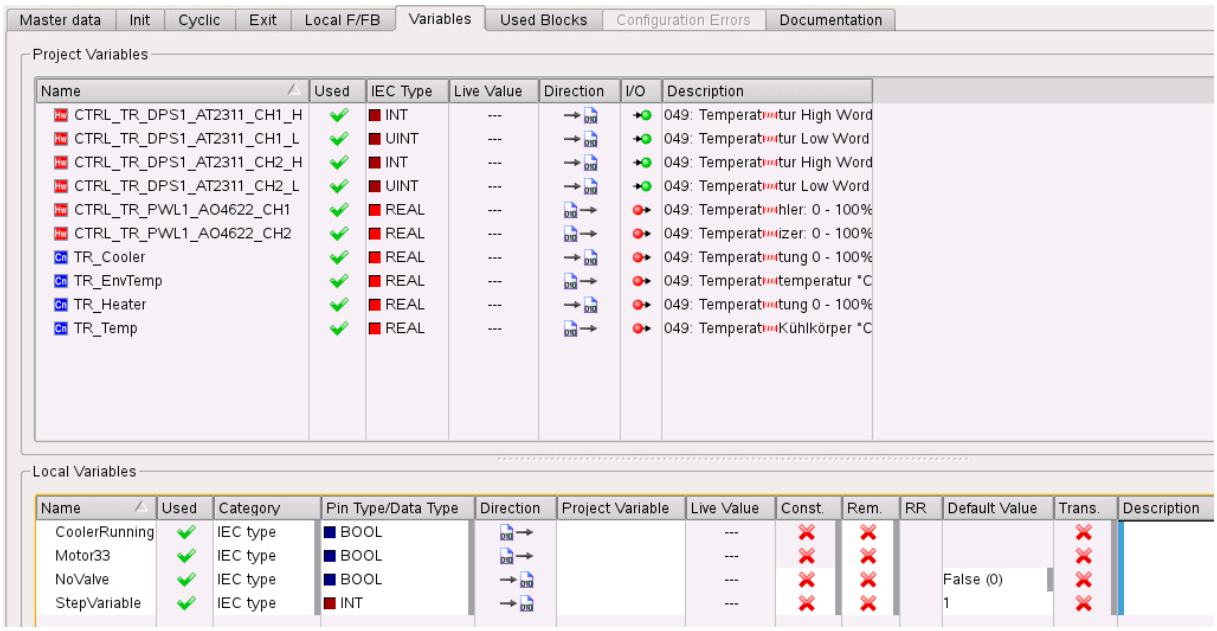


Figure 14: Variable list in the ST editor

If an ST program with many local variables has been created and some of them are indeed intended to be project variables (e.g. connectors), the name of the project variable can be entered in the corresponding "Project variable" column. A local variable is linked to a project variable in this way and the value is synchronized.

If hardware variables such as digital inputs and outputs should be written in the ST code, a **.ha** must be added to the end of the variable name.

```
DigitalOutput1.ha := DigitalInput1.ha AND (AnalogInput3.ha > 10.0);
```

A manual synchronization is possible at any time with the **'Synchronize'** context menu. All of the editor's tabs are synchronized with each other and new variable lists are eventually added, or the usage is updated (see green tick, second column in the variable list).

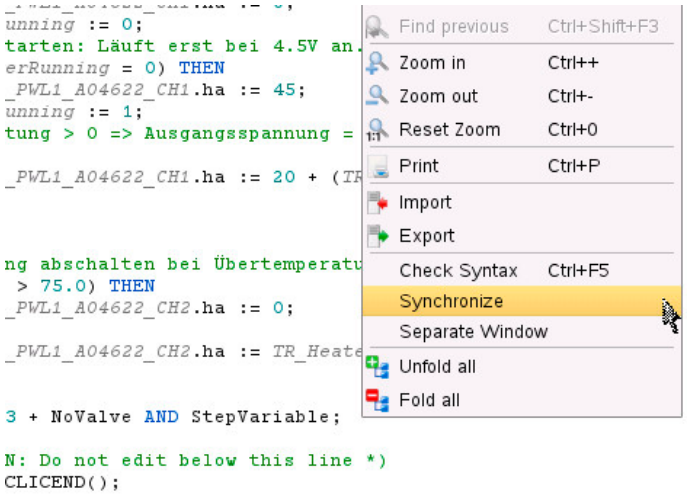


Figure 15: Synchronize in the context menu

Task: Fill level check

The level in a container should be monitored for three areas: low, high and OK.
Use an output for each of the low, ok, and high levels.

The level of liquid in the tank is read as an analog value (0 - 32767) and is converted internally to a percentage value (0-100%).

A warning tone should be given if the contents fall below 25 %.

Solve this application by using the CASE statement in an ST program.

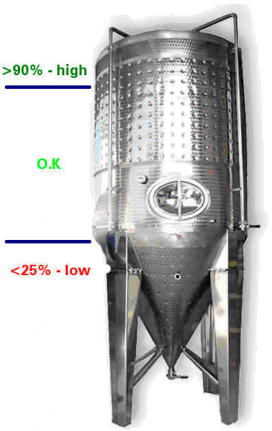


Figure 16: Evaluating the fill level of a container

Declaration:

```
VAR
     aiLevel : INT;
     PercentLevel : UINT;
     doLow : BOOL;
     doOk : BOOL;
     doHigh : BOOL;
     doAlarm : BOOL;
END_VAR
```

Table 3: Suggestion for the variable declaration

4.3.3 Local Functions and Function Blocks

You will learn how functions and function blocks are created in the ST editor, and how they are used in the code, in this chapter. They can only be used in the ST programs.

These functions and function blocks are programmed and instanced in the **'Local F/FB' tab**.

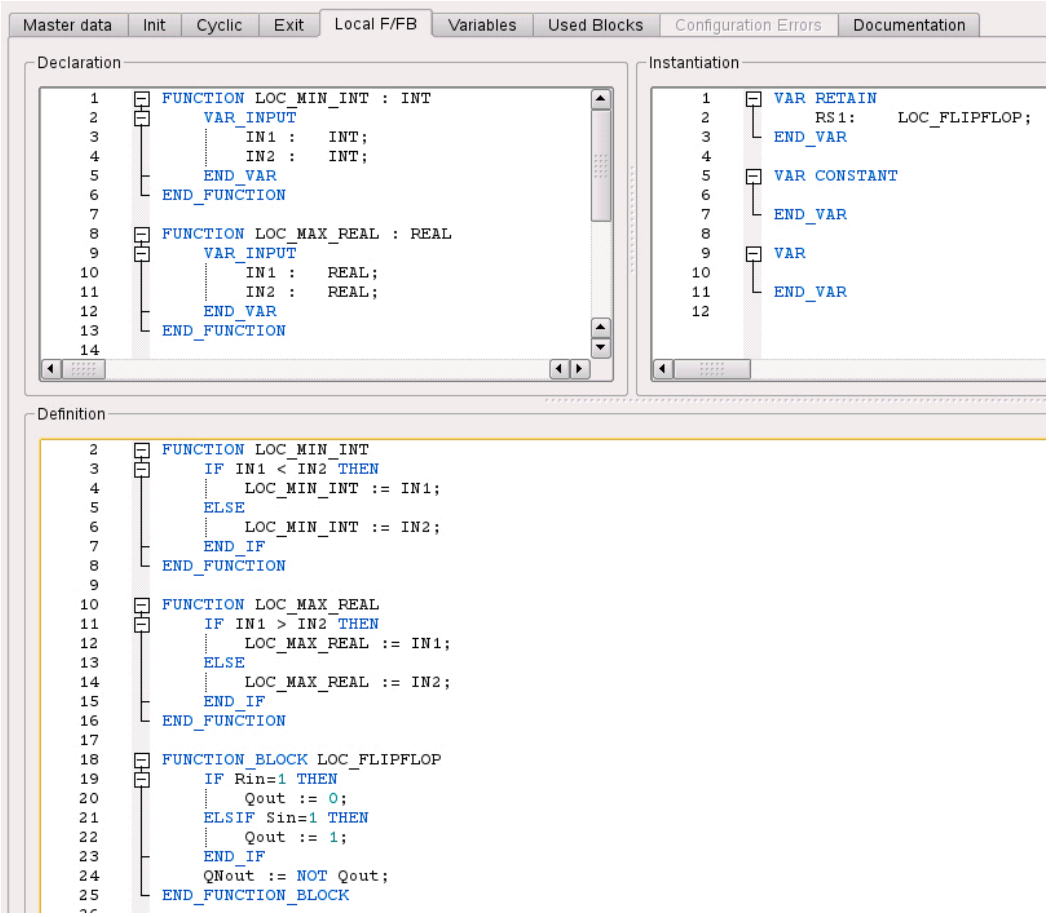


Figure 17: Example of local functions / function blocks

They are created as follows:

1 Firstly, the function or function block is declared in the left upper area. That means, the variables are defined here.

```
FUNCTION LOC_MINIMUM_INT : INT
    VAR_INPUT
        IN1 : INT;
        IN2 : INT;
    END_VAR
END_FUNCTION

FUNCTION_BLOCK LOC_FLIPFLOP
    VAR_INPUT
        R_IN : BOOL;
        S_IN : BOOL;
    END_VAR
    VAR_OUTPUT
        Q_OUT  : BOOL;
        QN_OUT : BOOL;
```

```
    END_VAR
END_FUNCTION_BLOCK
```

2 The goal of the function (block) is then defined. In our example, we define a "LOC_MINIMUM_INT" function to deliver the smaller value of two input variables, and a function block of the type flip-flop "LOC_FLIPFLOP".

```
FUNCTION LOC_MINIMUM_INT
    IF IN1 < IN2 THEN
        LOC_MINIMUM_INT := IN1;
    ELSE
        LOC_MINIMUM_INT := IN2;
    END_IF
END_FUNCTION


FUNCTION_BLOCK LOC_FLIPFLOP
    IF R_IN = 1 THEN
        Q_OUT := 0;
    ELSIF S_IN = 1 THEN
        Q_OUT := 1;
    END_IF
    QN_OUT := NOT Q_OUT;
END_FUNCTION_BLOCK
```

3 The function block must be placed and declared as volatile or remanent before we can use it in cyclic code. A variable with which the function block is called is defined while the instance is made. A function block can be instanced several times.

```
VAR RETAIN
    RS1 : LOC_FLIPFLOP
END_VAR
```

RS1 is the variable for "addressing" our function block. We have defined it as "retain" , meaning that all outputs are remanent.

Functions do not have to be instanced and can be used directly in the program code.

4 We can then use our functions and function blocks in the program code (Init, cyclic, or exit tab) as follows:

```
AnalogValue_Min := LOC_MINIMUM_INT( AnalogVal_01, AnalogVal_02 );

//Declare inputs
RS1.R_IN := DigitalValue_10;
RS1.S_IN := DigitalValue_11;

RS1();    //Call function block

DigitalValue_12 := RS1.Q_OUT;    //Read-back output value
```

4.3.4 Calling existing CAE blocks in ST

If there are many functionalities in the CAE library blocks, you do not need to program them from scratch, but can use them directly in your ST program.

All function blocks that are used in the context of the ST program are listed in the **'Used Blocks' tab**.

Function blocks can be placed in the table from the context of the navigation window via drag & drop.

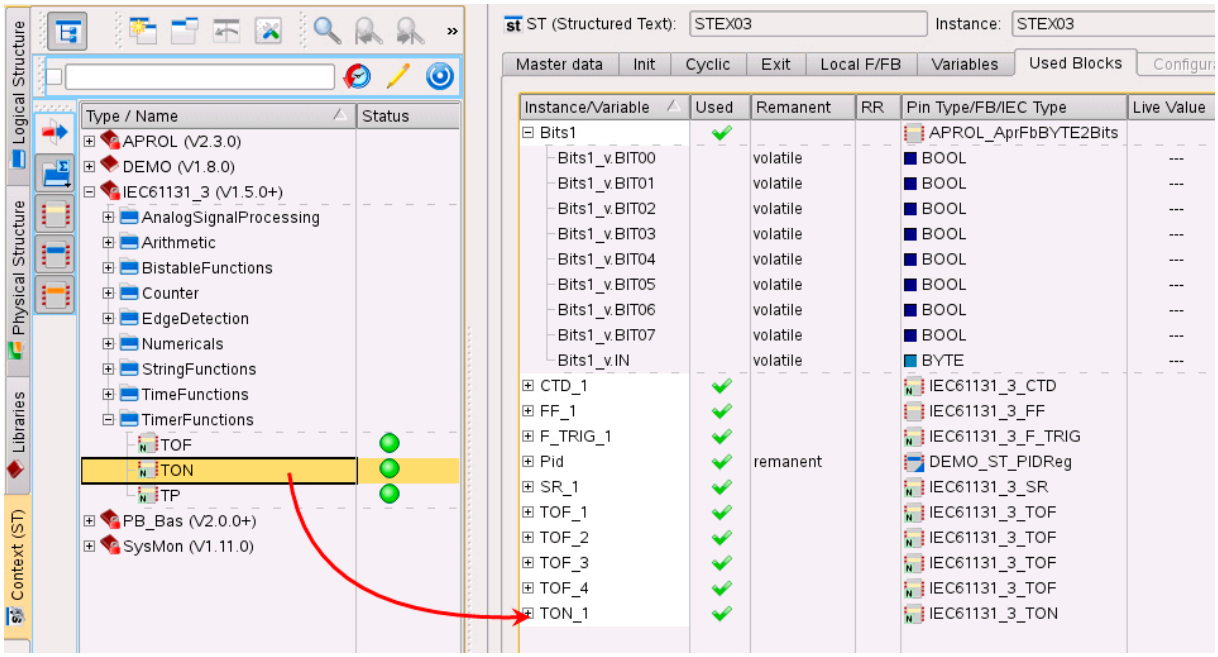


Figure 18: Used CAE blocks view

Each function block is listed with its inputs and outputs in table form. The display has a tree structure for a clear listing of the individual function blocks and their inputs and outputs, whereby each function block is represented by a root element, and the respective inputs and outputs are displayed as sub-elements.

Column	
Instance / Variable	Context function block: Instance name, with which the used function block must be addressed. Instance names can be edited. Context input / output: Name of the input or output.
Status of usage	Shows if the block is used in the context of the SFC program after synchronization takes place.
Remanent	Tells if the input / output (C block) or the block (ST / SFC block) is re-manent.
Pin type /FB / IEC type	**Context input / output:** IEC type or pin type of the I/O. **Context function block:** Name of the function block as it must be used in the code.
Live value	Value of the process variable in the debugging mode.
I/O constant	An I/O constant can be entered here instead of the default value from the library.
I/O	Context I/O: Specifies if it is an input or output.
Block description	Description from the CAE library.
Instance description	Description, which can be entered by the user, of the definite instance.

It can be used in the ST code after the function block is placed in the table via drag-and-drop and its inputs receive any necessary default values.

Functions from the CAE libraries must not be placed in the table, but directly in the ST code in the form <library name>_<block name>.

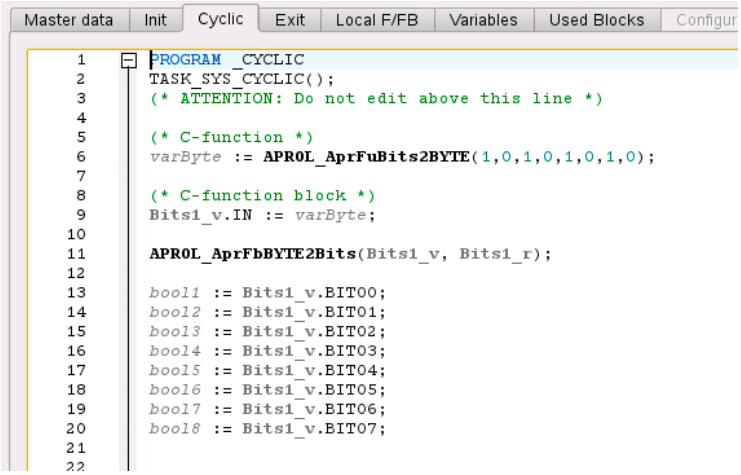


Figure 19: Calling CAE functions and blocks

Task: Using a function block from a library

Extend the previous "Fill level check" exercise with a timer block TON (from the IEC61131_3 APROL CAE library).

You could, for example, wait a certain delay time before triggering the warning horn if the container content was to fall below 25%.

4.4 Debugging ST programs

4.4.1 Live variable values

An ST program is switched to the 'live' debug mode in order to visualize and check the variable values.

Figure 20: Switch on online debugging

The debug mode can be switched on and off via the CaeManager toolbar. After it is switched on, the "Live value" column in the variable overview shows the current value of the variable.

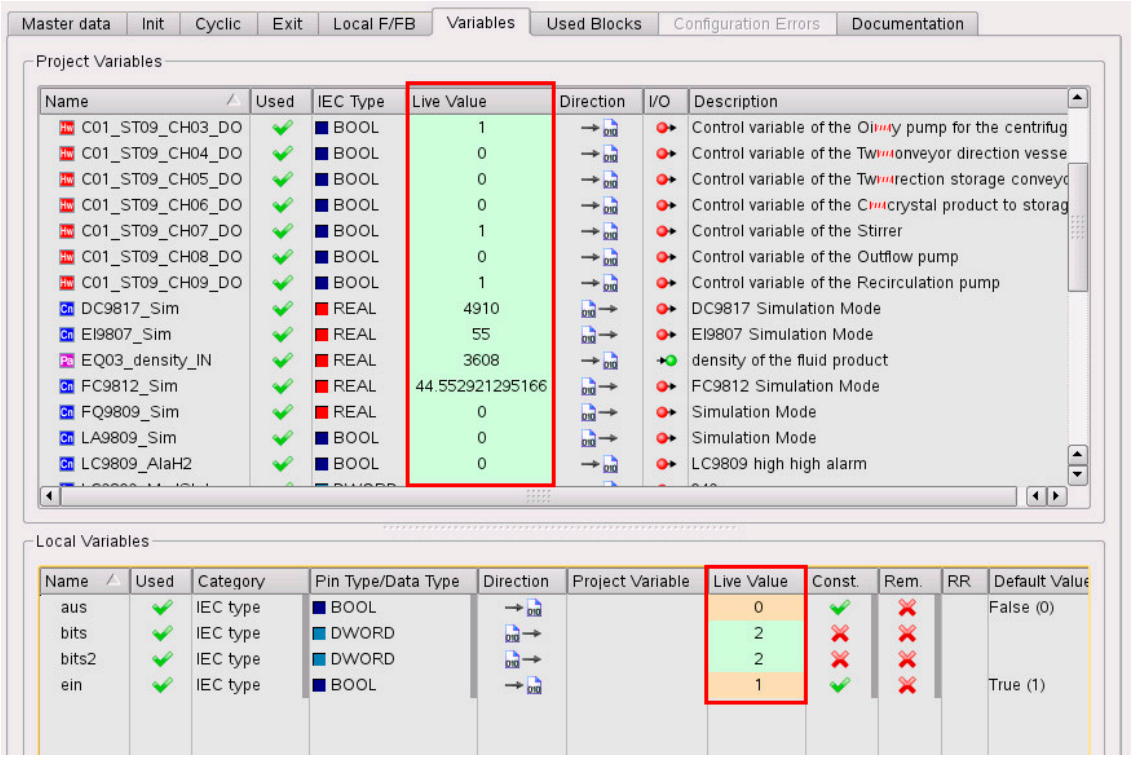


Figure 21: Variable list with live values

The "Live value" column is also in the list of used blocks, and shows the current value of function block inputs and outputs.

4.4.2 Variable Watch in the ControllerManager

Another way to debug the ST program is to use the variable Watch in the ControllerManager. All of the variables in an ST program together with their values can be shown with the help of the Watch window, and the values can be changed if required, i.e. forced.

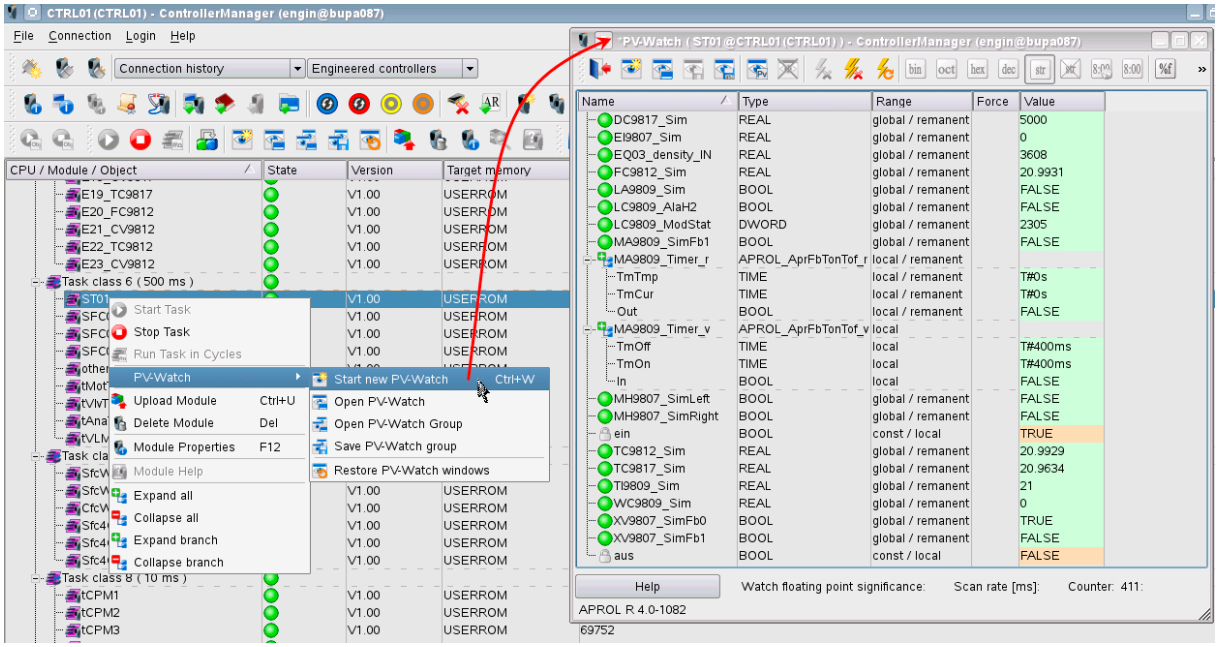


Figure 22: Watch window in the ControllerManager

Open the ControllerManager in order to get to the Watch window. Select the task of the ST program and then select **"Create New PV Watch"** in the context menu. One can drag individual or all variables into this window and see their values. The variable values can be set /forced here, according to their data type.

5 SFC in project engineering

5.1 Introduction and editor functions

When creating an SFC, we can also differentiate between an SFC program in the project and an SFC function block in a library.

Steps, transitions, jumps, and actions are inserted into an SFC program.

The SFC editor in APROL can be used in many ways:

- Toolbar: Create
- Shortcut menu for the SFC program, steps and transitions

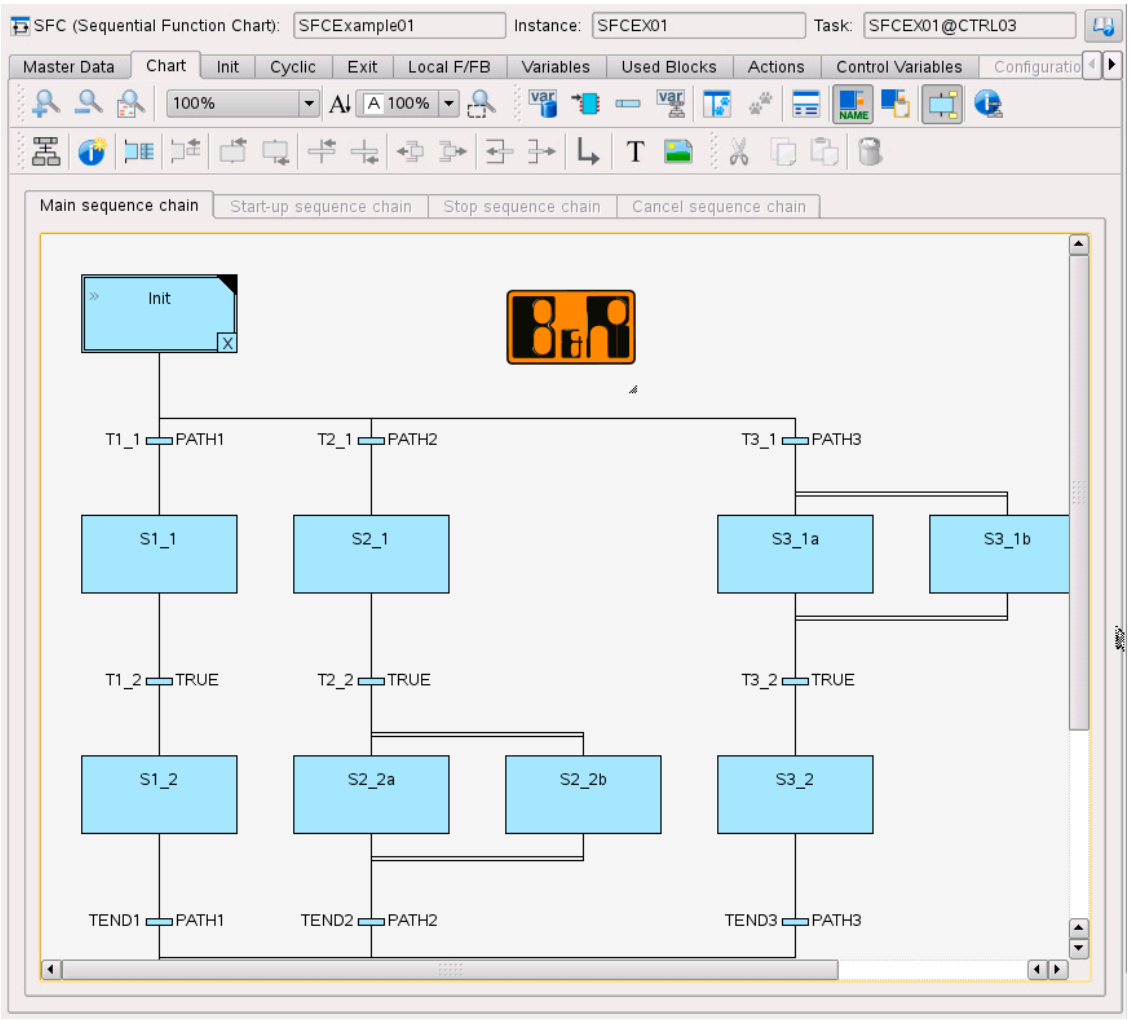


Figure 23: The SFC editor

Objects are inserted using the menu bar and context menus. Properties can be specified in the object's properties dialog. A double-click opens this dialog for the marked element.

The zoom toolbar or the keyboard shortcuts can be used to zoom in the SFC editor.

The SFC editor also offers, like the ST editor, functions for automatic completion, differentiation between capital and small letters, or the use of local functions and function blocks.

5.2 Creation of an SFC program

The following explains how an SFC program is created in the project

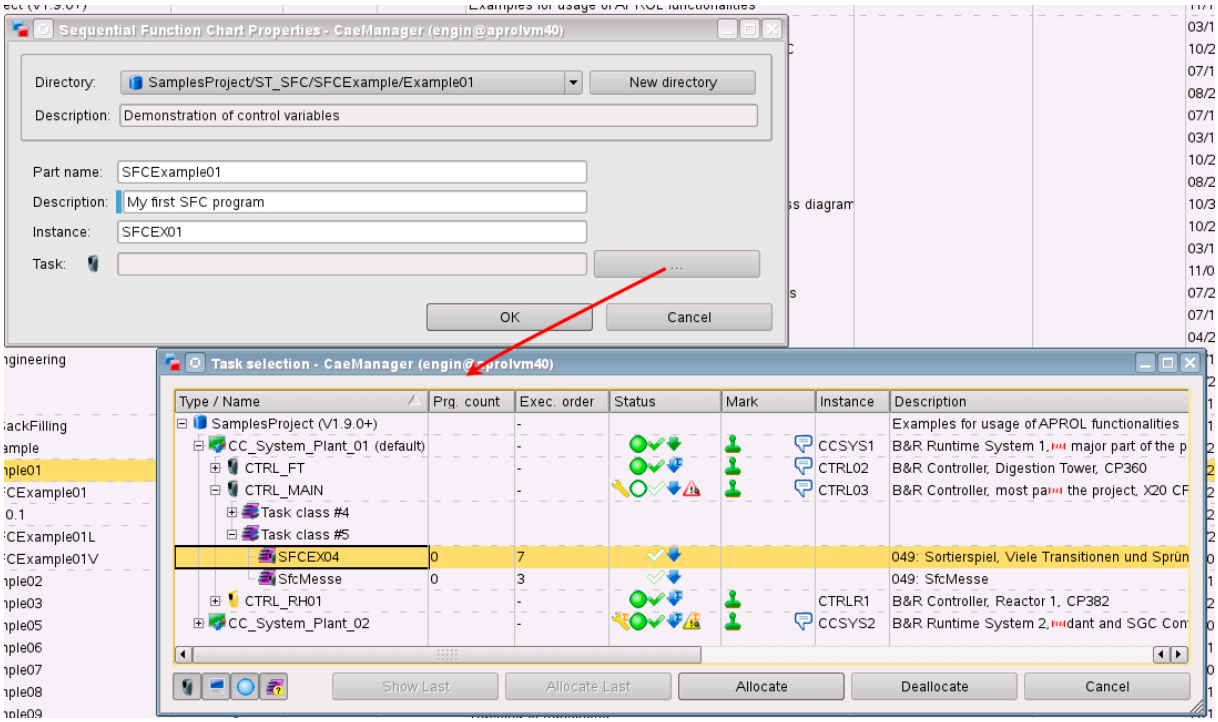


Figure 24: Creation and task allocation in the SFC program

Navigation: CaeManager / CAE project / File / New / SFC (Sequential Function Chart)	
Name	<Program name>
Instance	<Instance name>
task	Select task with [...]
Confirm with [**OK**]	

A task must not be assigned exactly at this very moment. This can be done later, however, the SFC program cannot be activated without a task allocation.

Apart from that, a separate controller task must be created for each SFC program. This task is neither allowed to be the default task nor contain any other project parts!

Further information about the allocation of a task can be found in chapter What are the special features of SFC in APROL?

The project part can be compiled after the SFC program has been created, as well as the task creation and a download to the hardware.

With this, the SFC program configuration is complete.

5.3 Working with the SFC editor

5.3.1 The editor tabs

As you can see after creating or opening an SFC program, many tabs are identical to those of the ST editor.

The following SFC editor tabs are identical to those in the ST editor. For further information, see The most important tabs in the ST editor**:**

- Init
- exit
- Local F/FB
- Variables
- Used Blocks
- Configuration error.
- Documentation

The additional tabs and their functionality are explained in the following:

The "Master data":

The general SFC program settings are made in this tab.

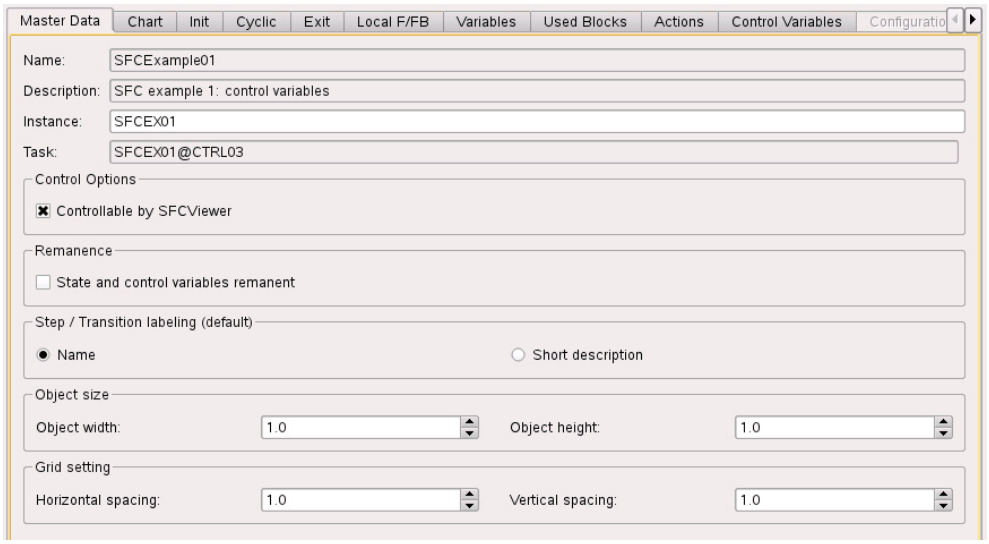


Figure 25: The master data of an SFC program

If the **'Controllable by SFCViewer'** option is activated, the processing of the SFC program can be controlled manually by the operator (e.g. pausing an SFC, forcing certain steps and actions) in the SFCViewer.

When the **'Status and control variables remanent'** option is activated, all status variables of steps, transitions, and actions as well as SFCViewer control variables (as long as the 'Controllable by SFCViewer' option is set), which are created automatically by the project part, are remanent.

Different options for the layout/display of SFC objects can be set underneath.

The 'Chart' tab

This is the most important tab - The graphical configuration of the SFC program is carried out here.

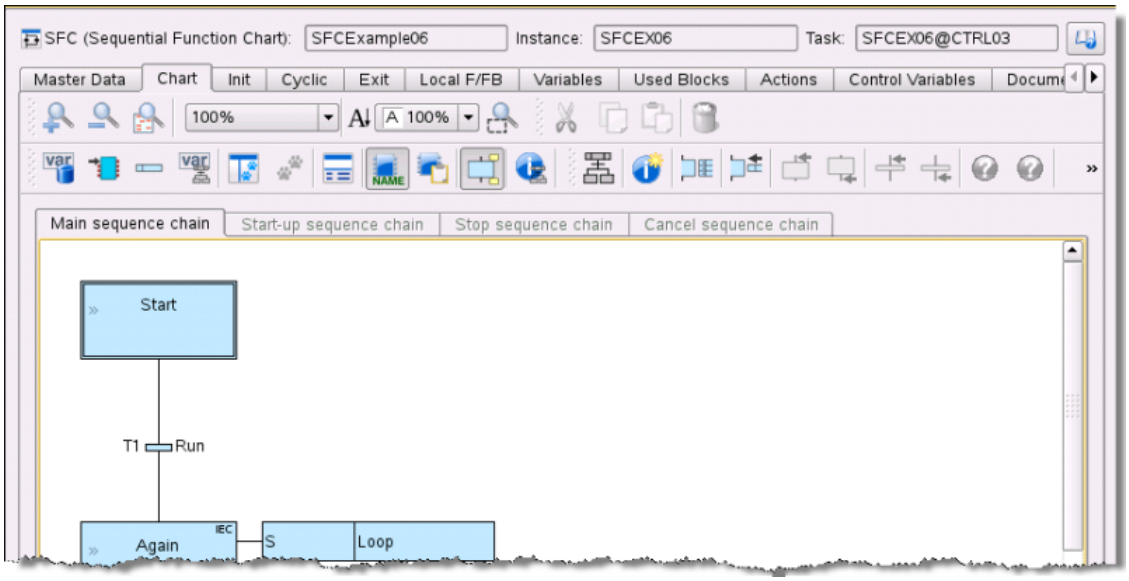


Figure 26: The SFC chart

Multiple-selection via keyboard and mouse:

Several steps or transitions can be selected in the SFC editor and SFCViewer with the mouse or the [Ctrl] and/or [Shift] keys, and the properties of these elements can then be shown via the **'Properties'** context menu. The advantage of this type of selection is that the mutual settings can be edited in one go.

Several beginnings of sequence selections can also be selected and moved via the 'Increase / decrease alternative path priority' context menu in the SFC editor by using the [Ctrl] and / or [**Shift**] keys. The first transition in each existing path must be selected for this purpose.

Placing images in the SFC:

Images can now be placed from the image container to anywhere in an SFC in order to improve the clarity and for a better overview. The images can be scaled as desired. The images can be placed under the SFC object layer with a setting in the object inspector, so that the SFC object is not covered by the image.

"Free text" can also be placed in the chart parallel to that, in order to describe one or more steps in detail.

The resulting SFC code

The SFC code which results from the graphic configuration is automatically entered into the **'Cyclic'** tab.

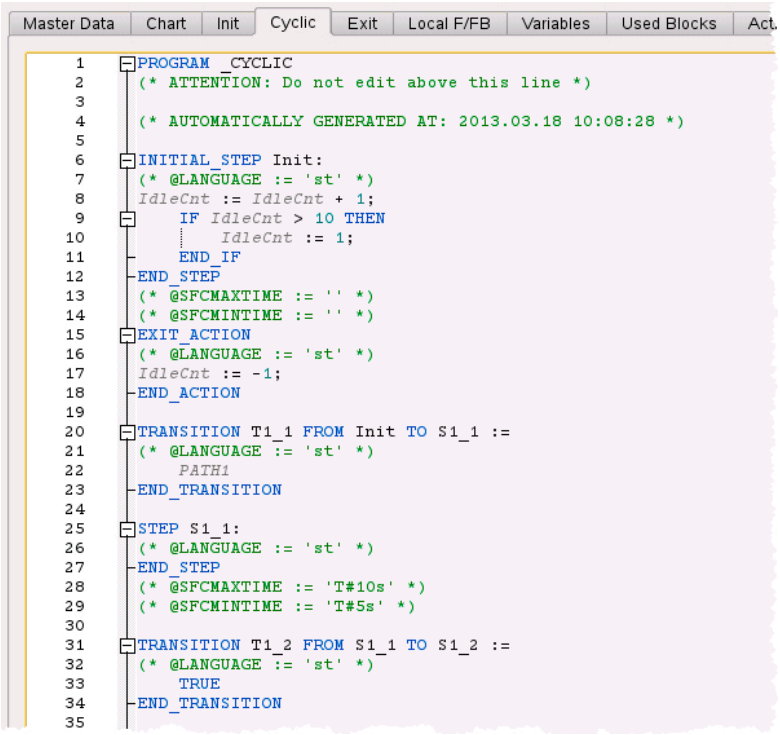


Figure 27: Code view of an SFC program

The cyclic part of the program code is provided in a read-only view. The configuration takes place graphically in the **'Chart'** tab. The resulting SFC code is necessary for the code generation and display in order to present more clarity.

Declaration of actions

Local actions can only be defined in the ST language in the **'Actions'** tab.

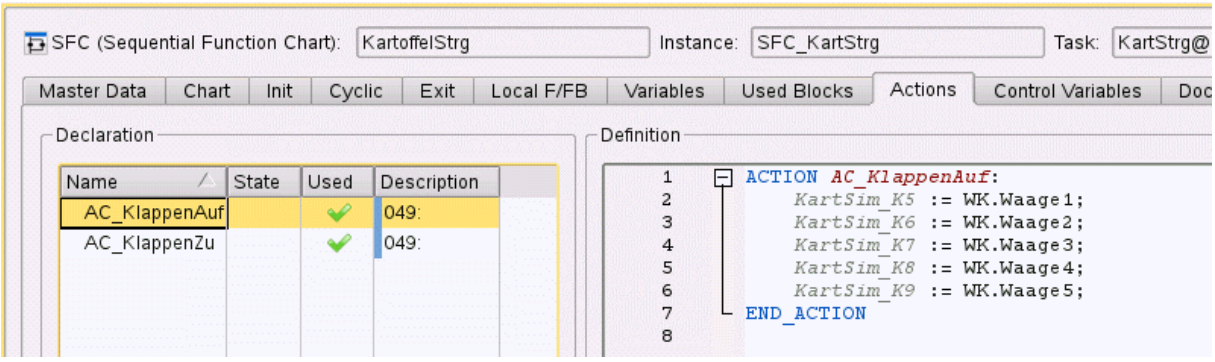


Figure 28: 'Actions' tab of an SFC program

Information about using actions, see Actions in IEC steps.

The SFC Control Variables

Variables for different additional SFC program functionalities can be declared in the **'Control variables' tab** if required.

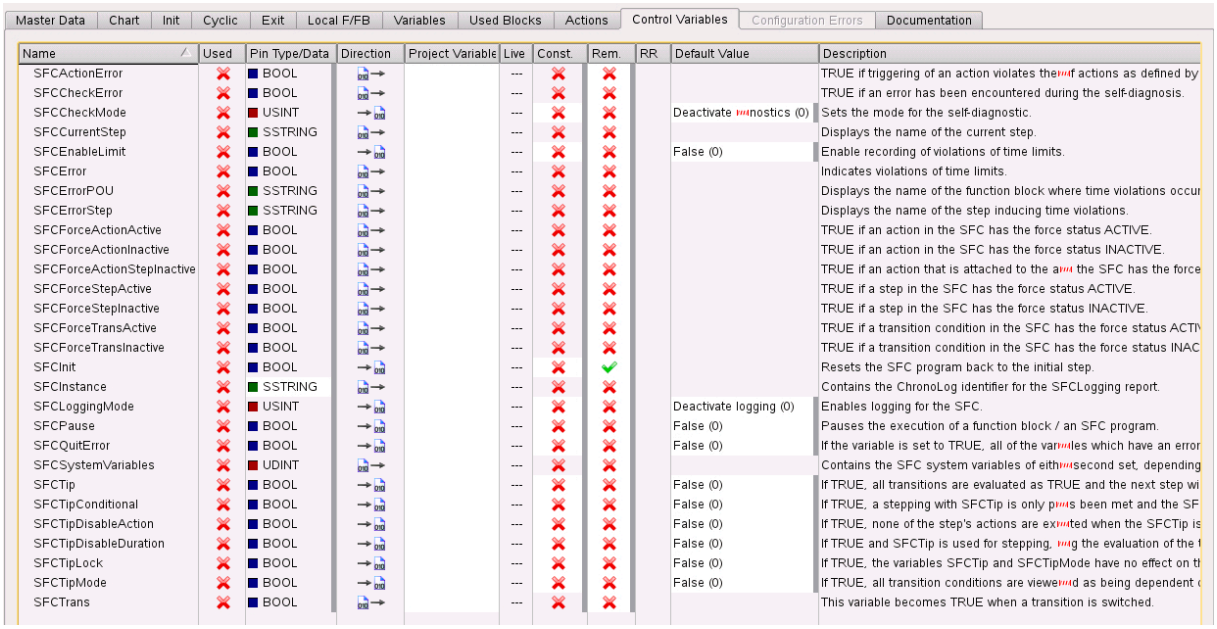


Figure 29: The SFC Control Variables

Further information about using control variables, see Using the SFC Control Variables.

5.3.2 The graphic configuration of an SFC network

The basic configuration (minimal configuration) of an SFC program is shown as an example in the following.

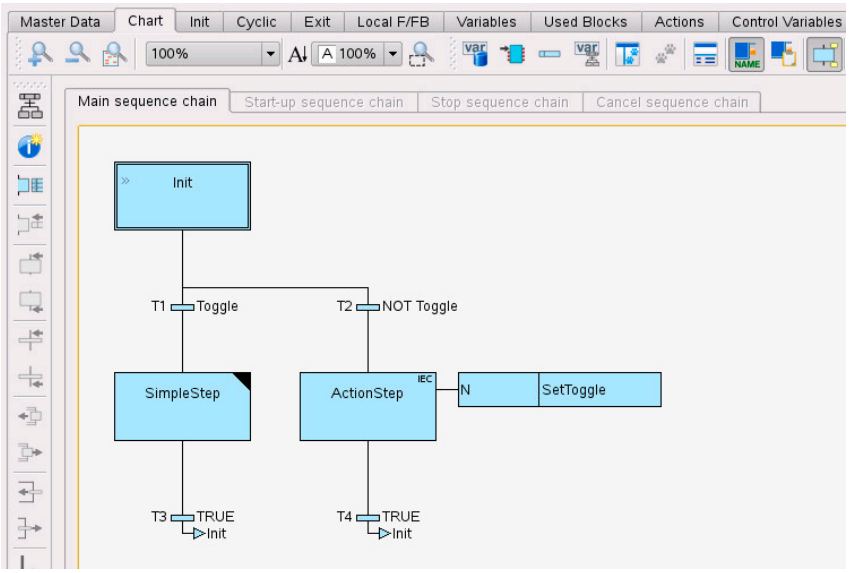


Figure 30: Chart view of an example SFC

The SFC program shown above (the finished configuration example is displayed) toggles between two alternative paths.

The example shows the use of simple steps, IEC steps and actions.

The following steps must be carried out in the scope of the example configuration.

Step	Configuration
1	Create SFC program
2	Configure master data
3	Configure network: Create transition / Create step / Create jump
4	Creating action Allocate action / step

Step 1 & 2: Creating an SFC program and configuring the master data

An explanation of how to create an SFC program and the configuration of the master data has already been shown above. For this, see chapter Creation of an SFC program.

Step 3: Graphic configuration of the SFC network

We start by editing existing transitions after the init step.

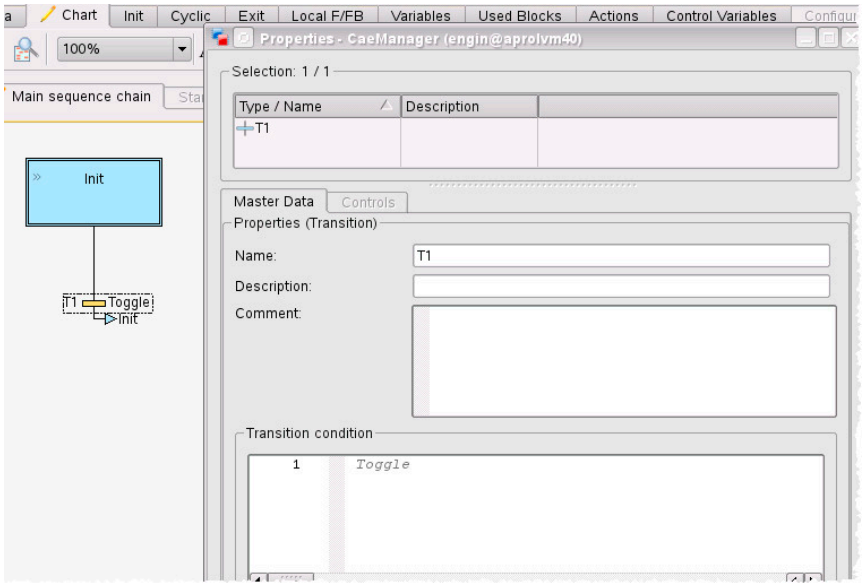


Figure 31: Properties dialog for transitions

Transition '_T1' / context menu 'Properties' or double-click:	
Name	T1
Transition condition	*Toggle*
[**Close**]	

Name, description or comment can be configured for each object in the SFC.

A simple step is inserted after a transition. An 'Entry', 'Cyclic', and 'Exit' action can be configured in a simple step.

The entry and exit actions should not be confused with the init and exit code of the project part.

The entry action is run through once upon entering (turning active) the assigned **step**, whilst the exit action is executed one time upon exiting the step.

In contrast to that, the code in the init and exit tabs is executed during the initialization and de-initialization of the entire SFC program on the **controller**.

The memory requirements on the controller are reduced by using a **'simple' step** (in comparison to an IEC step).

We now mark the transition 'T1' and select the **'Create step behind' context menu**.

Step '_S1' / context menu 'Properties':	
Name	SimpleStep
Min. retention time	T#5s
Cyclic action	*Toggle := FALSE;*

Step '_S1' / context menu 'Properties':	
Following transition '_T1' / context menu 'Properties'	
Name	T3
Transition condition	*TRUE*
[**Close**]	

Then we continue with the creation of an alternative path.

Transition 'T1', select the 'Insert alternative path right' context menu	
Transition '_T2' / context menu 'Properties':	
Name	T2
Transition condition	*NOT Toggle*
[**Close**]	

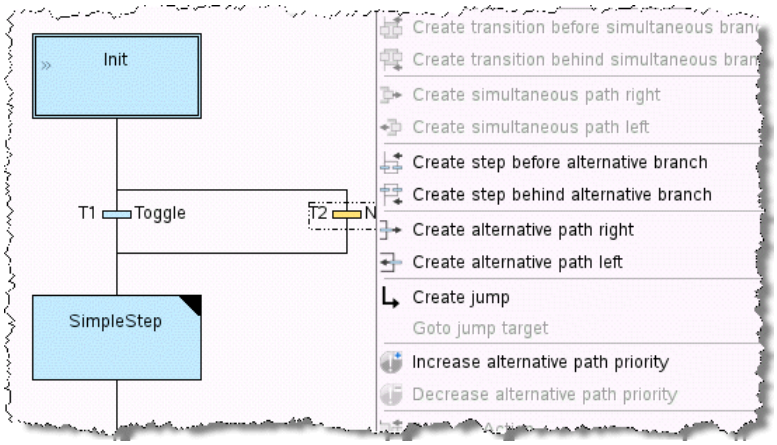


Figure 32: Next step: Insert jump

Please note that the previous step is specified automatically as the target when a jump is created. A jump target can be changed in the properties dialog if this is necessary.

In our example, this is coincidentally the desired init step and must not be changed.

Now insert the IEC step in which one or more local actions will be assigned.

The toolbar can set the **'Type of created step'** in order to create IEC steps.

Figure 33: IEC step selection / de-selection in the toolbar

Mark transition 'T2' / select 'Create transition behind' context menu	
Name	ActionStep

Mark transition 'T2' / select 'Create transition behind' context menu	
Min. retention time	T#5s
[**Close**]	

Step 4: Creation of an action and allocation to a step

In order to create actions, change to the **'Actions'** tab and select the **'Add Action'** context menu in the 'Name' column (section 'Declaration').

In our case, the action is named 'SetToggle' and executes the following program code.

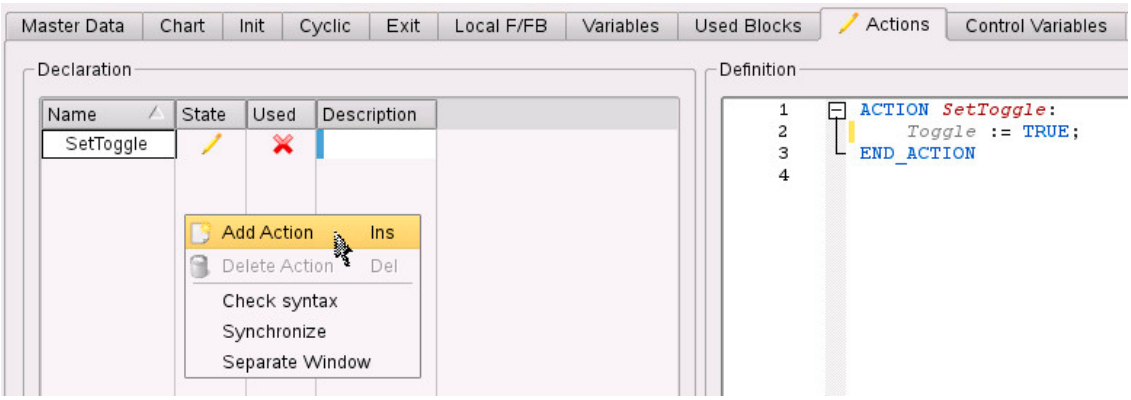


Figure 34: Configuring an action

You can allocate a created action to the 'ActionStep'. To do this, we change to the chart view, mark the step, and open the properties dialog.

Also open the editor for local actions in a separate window using the toolbar.

Figure 35: Toolbar 'Local actions'

In the 'List of actions' which now opens, the desired action can be assigned to a step via drag-and-drop ('Called actions' tab).

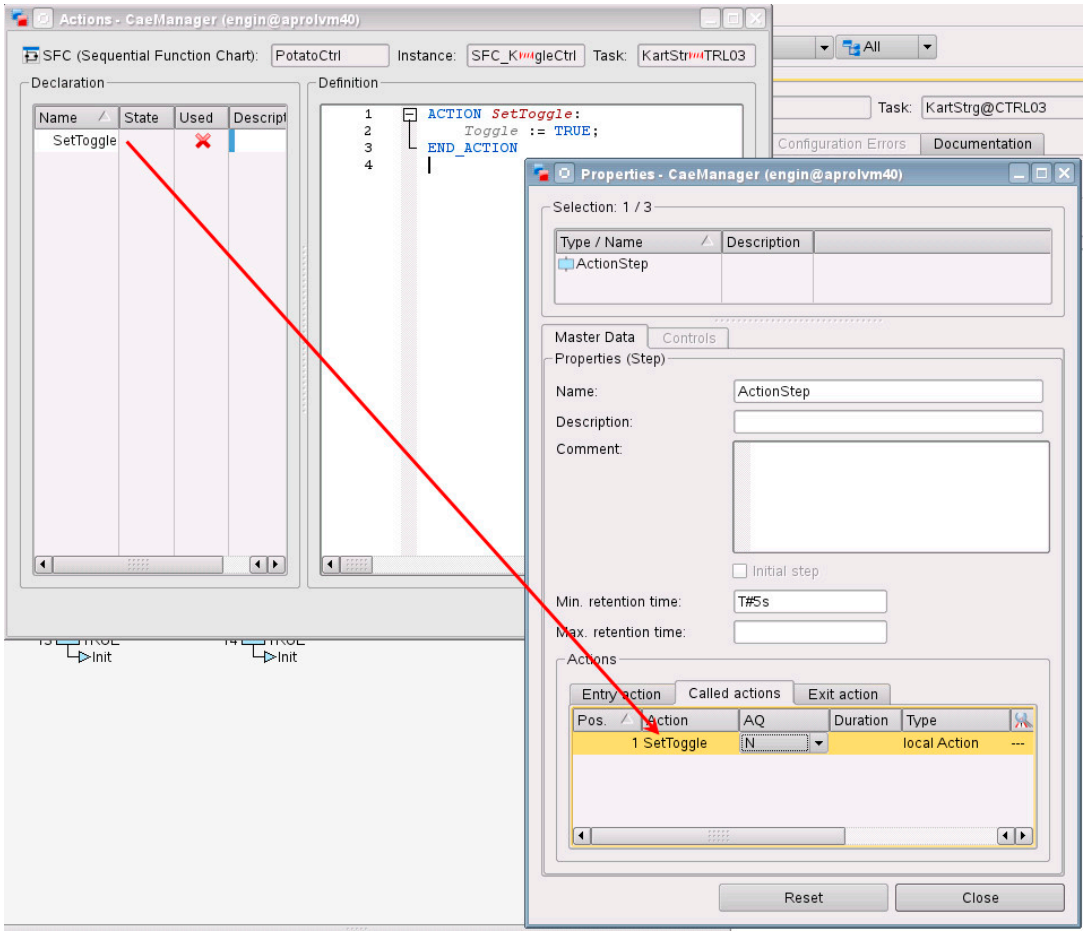


Figure 36: Allocation of an action to a simple step via drag & drop

The allocation of actions to steps can be done with action qualifiers, see chapter Actions in IEC steps.

The configuration of the SFC program is finished with the 'File / Save' menu and a compilation with [F8].

5.3.3 The time monitoring of steps

The timing of each step can be monitored (optional). This makes it possible to check whether a maximum or minimum time has been exceeded. The violation of the configured time can be queried using control variables. (see Using the SFC Control Variables)

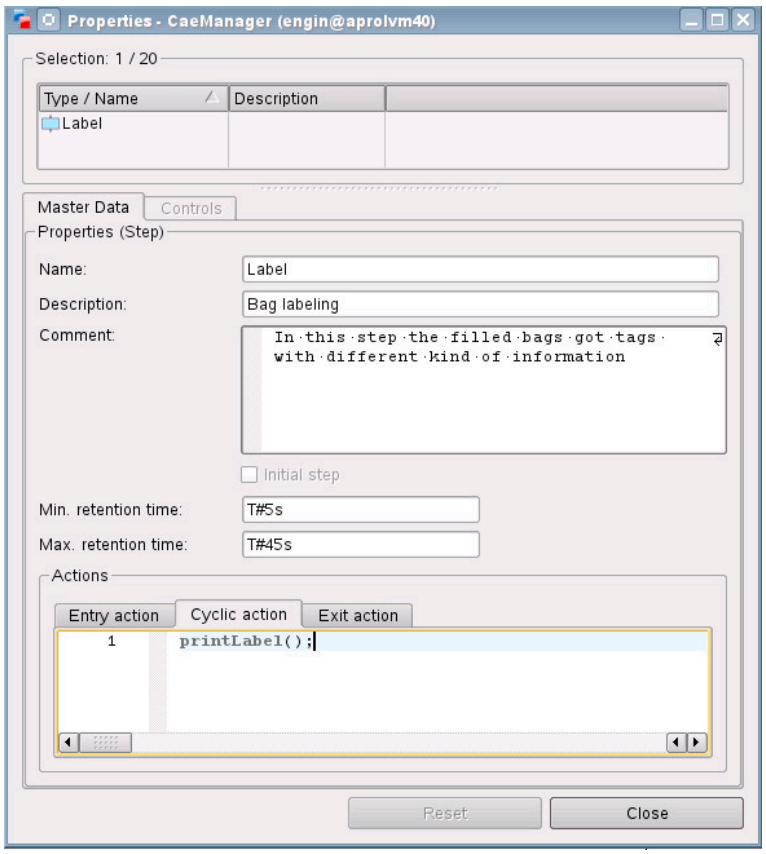


Figure 37: Properties dialog of a step with minimal and maximal dwell time

The minimum dwell time specifies the time that a step should be active for at least.

The maximum dwell time specifies the maximum time that the step is allowed to be active.

The SFC control variable **'SFCEnableLimit'** must also be set in order to monitor violations of the set dwell time.

5.3.4 Use of variables in an SFC

Local variables can be defined and project variables can be chosen for use in the **'Variables'** tab.

Project variables allow a data exchange with other programs (CFC, ST, ...)

Project variables:

Project variables can be adopted comfortably via drag & drop (or double-click) from the list of project variables.

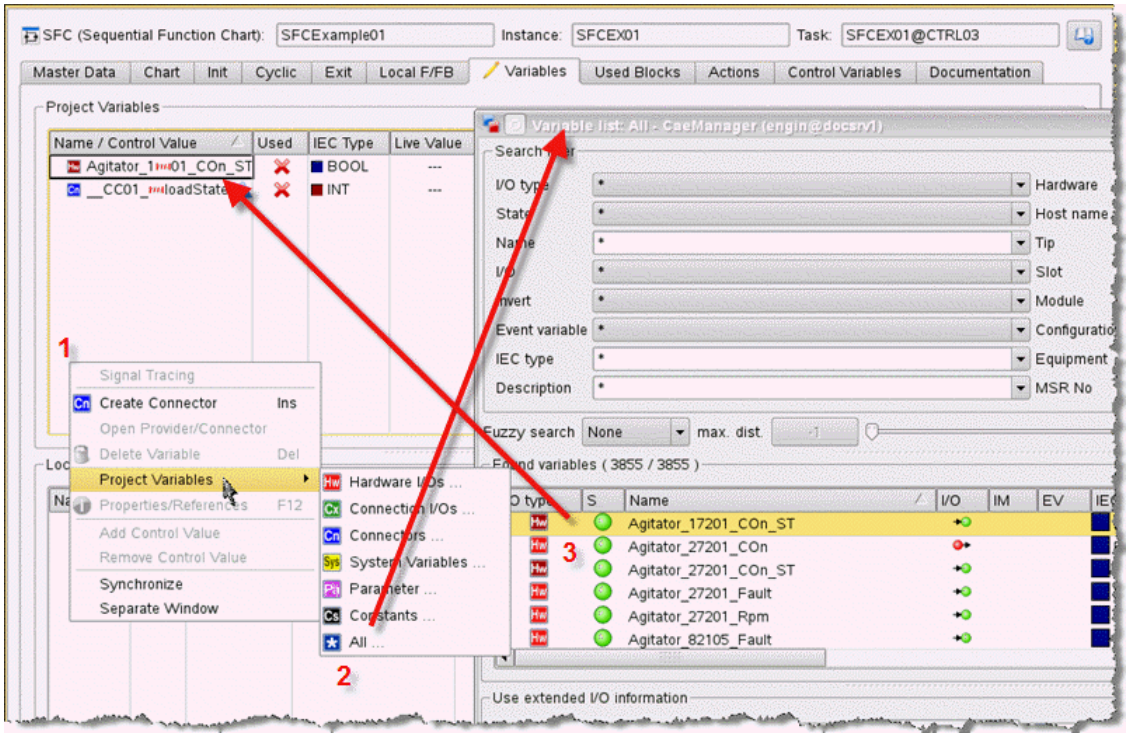


Figure 38: 'Variables' tab

Furthermore, project variables can also be used directly in the code in order to allow for an efficient engineering.

A check is made during the manual synchronization or the automatic synchronization during saving to see if the PVs are in use but have not already been configured. In this case, the PVs are entered into the list of project variables in the **'Variables'** tab.

Local variables:

Local variables can be defined either in the **'Variables'** tab before their use in the ST code, or used in the program code and then transferred to the **'Variables'** tab (synchronization).

The detection of the variables that are used in the program code but have not yet been defined can be carried out via the **'Synchronize'** context menu in the **'Variables'** tab, or takes place automatically during saving.

Variables of the type 'BOOL' are created per default.

The local variables that have been used in the programming must then be configured explicitly in order to make a task generation possible.

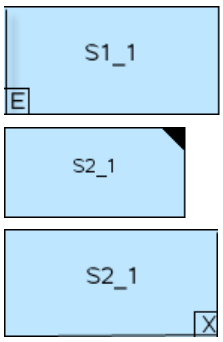
The way in which project and local variables are used and declared is no different in ST and SFC.

5.3.5 Actions in simple steps

Simple steps have one 'entry action', one 'cyclic action', and one 'exit action' that are stored as code in the properties dialog of the step.

The ST code can be entered directly in the 'Entry action', 'Cyclic action', and 'Exit action' tabs.

The type (entry, cyclic, exit action) is marked explicitly in the SFC chart:



Entry action
An action that **is executed once when the corresponding step is active** can be declared in the respective tab.

Cyclic action
An action that is executed **as long as the corresponding step is active** can be declared in the respective tab.

Exit action
An action that **is executed once when the corresponding step is inactive** can be declared in the respective tab.

The exit action of a step and the entry action of the **following step** are processed in the same cycle.

5.3.6 Actions in IEC steps

The configuration and display of local actions, which are used within the project part and can be allocated to IEC steps in the SFC network, takes place in the SFC chart, in a separate 'Local actions' editor.

The configuration can also be made in the **'Actions'** tab.

The editor is divided in a declaration part and a definition part. An overview of all configured local actions is on the left side (declaration).

On the right side (definition), there is a text editor that shows the code of the action selected in the table. The code can be edited here and is checked for its syntax.

Declaration:

Column	Short description
Name	The name of the local action for use in the SFC, and triggering in a step, is entered here.
Status of the action	Possible statuses are: - the action code was modified - the code was detected as being erroneous during the syntax check - a warning was generated during the syntax check of the code - a message was output during the syntax check of the code
Status of usage	Possible statuses are: - the action was allocated to an SFC step - the action is not allocated to an SFC step
Description	A description of the action can be entered by the user here.

Operations are available via a context menu, e.g. inserting of removing local actions:

Menu item	Description
Add Action	A new action is added. An action name is specified automatically.
Remove Action	The selected action is removed from the list of local actions. When an action is removed from the list of local actions (via 'Remove action'), the user is presented with an information dialog and can selectively remove existing assignments.
Check syntax	A syntax check of the code of **all actions** is carried out.
Synchronize	A synchronization of the used local actions is carried out.

Definition:

The text editor is used to define the action code of the selected action. The editor offers all of the comfortable normal APROL functions.

The following syntax is valid for actions:

```
ACTION <Aktionsname>
    ...
END_ACTION
```

This syntax is stipulated automatically by the system, so that the actual ST code is edited within the automatically generated code frame.

The 'Action selection' dialog is opened with the respective toolbar button, which offers a confortable selection of the Boolean and local actions that are available in the project part.

Figure 39: Opening the action allocation dialog from the toolbar

The actions that are available for selection are shown in a dialog, are sorted by type, and can be assigned to the IEC steps via drag & drop.

The steps which have the selected actions assigned to them are listed in the lower part of the dialog.

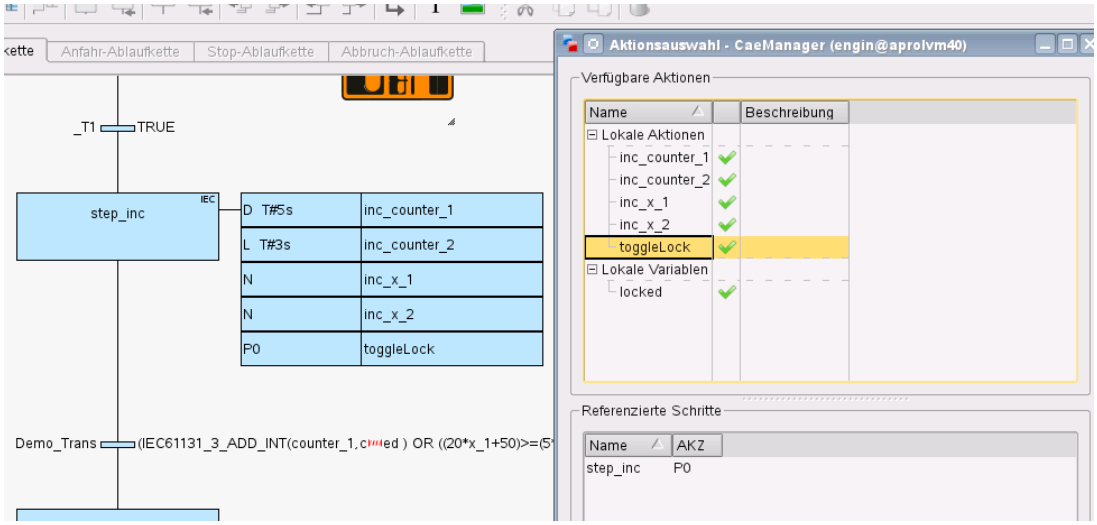


Figure 40: Selection of the available Boolean and local actions

How is a Boolean action used in an IEC step?

Step	Description
1	Open the list of project and local variables with the respective toolbar button in the 'Chart' tab.
2	Open the 'Properties' dialog of the IEC step in which the Boolean action should be inserted. Select the 'Called actions' tab.
3	Drag the desired variable (which must be of the type BOOL) into the 'Called actions' tab via drag & drop.

Hardware I/Os can only be used **as Boolean actions in an IEC step** via a local PV mapping. After configuring the mapping, the local variable must be allocated as Boolean action instead of the project variable.

What are action qualifiers?

Action qualifiers (AQ) define an activation condition for the corresponding action, i.e. the activation or deactivation of the action, or the value for the allocation to the Boolean variable is controlled with the AQ. The release condition is only applicable when the action is active.

With these qualifiers, actions can be executed in a saving, non-saving, pulse-triggered, or time-delayed manner.

A complete overview of all action qualifiers can be found in the APROL documentation in **B2 Project Engineering / Logic creation with SFC / How are steps and actions used in an SFC? / Actions in IEC steps**

5.3.7 Using the SFC Control Variables

The additional functions (e.g. time monitoring of steps) can be activated for each SFC program with the declaration and usage of SFC control variables.

The SFC-internal control variables can be mapped to project PVs in order to use them in the logic, for example, to be written by an external logic.

The project variables can be created in the 'Control Variables' tab with the **'Create Project Variables Automatically'** comfort function (context menu).

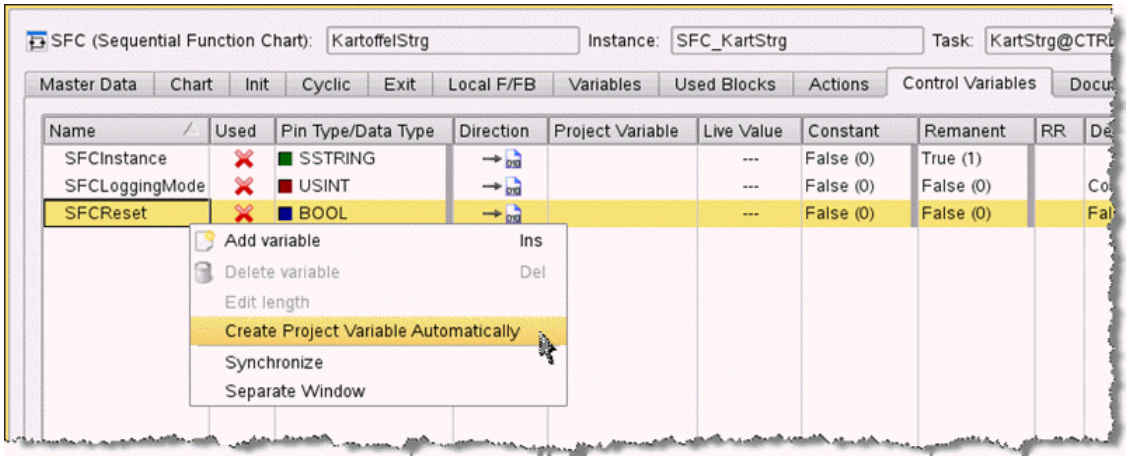


Figure 41: 'Create Project Variables Automatically' context menu

A prefix can be entered for the name of the project variables in the dialog that follows. The input field is predefined with the instance name of the SFC per default.

The name of the project variables are formed as follows:

<Prefix>_<Name of the control variables>

If project variables with different names have been configured, the overwriting of the variables that have already been configured must be confirmed by the user in a separate dialog.

All control variables of the type 'input' can be written by the logic.

If control variables are written by the logic, the configured operator rights for the SFCViewer do not have an effect.

Therefore, project-specific operator rights must eventually be configured in the visualization.

A complete table of all available action qualifiers can be found in the APROL documentation in **B2 Project Engineering / Logic creation with SFC / How are control variables used in an SFC?**

5.3.8 Exercise - Dispersion mixer

A mixing plant must be configured. The elements water and color will be mixed to a dispersion for this.

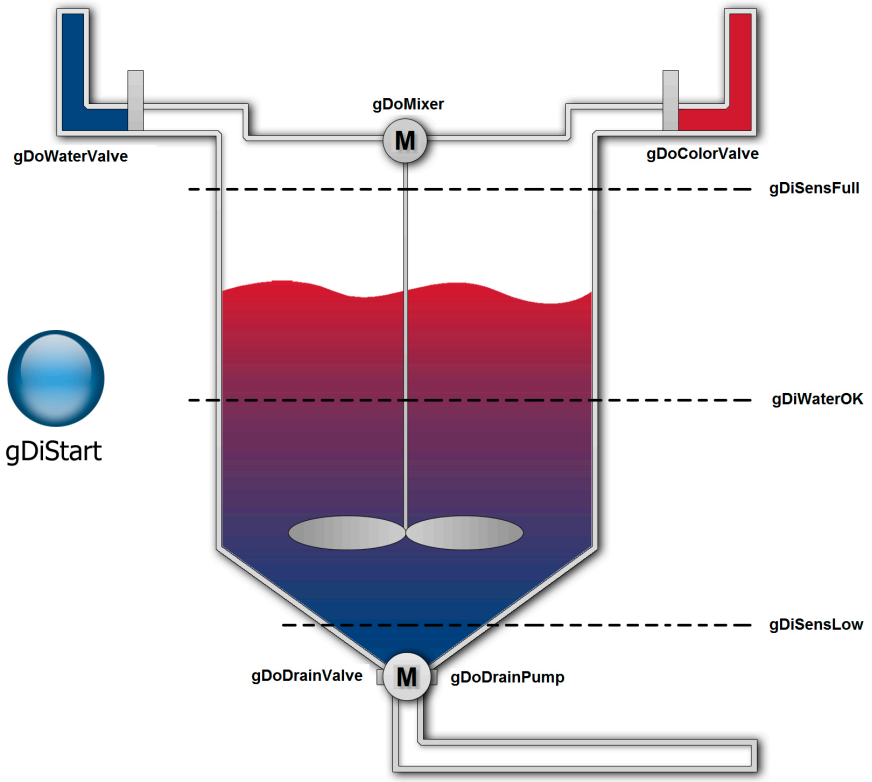


Figure 42: Schematics of a paint mixing system

The mixing program will run according to the following procedure:

- The mixing program waits until the start button is pressed (**gDiStart**).
- Water (**gDoWaterValve**) is added to the container until the "**gDiWaterOK**" sensor is triggered.
- The mixing unit (**gDoMixer**) starts and paint (**gDoColorValve**) is added until the "**gDiSens-Full**" sensor is triggered.
- The mixing time should take 30 seconds.
- The drain (**gDoDrainValve**) and drain pump (**gDoDrainPump**) are turned on to empty the container.
- The draining process is complete when the "**gDiSensLow**" signal is triggered.
- The starting situation is restored.

Task: Implementing the mixer control

The mixing procedure just described should now be programmed. The following points must be carried out:

1) Sketch out the necessary SFC.
    - Define the steps.
    - Define the transitions.
    - Define the actions.

2) Determine which steps need an IEC action.

3) Realization of the demands in an SFC program

The stirring unit will be active over several steps.

Its functionality can be implemented in a number of different ways:

- Switching the stirrer on and off in the input and output action of the steps
- Stirring unit is a parallel step to the other steps.
- Stirrer is turned on with the "S" action qualifier and turned off at the end of the sequence in an action which has the "R" action qualifier.

An example can be found in the appendix, in Exercise solutions.

5.4 Diagnostic functions in an SFC

5.4.1 The monitor mode in the SFC editor

Prerequisite for the debugging of an SFC on a controller is that the debugging database exists in the runtime environment and that the engineering database can be reached.

Figure 43: Activation of the debugging

After choosing the debugging mode (via toolbar or key combination [**Ctrl**] + [**D**]), the actual state of the SFC on the controller is displayed by a colored marking of the steps, actions, and transitions in the SFC editor.

Live values are shown in all respective tabs when the debug mode is active.

The corresponding steps and actions are marked with a red cross and the live values are not displayed if the necessary status variables are not valid.

In order to allow an SFC debugging **with an influencing of the sequential control**, the 'Controllable by SFCViewer' control option must be activated in the master data of the SFC.

The activation of the option allows control possibilities in the SFCViewer and SFC editor to be chosen.

The debugging mode can be activated (via toolbar or [**Ctrl**] + [**D**]) after the 'SFC' project part is activated, compiled, a 'build' of the respective task is made, and loaded to the controller.

If the **SFCViewer is started in the runtime environment**, the operator rights for SFC debugging must be configured (in the OperatorManager).

No additional rights other than 'Sequential Function Chart: View' are necessary for the **SFC debugging in the CaeManager** (via SFC editor).

(Menu 'Extras / User Management / User / tab 'Rights Management' / Project rights 'SFCViewer'')

The color environment of the SFC objects and SFC view can be user-specifically configured in the user options of the CaeManager ('**Extras / User Options / 'General**' tab).

Color settings for the debugging view:

The color display of the steps that have not / have been executed up-to-now, the active steps and actions, the forced objects and the background in the SFC can be set **project-globally** in the 'Runtime options (2)' tab.

The influencing course of an SFC or the states of the SFC elements in an SFC is the same as the functionality / configuration that has already been described for the SFCViewer. For this, see chapter The SFCViewer.

5.4.2 The SFCViewer

An SFC can be diagnosed and influenced by using the SFCViewer. It can be opened in the engineering and runtime environments.

Amongst others, it may be necessary to open several SFCs at the same time (e.g. in a multi-screen environment).

The configuration of the instances (maximum 5) takes place in the CaeManager, 'APROL system' project part, section 'System monitoring', SFCViewer context menu 'New Instance'.

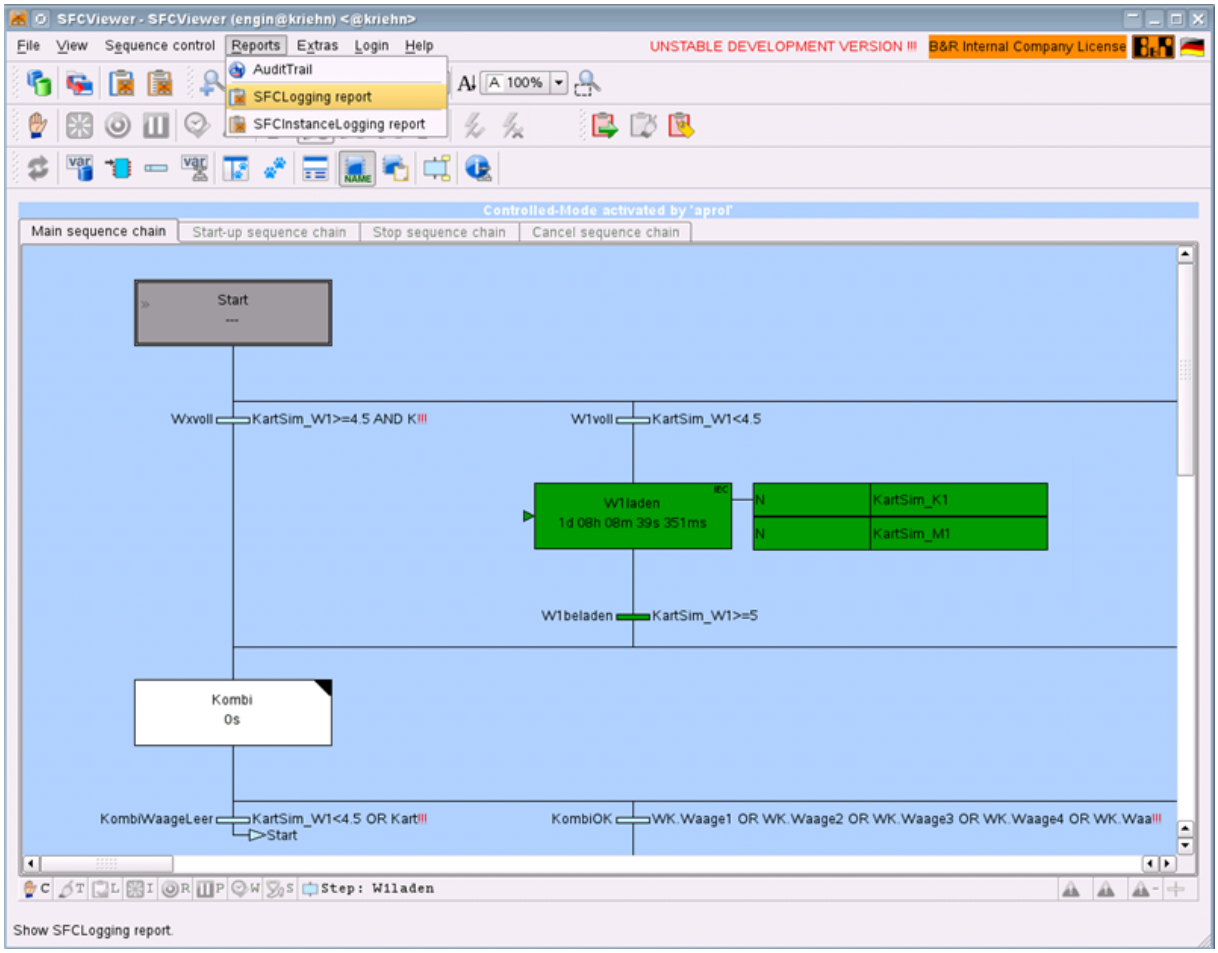


Figure 44: SFCViewer

Online debugging for function blocks (SFC)

An instance in an SFC program or CFC, i.e. in a CAE project, is the pre-requisite for debugging function blocks (SFC).

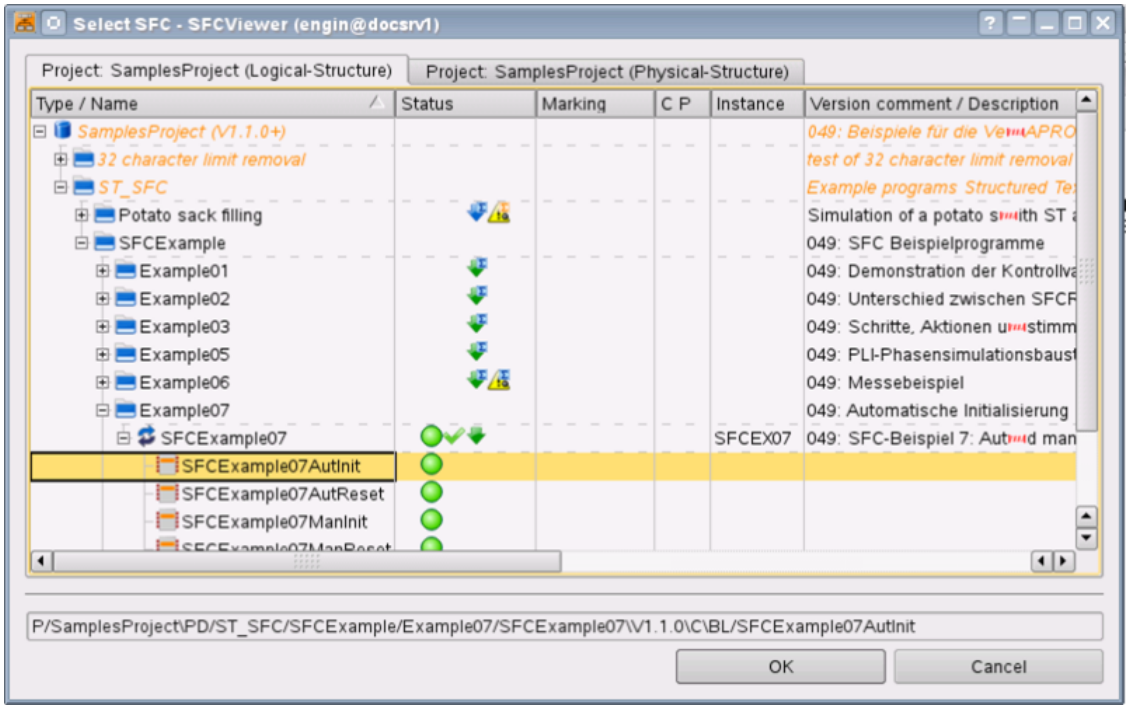


Figure 45: Selection of the function block (SFC) in the SFCViewer

Open the SFC selection dialog (**'File / Select SFC'** menu) and select the function block (SFC) instance.

After the debug mode is activated in the SFCViewer, an online debugging (depending on the rights granted) can take place.

5.4.2.1 Influencing the course of an SFC

The sequential control can be influenced by activating the 'Controlled-Mode' in debugging mode.

This has the effect that the SFC is set into a state where the activated operational elements, respective of the rights granted, have an influence on the course that has been engineered.

Illustration: Activation of the debugging and the sequential control

Possible actions for sequential control of an SFC:

Icon	Description of the possible actions
	**Controlled Mode** Activation of the Controlled-Mode for influencing the sequential control of this SFC.
	**Init** Sets the SFC back to the initial step. The initial step is active as long as 'Init' is activated, but it is not processed and the time monitoring is switched off. The SFC is processed normally again when 'Init' is set back to FALSE. The transition thereafter is not evaluated as long as 'Init' is active.

Icon	Description of the possible actions
	**Pause** Pauses the processing of the SFC. The state of the SFC remains unchanged. The time monitoring is switched off as long as 'Pause' is activated. When the state is changed to 'SFC pause', the transition evaluation, the non-saving actions and the saving actions are left out, i.e. are stopped.
	**Time monitoring** Activates / deactivates the time monitoring for the step execution. If activated, time overruns are registered in the 'SFCError' control variable.

This is only an excerpt of the possible actions for sequential control. A complete list can be found in the APROL product documentation in **B2 Project Engineering / Logic creation with SFC / How can the course of an SFC be monitored in the CaeManager?**

There are differentiated rights for authorization available for **influencing the course of the SFC** in both the engineering and runtime environment.

5.4.2.2 Influencing the state of the SFC object by forcing

With respect to the necessary rights, individual steps, actions, and actions can be 'forced active' or 'forced inactive' in order to have a differentiated debugging function.

Certain steps can be activated with this, for example during commissioning, and thus the SFC can be put into a defined state.

The actual forcing is not yet adopted by this (i.e. the steps, transitions, and actions have not been influenced) because **the actual forcing must be carried out with the 'Confirm all force states' function (via context menu or toolbar)**.

Prerequisite for forcing is that the **'Controlled-Mode'** is activated.

The forcing can be turned off in the same way, with the 'Remove force requests' and 'Remove all force states' context menus.

Special features when forcing actions

When forcing actions, there is the differentiation of **forcing actions in general** (inactive / active forcing) and **forcing in the context of the step** (only inactive forcing).

The **general forcing** ensures that one action with an allocation to several steps is always forced respectively (active / inactive) in all of the steps.

The forcing **in the context of a step** limits the inactive forcing to the single allocation in the selected step. When the step is active, the action does not become active in the context of this step; other instances of the same action in the context of other steps will eventually become active.

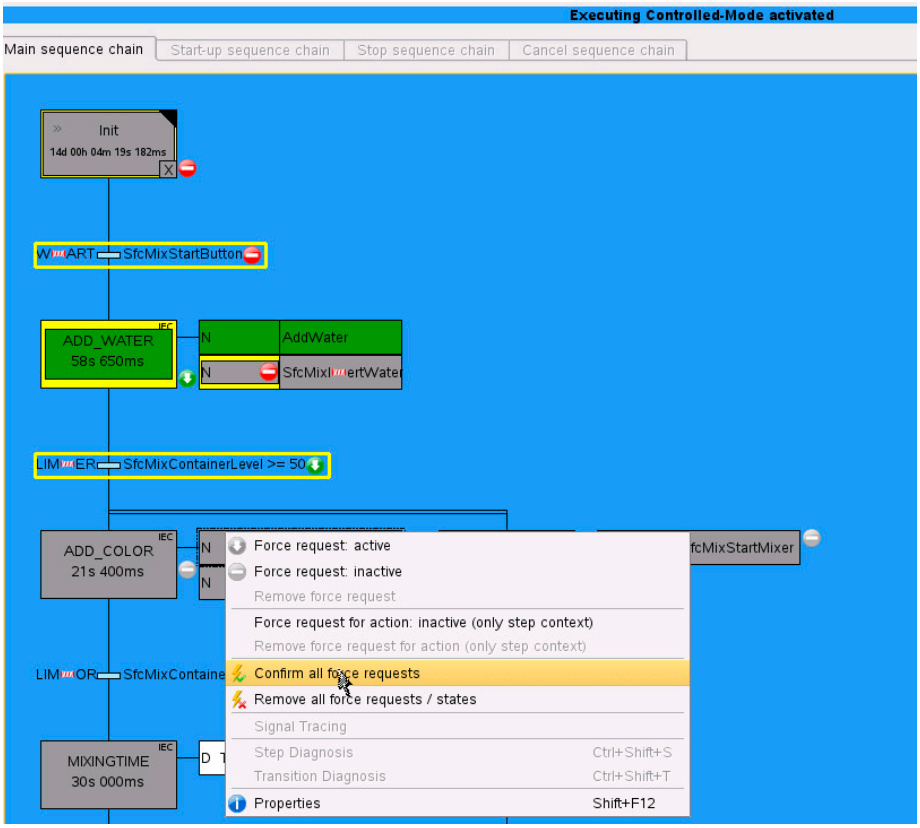


Figure 46: Forcing steps, transitions, and actions

Action	Short description
Force request: Active	Forces the execution of the selected element and overwrites the transition condition with TRUE. Must be confirmed with *'Confirm all force requests'*.
Force request: Inactive	Forces the non-execution of the selected element and overwrites the transition condition with FALSE. Must be confirmed with*'Confirm all force requests'*.
Remove force request	Removes the force request of the selected element.
Force request for action: inactive (only step context)	Forces the non-execution of the selected action when the respective step becomes active. Must be confirmed with *'Confirm all force requests'*.
Remove force request for action (only step context)	Removes force states for actions 'in context of a step'. Must be confirmed with *'Confirm all force requests'*.
Confirm all force requests	Confirms and activates all set force states.
Remove all force state	Removes all set force states.

5.4.2.3 Influencing the state of the SFC object by tipping

Apart from forcing, the tip mode is a second way of influencing the sequential control of an SFC. Switching to the 'tip mode' allows an operator a controlled stepping in the scope of the normal course of the SFC.

Prerequisite for tipping is also that the 'Controlled-Mode' is activated.

The tip mode is activated over the toolbar. The settings explained below can then be made and it is possible to tip through the SFC network.

A tip icon before a step shows you that a step has been jumped to using a 'tip'.

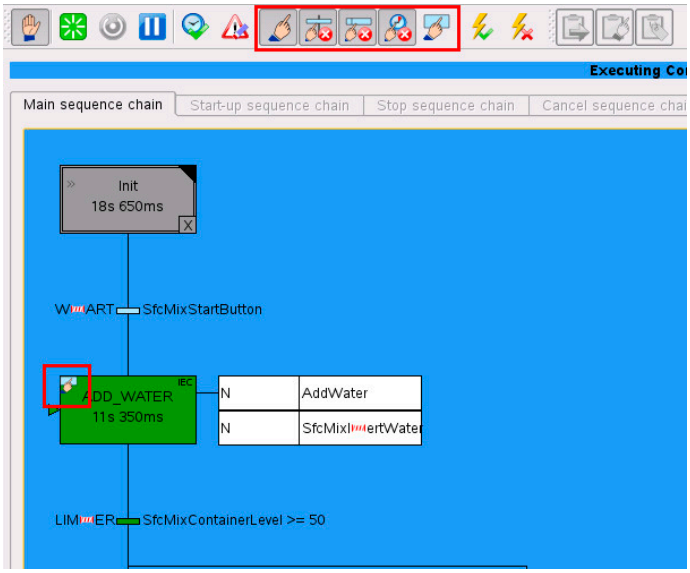


Figure 47: Forcing steps, transitions, and actions

Action	Short description
Tip mode	Activates/deactivates the tip mode for this SFC. If activated, all transition conditions are evaluated as being FALSE. This will switch manually from one step to the next if a 'tip' is used.
Ignore transition condition during 'tip'	Activates/deactivates taking the transition condition into account while the tip mode is active. Note: If activated, the transition condition will not be taken into account while stepping manually via 'tip'.
Do not execute actions when tipping	Activates/deactivates the execution of actions in steps which are activated by a 'tip' while the tip mode is active.
Ignore dwell time during 'tip'	Activates/deactivates the minimum dwell time for steps which are activated by a 'tip' while the tip mode is active.
Tip	If 'Ignore transition condition during tip' is deactivated, then a stepping to the next step only takes place when the transition condition is met. Otherwise, a switch is made without taking the transition condition into account.

While tipping through an SFC network which has alternative branches, the outermost left path is always jumped to, because of the highest priority and if all transitions are TRUE or the option for ignoring the transition conditions is set.

Task: Forcing and tipping an SFC program

After you have finished and downloaded the previous 'mixing control' task, open the corresponding SFC program in the SFCViewer.

Test the different functionalities which the SFCViewer has to offer. You can, for example, **actively/inactively force** different objects or **tip** through the SFC network.

5.4.2.4 Locking the control of an SFC

An SFC can be controlled by activating the Controlled-Mode. The signals from the logic are ignored in Controlled-Mode, and the operator can thus have a direct influence on the course and the state of the objects.

Two operators viewing the same SFC in Controlled-Mode both have the respective possibility to intervene, and would most probably interfere with each other.

In order to avoid this situation, a locking mechanism has been made available that gives an operator an exclusive access to the Controlled-Mode and thus the possibility to influence the SFC. The exclusive access of an operator in Controlled-Mode is called 'executing Controlled-Mode'.

The adoption of the SFC Controlled-Mode can take place either **with or without** a query to the currently executing operator.

An immediate adoption of the Controlled-Mode by the querying operator **can** take place **at any time**, even if the executing operator does not give a prompt response.

The locking mechanism offers the following additional benefits:

- **Security during concurrent access**
  The exclusive access of an operator in Controlled-Mode (executing Controlled-Mode)
- **Possibility to adopt the executing Controlled-Mode**
  Adoption with query to the operator actually controlling the process
  Immediate adoption of the Controlled-Mode (guaranteed adoption in case of emergency)
- **Controlled-Mode overview**
  Overview of all controlled SFCs in one dialog

Executing Controlled-Mode

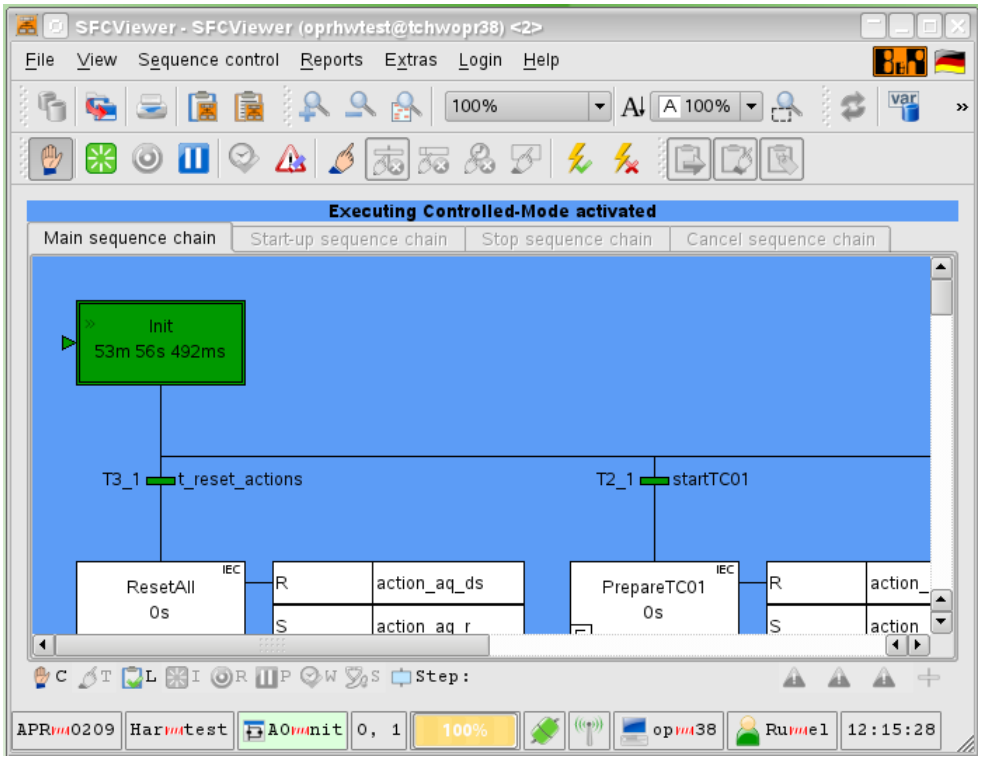


Figure 48: Display of an executing Controlled-Mode

The operator who first activates the Controlled-Mode in an SFC is the one who obtains the Controlled-Mode.

The other operator can see which operator has a currently activated Controlled-Mode in a status message above the chart view.

5.4.3 Graphic transition diagnosis in the SFC

The graphic transition diagnosis offers the possibility to view the inputs of a transition condition and by analyzing the conditions, to be able to construct an opinion about the state of the transition.

It is possible to diagnose why a transition does not switch with the transition diagnosis.

Because the interpretation of complex Boolean expressions in a transition condition is a difficult thing to do, a transition condition is displayed graphically (block diagram).

The transition diagnosis corresponds to the transition expression, which was programmed in the IEC language 'ST'.

The following applies to the **debug mode in both the CaeManager** and the **SFCViewer**:

All input variables, their current values, all values of the partial results, and the end result are visualized for the respective transition.

It is also shown if a transition is 'forced'.

The graphic transition diagnosis offers the following comfortable functions:

- Display of the descriptions of the input variables and the APROL library functions in the tool tip of the object.
- Direct forcing of transitions from the transition diagnosis.
- Marking of input values and intermediate results that cannot be accessed properly, e.g. when there is no connection to the controller.
- The corresponding transition is marked in the SFC chart with a blue frame when the mouse is placed over the 'SFC Transition Diagnosis' dialog. It is thus ensured that the respective transition can be identified if several 'SFC Transition Diagnosis' dialogs are open at the same time.

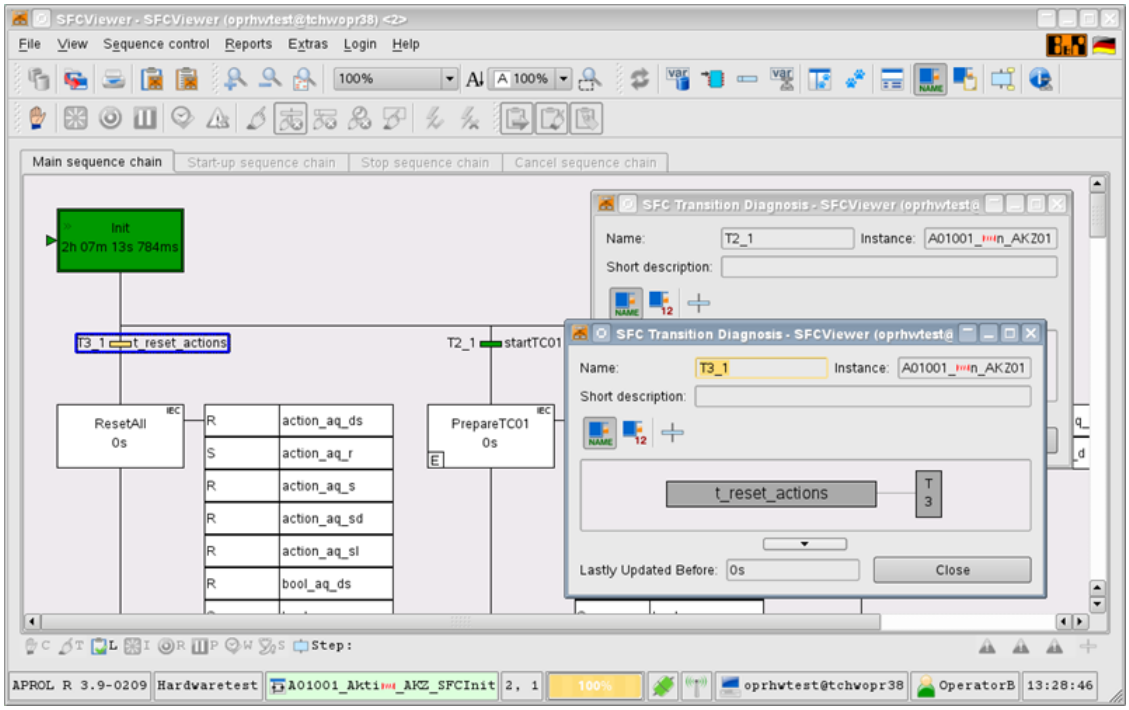


Figure 49: Diagnostics of a transition which is currently being evaluated

- There is a display field that displays the period since the last refresh for the currently displayed transition in the bottom part of the dialog.
- Automatic size adjustment: If the window size of the 'SFC Transition Diagnosis' dialog is changed, then the corresponding graphic display is adjusted to fit the window space that is then available.

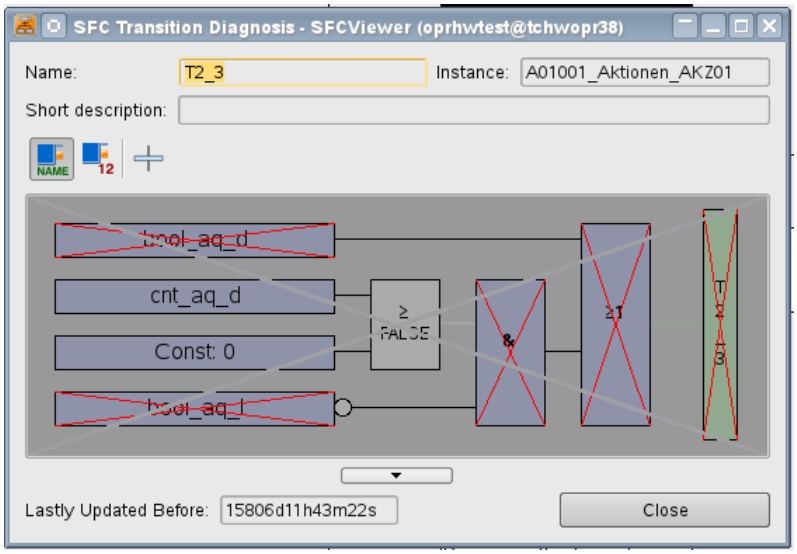


Figure 50: Graphic transition diagnosis in the SFC

Opening the transition diagnosis:

- Selection of the transitions and then the 'SFC Transition Diagnosis' context menu: The transition diagnosis of this transition is displayed as a block diagram.
- Selection of the 'SFC Transition Diagnosis' context menu on a free area of the SFC: The transition diagnosis of the transition currently being processed is displayed as a block diagram. When dealing with alternative paths, the transition in the outermost right hand alternative path is displayed.
- **SFCViewer**: Via double-click on the transition.
- **CaeManager**: Via double-click on the transition **when the debugging is switched on**.

The following display options are available to you via the toolbar buttons:

The names of the input variables are displayed.

The current values of the input variables are displayed.

The display mode toggles between 'Display the last evaluated transition' and 'Display the last selected transition'.

The graphic elements are displayed in 'grey' and the toolbar buttons are insensitive if the **debugging mode** is not active.

The following colors can be configured in the project properties ('Runtime 2/2' tab):

- Color for Boolean states with status 'FALSE'
- Color for Boolean states with status 'TRUE'
- Color for numeric states
- Background color for a transition **evaluated as active**
- Background color for a transition **evaluated as not active**

5.4.4 The diagnosis of steps

Motivation and advantages

All of the relevant data concerning the current SFC step is shown in a separate window in the SFC step diagnosis. Variable names, descriptions, live values, expected values, and usage type are all a part of this.

The following question can thus be answered: What happens when the step is executed?

This means:

- The results which a step has on the sequence chain or externally (project communication) can be detected directly with the SFC step diagnosis.
- The SFC step diagnosis displays **values before, during, and after the execution of the step**.

The SFC step diagnosis can be opened in the context of an SFC step, via the respective 'SFC Step Diagnosis' context menu. It is opened via a double-click on the corresponding block in debug mode or in the SFCViewer.

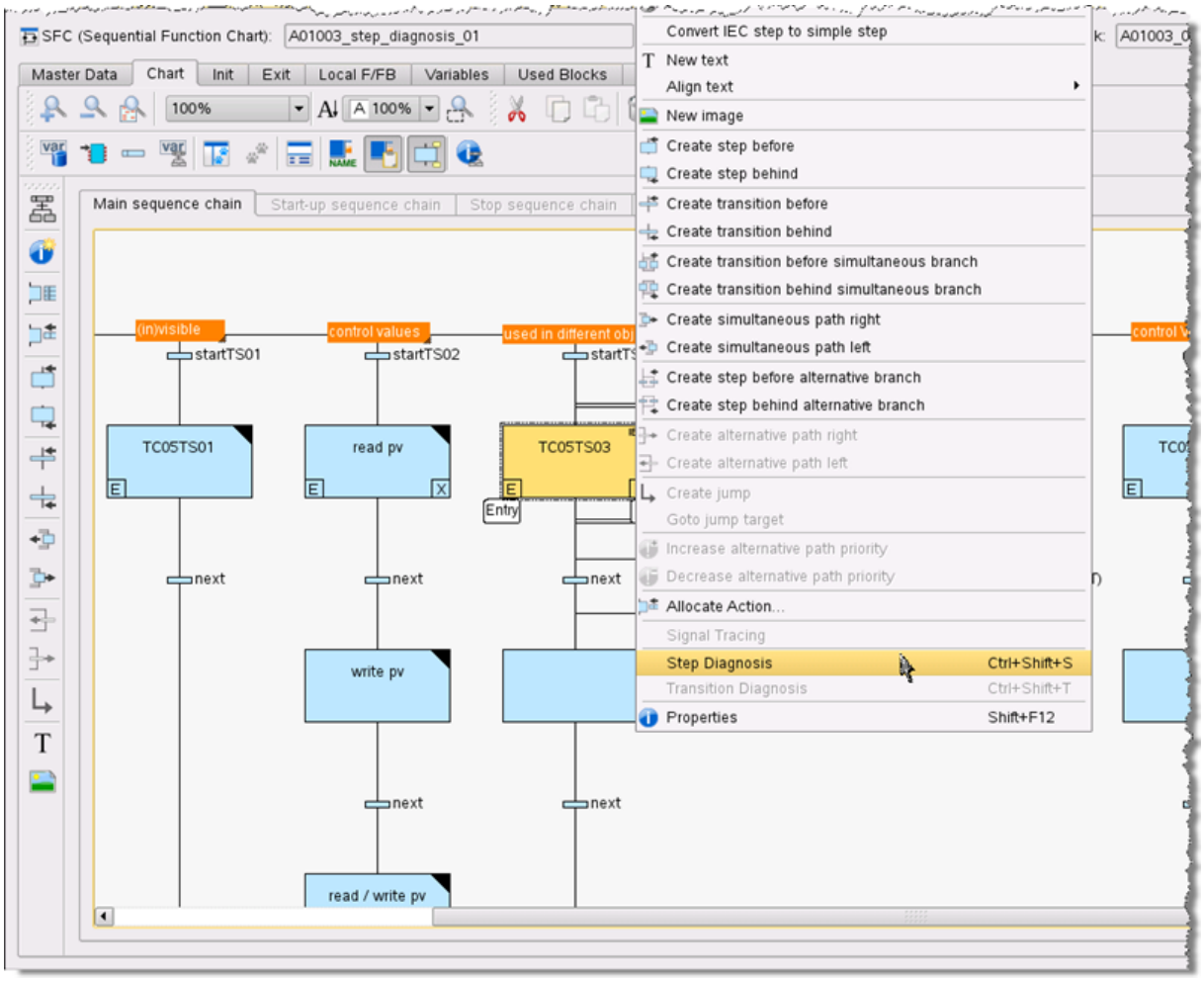


Figure 51: Opening the SFC step diagnosis

The step diagnosis can be opened for several steps (in the same way as the transition diagnosis), but only one instance per step.

If several dialogs are open and one obtains the focus, the respective step is marked blue in the SFC.

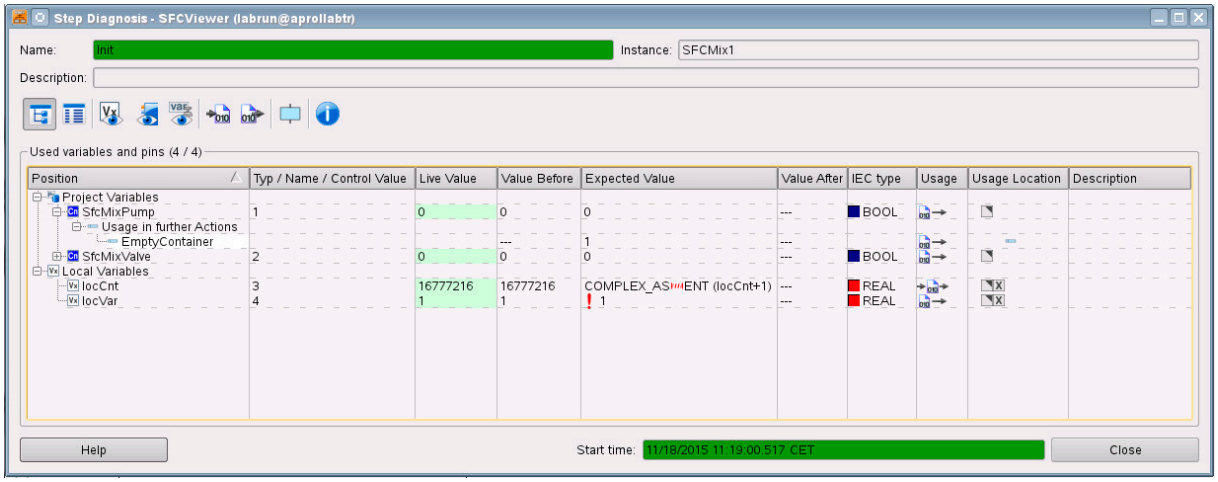


Figure 52: Opened SFC step diagnosis

Display of the used variables/pins per step

- The project variables which are used and those which have been configured to map to local variables are shown in SFC programs.
- Only the input and output pins are shown in SFC function blocks per default, depending on their visibility.

The **configuration of the step diagnosis** , e.g. which variable is shown in which position or the general visibility of variables in the step diagnosis, is carried out in engineering / in the SFC editor of the respective step.

If a variable is locally mapped, only the corresponding project variable is shown in the 'Project variables' node. The name of the local variable is in the tool tip.

There is a separate entry in the view for each variable, in which all relevant information is shown.

The following 'Child elements' can also be displayed for each variable:

- Other steps in which the variable is used (including the write/read direction)
- Other IEC actions in which the variable is used (including the write/read direction)
- Assigned control values

There are also many filter possibilities via a toolbar. An explanation of the filter possibilities can be found in the respective tool tips and the APROL product documentation.

The filter buttons ´can be used in debug mode and in the SFCViewer.

Figure 53: Filter Options

Apart from the filters, the toolbar offers the following functionalities:

Display active step:
Activates the 'Tracing mode' which shows information about the last active step.
In simultaneous chains, this is the step with the 'highest priority', i.e. the step in the first simultaneous path.

Step properties:
The object inspector of the step which is open in the step diagnosis is shown.
The object inspector of the selected step can be opened via the context menu.

Detailed information about the 'Step diagnosis':

A detailed description and explanation of the individual columns can be found in the APROL documentation in B2 Project Engineering / Logic creation with SFC / Note about the implementation of an SFC.

There is also information about how the step diagnosis **table layout** and the **color selection** of an AFC network is carried out.

5.4.5 Logging functions in an SFC

What are the benefits of SFC logging in APROL?

APROL allows a seamless logging of the course on an SFC.

The SFC logging offers you the following basic benefits:

- Recording the state of steps / actions / transitions
- SFC-specific configuration (in the CAE) via control variables
- Recording of data in separate ChronoLog container
- Comfortable analysis via 'SFCLogging report'
  - Opening a project-global 'SFCLogging report'
  - Opening the 'SFC logging report' in the context of the SFC in the SFCViewer
- Extensive filtering possibilities (e.g. name, instance ...)
- Creation of custom-made reports (via ChronoLog query language)

The 'ApCnfLog' module must be present on all controllers where SFC tasks are running in order to be able to log an SFC. Information about how these modules are added to a controller can be found in the APROL product documentation:

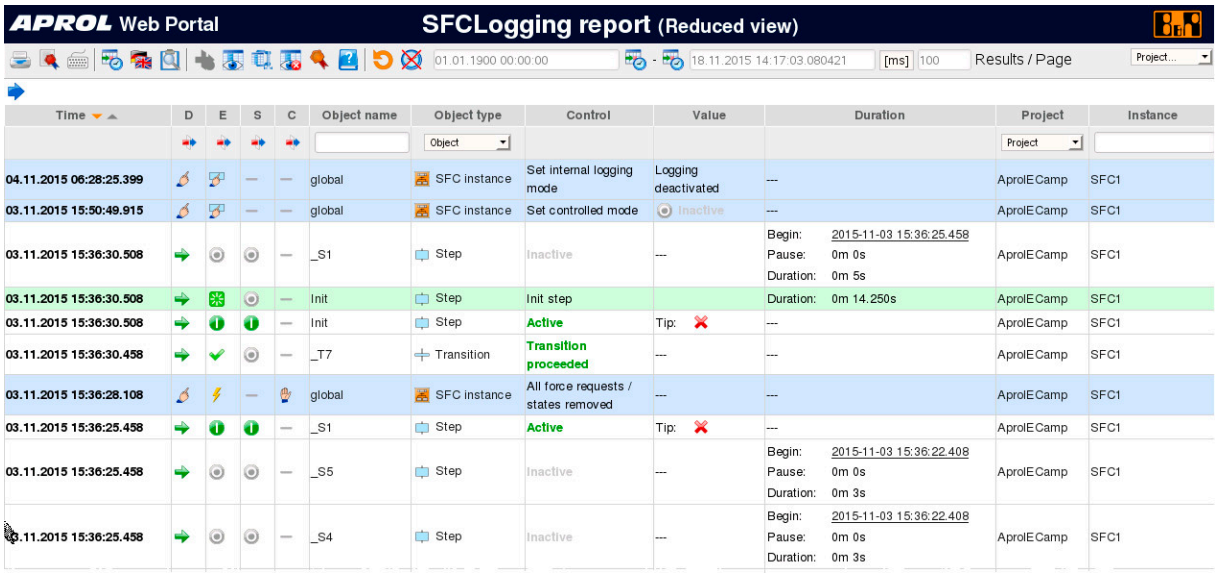
Manual 'B2 Project Engineering', chapter Logic creation with SFC (IEC 61131-3) / Which logging possibilities are supported for SFC? / How is the SFC logging configured in APROL?

The sequence data, the control data, and the SFC context data can be logged selectively. During the creation of SFCs or SFC blocks, an '**SFCLoggingMode**' control variable is offered for setting the logging mode; this turns on the logging of the data per SFC or SFC block in a **bit-coded** manner.

Further information about the sequence data, control data, and context data can be found in the APROL documentation in **B2 Project Engineering / Logic creation with SFC / Which logging possibilities are supported for SFC?**

The SFC logging report in the context of the SFC

The SFC logging report can be opened via the KDE menu or from the SFCViewer (toolbar) within the context of the SFC being monitored.



Time	D	E	S	C	Object name	Object type	Control	Value	Duration	Project	Instance
						Object				Project	
04.11.2015 06:28:25.399					global	SFC instance	Set internal logging mode	Logging deactivated	---	AprolECamp	SFC1
03.11.2015 15:50:49.915					global	SFC instance	Set controlled mode	Inactive	---	AprolECamp	SFC1
03.11.2015 15:36:30.508					_S1	Step	Inactive	---	Begin: 2015-11-03 15:36:25.458 Pause: 0m 0s Duration: 0m 5s	AprolECamp	SFC1
03.11.2015 15:36:30.508					Init	Step	Init step		Duration: 0m 14.250s	AprolECamp	SFC1
03.11.2015 15:36:30.508					Init	Step	Active	Tip:	---	AprolECamp	SFC1
03.11.2015 15:36:30.458					_T7	Transition	Transition proceeded	---	---	AprolECamp	SFC1
03.11.2015 15:36:28.108					global	SFC instance	All force requests / states removed	---	---	AprolECamp	SFC1
03.11.2015 15:36:25.458					_S1	Step	Active	Tip:	---	AprolECamp	SFC1
03.11.2015 15:36:25.458					_S5	Step	Inactive	---	Begin: 2015-11-03 15:36:22.408 Pause: 0m 0s Duration: 0m 3s	AprolECamp	SFC1
03.11.2015 15:36:25.458					_S4	Step	Inactive	---	Begin: 2015-11-03 15:36:22.408 Pause: 0m 0s Duration: 0m 3s	AprolECamp	SFC1

Figure 54: The SFC logging report

5.4.6 AuditTrail report for the SFC

The AuditTrail report can be opened, amongst others, via the **'Reports / AuditTrail'** menu item. A comfortable filtering is possible with the 'SFC intervention' action group and the selected switching operations.

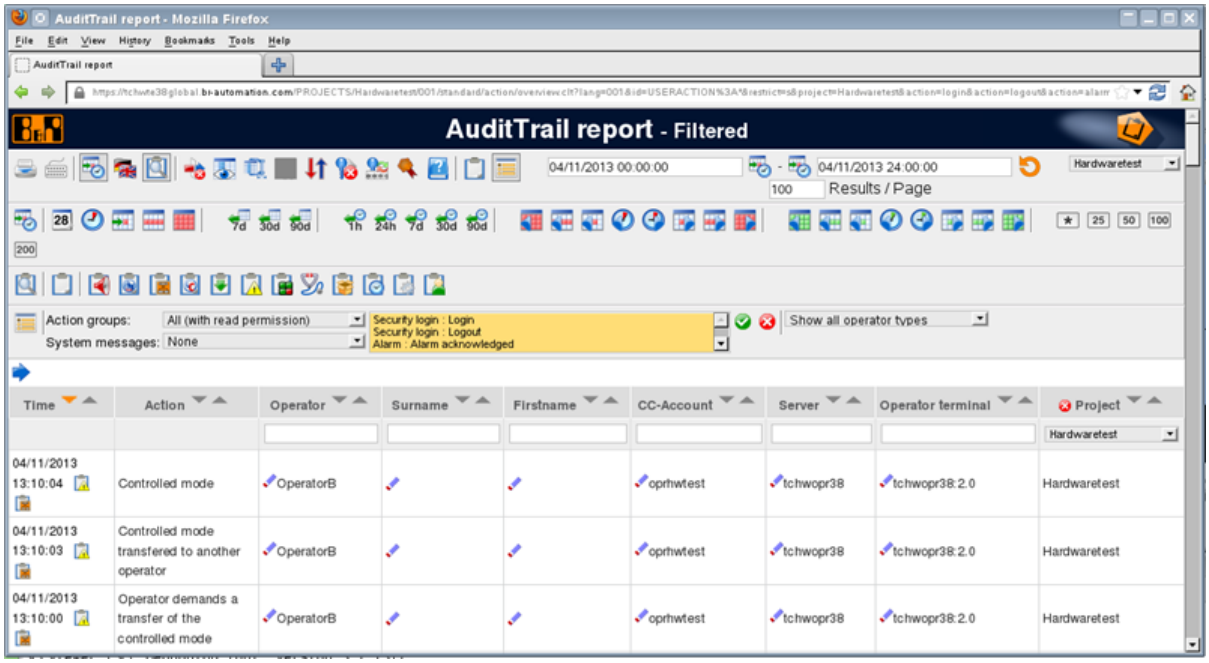


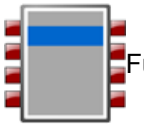
Figure 55: AuditTrail report of the SFC interventions

6 ST in library engineering

6.1 Introduction and editor functions

The following project parts can be created in APROL libraries to introduce a structure and for a program-comprehensive re-usage of the ST code.


Functions (ST)


Function blocks (ST)

These functions and function blocks contain a block layout although this is not necessary for the direct use in the ST program. The ST functions and ST function blocks can also be used within a CFC via graphic programming due to the block layout.


Apart from specifying the master data and the block view, the ST editor offers the same configurations as that for ST programs in the project. For that, see 4.1 "Introduction and editor functions".

Only the tabs which are different to those in the ST program will be described in the following.

6.2 Creating an ST Function Block

The following steps must be carried out in the scope of creating ST functions and function blocks:

Then there is an example for creating a function block (ST). The creation of a function written in ST is also possible.

Step	Configuration
1	Create ST function block
2	Configure master data
3	Write ST code
4	Declare inputs and outputs

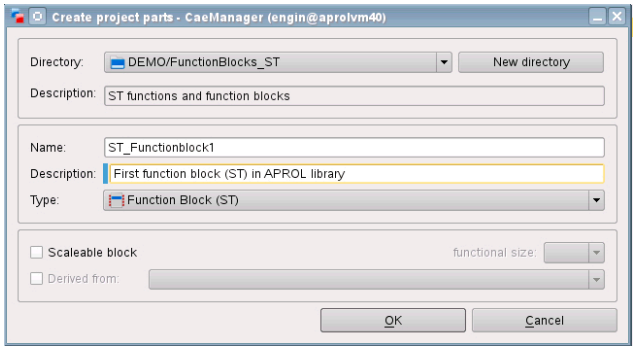


Figure 56: Create ST function block

Navigation: CaeManager / CAE library / File / New / Function block (ST)	
Name	<Block name>
Confirm with [OK]	

6.3 The master data of an ST function block

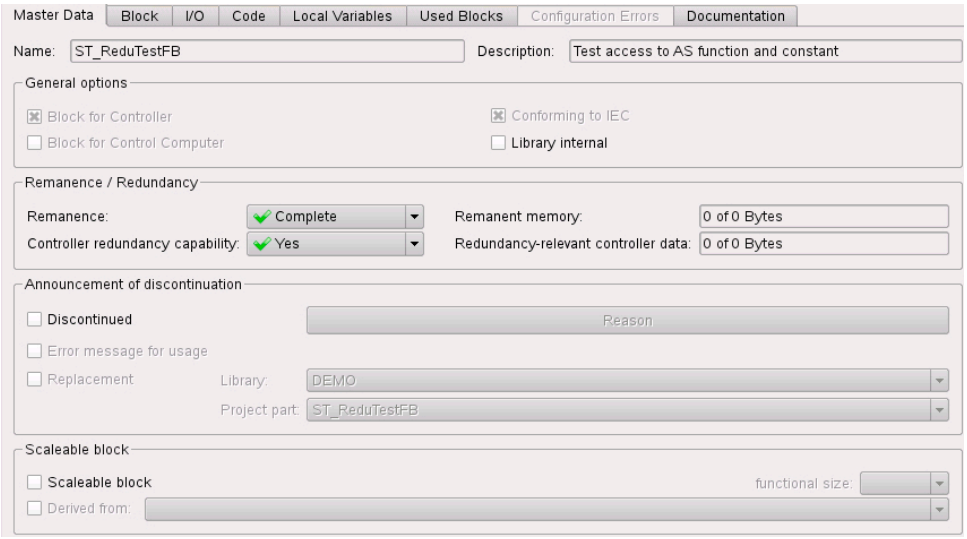


Figure 57: The Master Data

The following library-specific settings are available in the first ST editor tab, master data.

- An ST project part always requires a controller and is always IEC conform and therefore these options are always preset. One can only mark the project part as being library internal.
- Remanence: The entire block - all block outputs - can be marked as being remanent here.
- The respective redundancy-capable selection must be made here, so that the block can be used on a remanent controller.
- The block can be cancelled - with error message and substitute block - underneath, as well as setting the scalability if required.

6.4 Use of pins and variables

Variables can be used in many different ways in the function block (ST).

- As inputs and outputs in the block view
- As inputs and outputs in the I/O view
- As local variables

Inputs and outputs in the block view

Input and output pins can be specified as usual in APROL, in the 'Block' tab, thus creating the block layout.

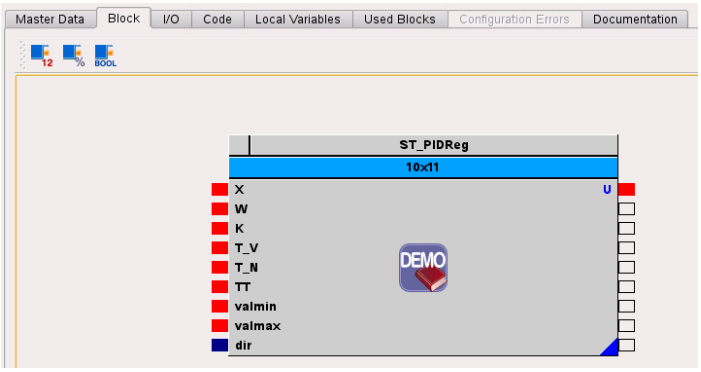


Figure 58: Block view of a function block (ST)

Select the **'Create New Pin' context menu** on a free part of the block in order to create a new pin. You can make more space with drag & drop on the bottom edge of the block if there is no free space available.

Inputs and outputs in the I/O view

All block input and output pins are listed in the **'I/O'** tab. New pins can be created via the **'Insert new pin'** context menu.

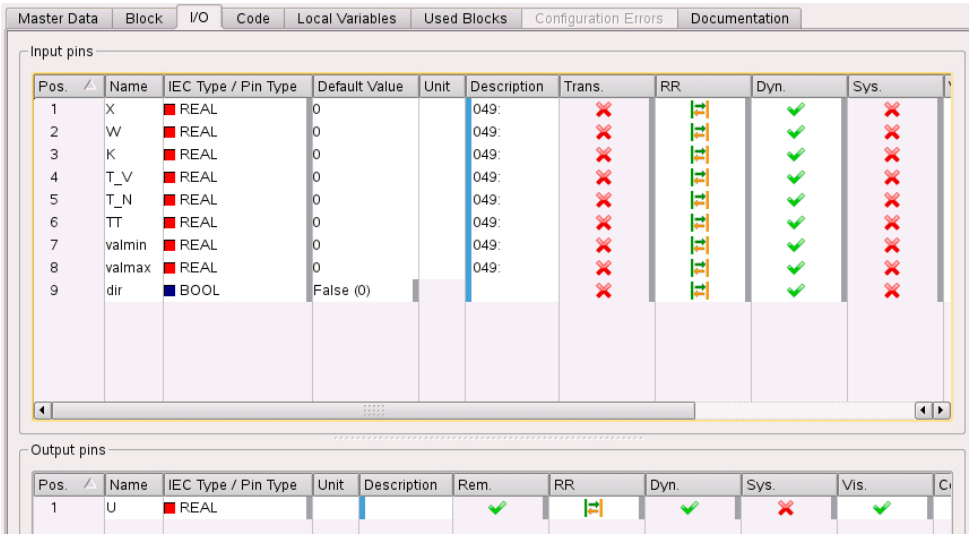


Figure 59: I/O view of a function block (ST)

The functionalities in this tab are the same as those in the previous **'Block'** tab.

Local variables

Variables which are used in the ST code and are not already defined as inputs or outputs are created automatically during the process of saving or the manual synchronization. They are listed here and are furnished with other properties.

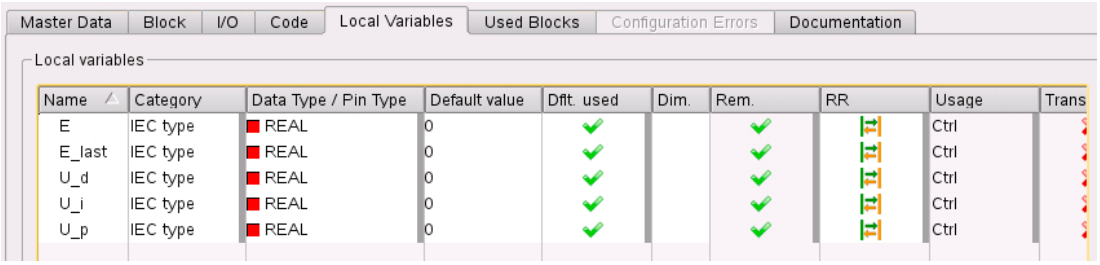


Figure 60: 'Local variables' tab of a function block (ST)

It is possible to let all tabs appear as a separate open window with the **'Separate Window' context menu**. This allows you to maintain an overview of the CAE engineering.

The way to work with the ST editor can be found is mentioned before in the training module, as it is now possible to work with the editor, similar to the project context. See section Working with the ST editor.

6.5 Use of Automation Studio blocks (AS) in an ST function block

It is not only possible to use the IEC basis data types (list of the available data types on the controller) for the local variables, but also the definitions of a structures or external function block types.

The configuration that is necessary for this is made in the 'Category' column in the local variables.

In the following, the use of the AS function block 'RF_TRIG' from the 'standard' AS library is demonstrated.

An overview of the functions and function blocks can be found in the Automation Studio help (Programming / Debugging & Diagnosis / Libraries).

The respective AS library must be linked in the properties of the CAE library, in the **'AR OS versions'** tab, so that functions and function blocks from the **Automation Studio** can basically be used.

After the [**Dependencies**] button has been pressed, the selection of the system modules must be opened (via [**System modules**] button) in the following 'AR OS version dependencies' dialog, and the desired AS library (**'standard' in the example**) must be selected and inserted to the 'Module dependencies'.

Subsequently, a 'Build all (library)' must be carried out.

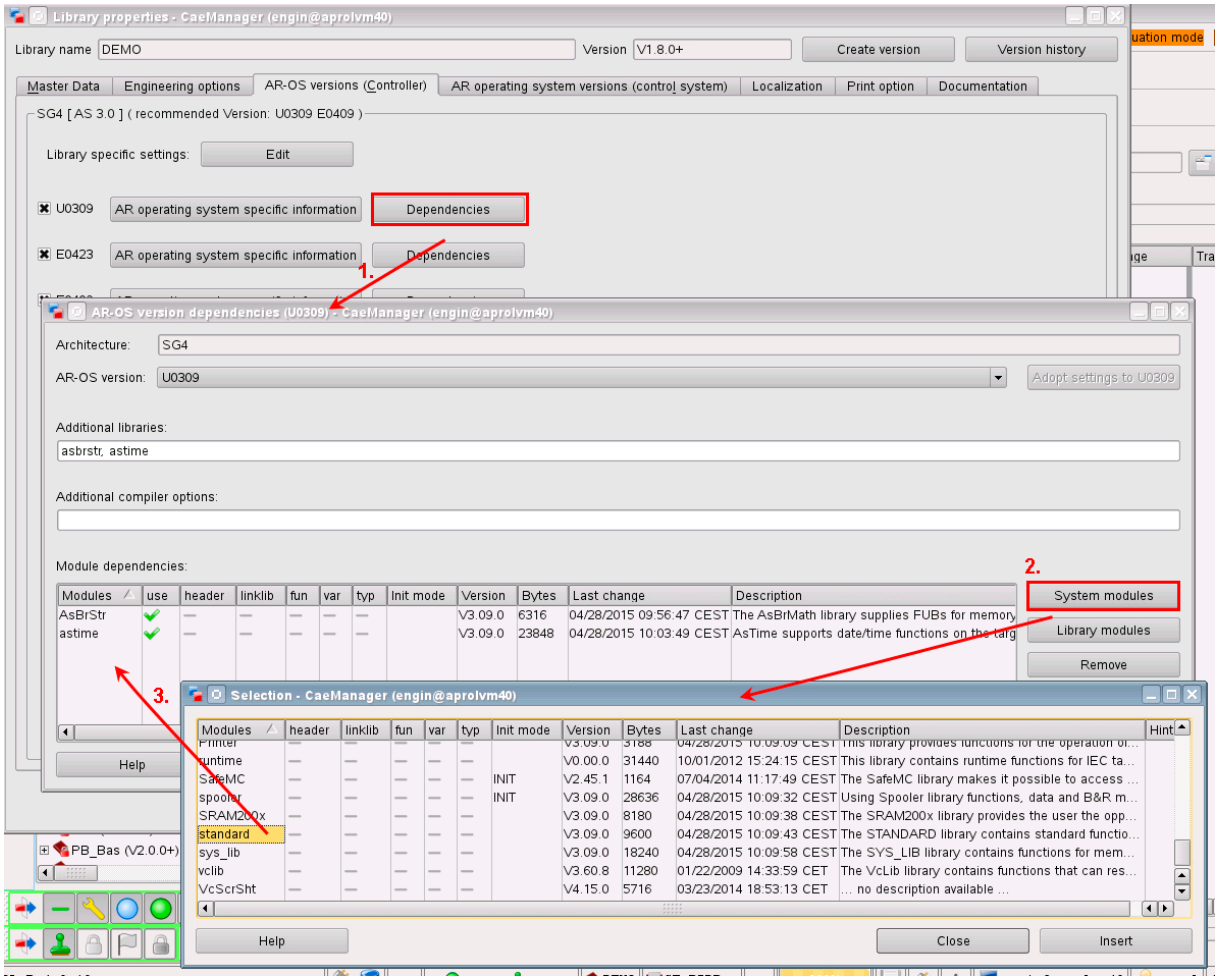


Figure 61: Inserting the 'standard' AS library (Properties of the CAE library)

The following configuration must be made in the **'Local variables'** tab.

Column	
Name	Specification of the instance name of the function block as it should be called later on in the code. *In the example: 'RF_TRIG_01'*
Category	Selection of the 'IEC-FB' entry from the combo box.
Pin type / data type	Input of the function block name. *In the example: 'RF_TRIG'*
Description	A description of the variable can be entered by the user here.

No further entries or changes are necessary in the other columns for this purpose.

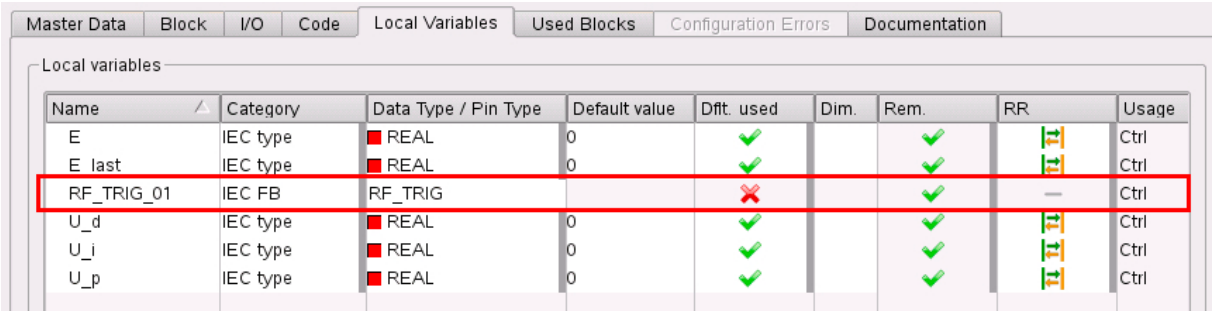


Figure 62: Creation of the AS function blocks in the 'Local variables' tab

Hereupon, the previously configured instance of the AS function block can be called in the code.

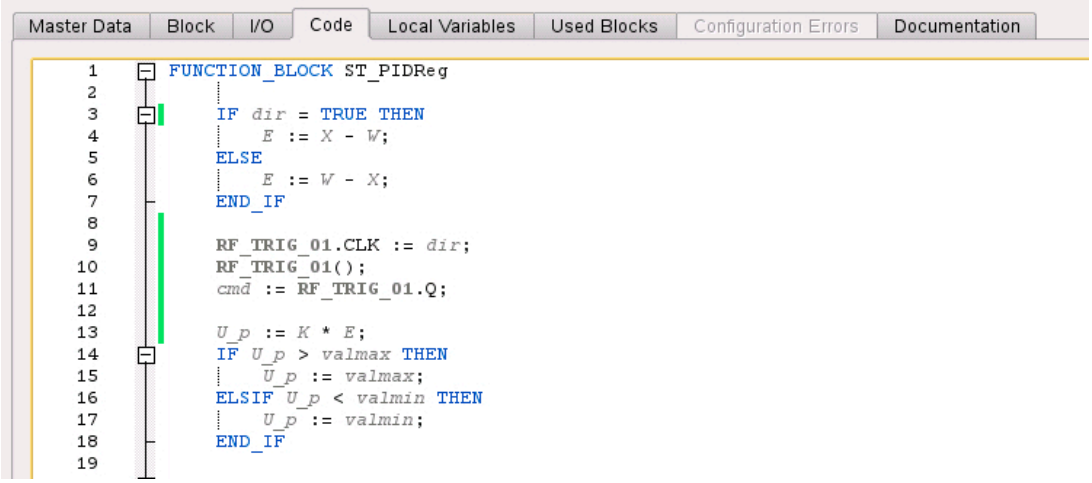


Figure 63: Calling the 'RF_TRIG_01' function block in the code

The use of existing CAE library blocks, see Calling existing CAE blocks in ST, is also possible in ST library blocks.

6.6 Practice task - crane

Task: Crane total load

A crane can lift five loads at once. The load receptors are each connected to an analog input and return values in the range of 0 to 32767. In order to calculate the total load as well as the average value, the individual loads must first be added up and divided by the number of load sensors. Create a solution for this exercise using a FOR loop.

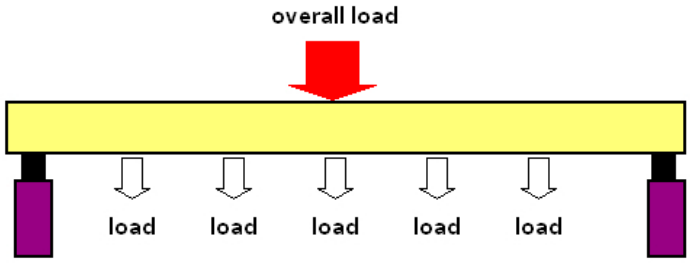


Figure 64: Crane with five loads

Declaration:

```
VAR
    aWeights : ARRAY [0..4] OF INT;
    iCnt : USINT;
    sumWeight : DINT;
    avgWeight : INT;
END_VAR
```

Table 4: Suggestion for the variable declaration

Create an ST function block which contains arrays to solve the task. Array variables can only be local variables - simply enter the desired length in the "Dimensions" column.

7 SFC in library engineering

7.1 Introduction and editor functions

The following project parts can be created in an APROL library in order to introduce a structure and for a program-spanning re-usage of the ST code.


Function blocks (SFC)

These function blocks contain a block layout although this is not necessary for the direct use in the ST and SFC programs. The SFC function blocks can also be used within a CFC via graphic programming due to the block layout.

Apart from specifying the master data and the block view, the SFC editor offers the same configurations as that for SFC programs in the project. For that, see Introduction and editor functions.

Only the tabs which are different to those in the SFC program editor will be described in the following.

7.2 Creating an SFC Function Block

The following steps must be carried out in the scope of creating SFC function blocks:

Step	Configuration
1	Create SFC function block
2	Configure master data
3	Create SFC network
4	Declare inputs and outputs

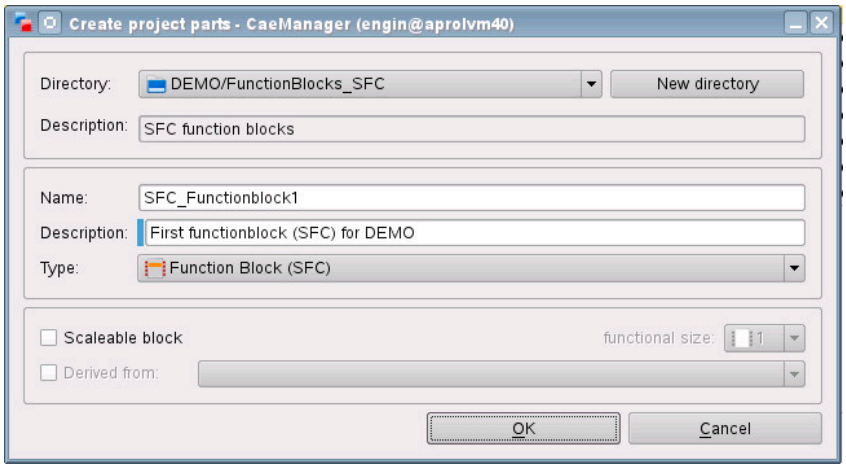


Figure 65: Create SFC function block

Navigation: CaeManager / CAE library / File / New / Function block (SFC)	
Name	<Block name>
Confirm with [OK]	

7.3 The master data of an SFC function block

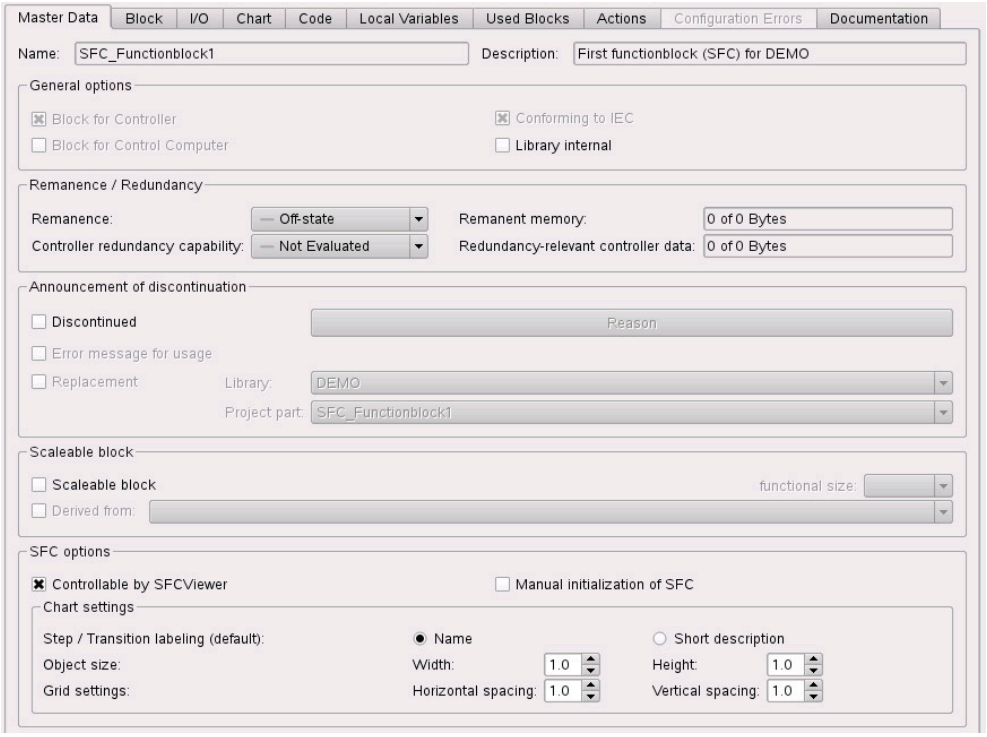


Figure 66: The Master Data

The following library-specific settings are available in the first SFC editor tab, master data.

- An SFC project part always requires a **controller** and is always **IEC conform** and therefore these options are always preset. One can only mark the project part as being library internal.
- **Remanence**: The entire block - all block outputs - can be marked as being remanent here.
- The respective **redundancy-capable** selection must be made here, so that the block can be used on a remanent controller.
- The block can be **cancelled** - with error message and substitute block - underneath, as well as setting the **scalability** if required.
- The SFC is prepared for a later influencing of its course in the SFCViewer with **'Controllable by SFCViewer'**. Furthermore, the **manual initialization** can be configured here.
- Furthermore, settings can be made for the **size** of the SFC elements and the **layout** of the SFC network.

7.3.1 The manual initialization of an SFC function block

A manual initialization means that an SFC is not started automatically during runtime, but must be started with an explicit trigger.

The **'Manual initialization'** of the function block can be configured in the 'Master data' tab with the respective checkbox.

I.e. the function block only starts after the 'SFCInit' / 'SFCReset' input in a CFC, SFC, or ST program is set from 'TRUE' to 'FALSE' and back to 'TRUE' (rising and negative edge).

If the 'Manual initialization' is chosen, the 'SFCInit' / 'SFCReset' input pin is marked so that a connection is required.

The pin names are displayed in bold if the 'Manual initialization ' has been set.

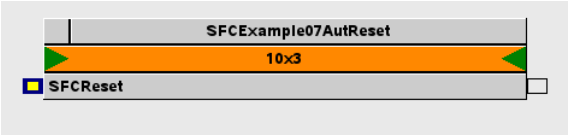


Figure 67: 'SFCReset' input pin

If an SFC function block is initialized automatically it can be recognized with the green 'SFCInit' or 'SFCReset' pin names in the 'Block view' of the block, as well as in the views when being used in a hyper macro or CFC.

The use of the 'Controllable by SFCViewer' option is marked by two 'green triangles' in the block layout.

The activation of the option allows control possibilities in the SFCViewer and SFC editor to be chosen.

An automatic initialization is the default setting when creating a new SFC function block.

7.4 Graphic configuration of the function block

The graphic configuration of the SFC network is done in the 'Chart' tab.

Information about the graphic configuration and the creation and use of actions can be found in the chapter "The graphic configuration of an SFC network" later in this document.

7.5 Use of pins and variables

Variables can be used in many different ways in the function block (SFC).

- As inputs and outputs in the block view
- As inputs and outputs in the I/O view
- As local variables

Inputs and outputs in the block view

Input and output pins can be specified as usual in APROL, in the 'Block' tab, thus creating the block layout.

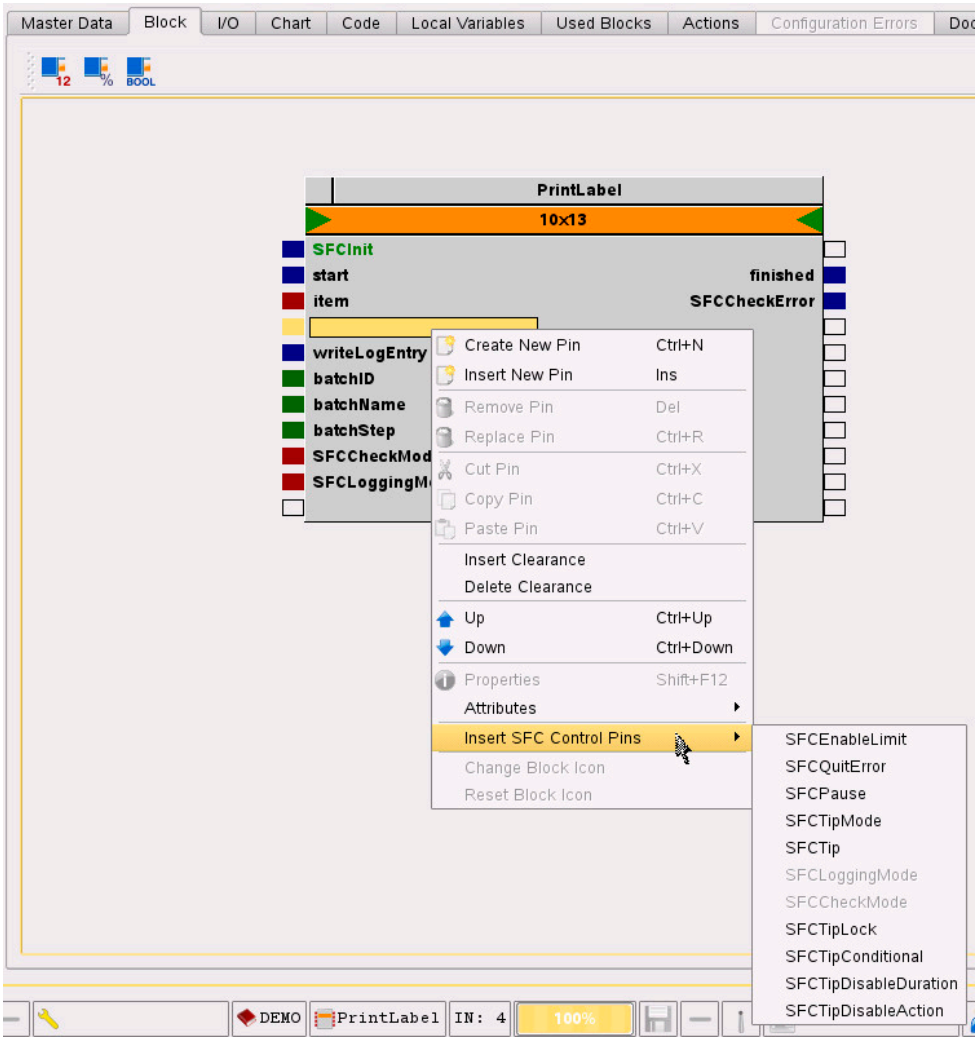


Figure 68: Block view of a function block (ST)

Select the **'Create New Pin' context menu** on a free part of the block in order to create a new pin. You can make more space with drag & drop on the bottom edge of the block if there is no free space available.

Special **'SFCContext' pin types** are used for the context data of SFCs. Select this with **Context menu 'Create New Pin' / Radio button 'Pin type' / [...]**

Control variables are assigned with the **'Insert Control Variables'** context menu.

Inputs and outputs in the I/O view

All block input and output pins are listed in the **'I/O'** tab. New pins can be created via the 'Insert new pin' context menu.

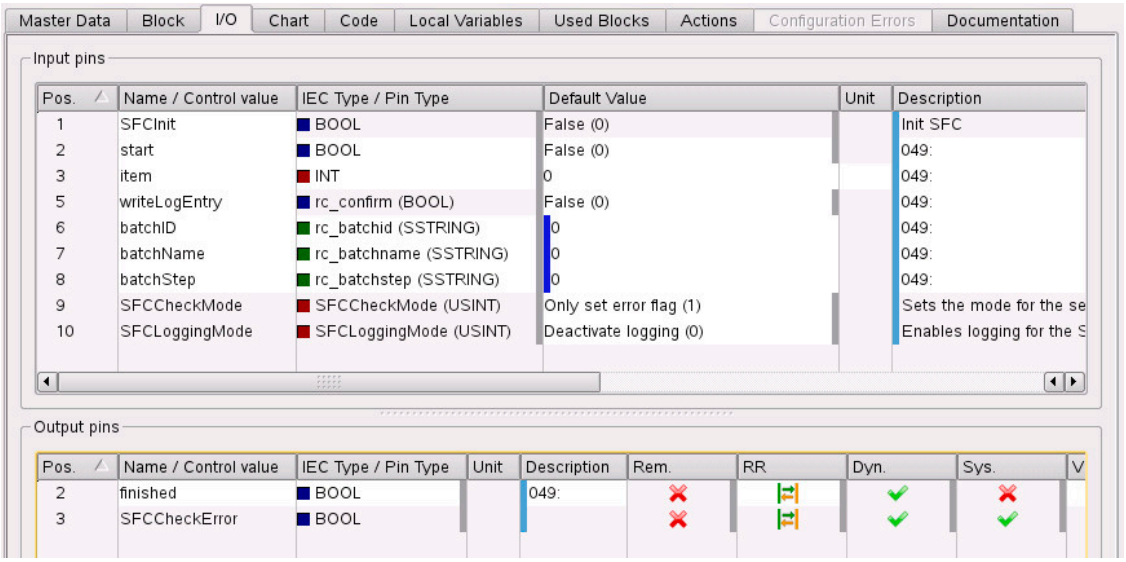


Figure 69: I/O view of a function block (ST)

The functionalities in this tab are the same as those in the previous **'Block'** tab.

Local variables

Variables which are used in the ST code and are not already defined as inputs or outputs are created automatically during the process of saving or the manual synchronization. They are listed here and are furnished with other properties.

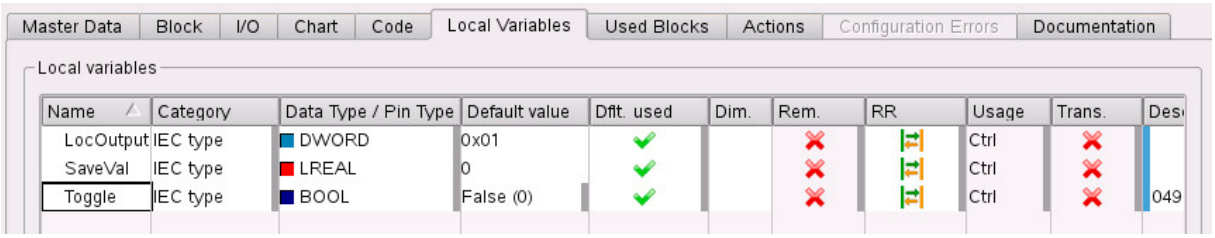


Figure 70: 'Local variables' tab of a function block (ST)

It is possible to let all tabs appear as a separate open window with the **'Separate Window' context menu**. This helps you well to maintain an overview of the CAE engineering.

7.6 Calling functions and function blocks

In the context of a library, you can create local functions and function blocks, CAE library blocks, and Automation Studio function blocks.

The ST editor and the SFC editor have the same behavior with regard to this functionality.

Read the corresponding chapter in the structured text section in order to create and use **local functions and function blocks** : Local Functions and Function Blocks

In the further sections for structured text, you will learn how to use **existing library blocks** (see chapter Calling existing CAE blocks in ST) in a function block (SFC) and **Automation Studio function blocks** (see chapter Use of Automation Studio blocks (AS) in an ST function block).

7.7 Practice task - Container control

The fill level and temperature of the water in an aquarium need to be controlled. This concerns two processes which should be viewed as being separate.

A heating element is used to increase the temperature until it reaches the setpoint. If the setpoint is reached, the heating element will be switched off again.

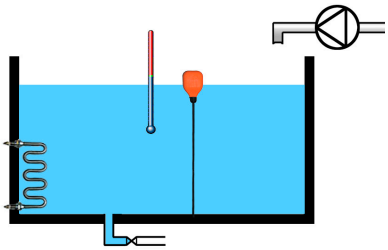


Figure 71: Schematics of the exercise

Control of the fill level is handled using a pump and a floating switch. As soon as the float switch detects a low fill state, a pump should fill water into a container until the fill level is reached. A replenishing time is added to reduce the switching rate of the pump when the water ripples.

SFC Application

Because this concerns two different processes, this can be realized with two parallel branches running simultaneously. The first part of the parallel branch controls the heating, while the second branch is responsible for regulating the fill level.

Implementing the fill level control

The task described before should be solved in one function block (SFC) with one parallel branch. Each branch contains the states which are described in the state diagrams.

1) Creating an SFC Function Block
2) Create the parallel branch.
3) Create the steps and jumps.
4) Declare the transitions.

   The transitions can be used to evaluate the setpoint temperature and the actual temperature. The result must be of data type BOOL

5) Program the actions and switching commands.

   The switch-on and switch-off commands for the heating element and the pump should be implemented using the steps' entry and exit actions.

6) Use an IEC action for the time delay.

   The transition which deactivates the wait step, must be switched with an action which has the qualifier "D". The delay time should be set to 5 seconds.

7) Testing the sequence in the automatic, tip, or force mode

One possible solution can be found in the appendix

8 Summary

What have we achieved so far?

We now have knowledge of the 'Structured Text' and 'Sequential Function Chart' programming languages. Both languages are norm high-level programming languages, are used on programmable logic controllers, and offer a wide range of functionality.

The **Structured Text** contains all the tools necessary for an application. This programming language is especially powerful when using arithmetic functions and formulating mathematical calculations.

The **Sequential Function Chart** builds a clear and good structured programming language with its graphical user interface. State diagrams and state machines can be realized in the form of steps and transitions very well in SFC. SFC programs can be diagnosed and maintained well with the use of the many diagnostic features.

Repeated actions and function blocks round up the scope of functions in ST and SFC.

The combination with other IEC programs, such as Continuous Function Chart or Structured Text, makes SFC a powerful language.

Figure 72: The extensive diagnostic tool: The SFCViewer

9 Appendix

9.1 Example solution of the ST program

Task: Fill level check part 1

Declaration:

Local Variables

Name	Used	Category	Pin Type/Data Type
Alarm	✔	IEC type	BOOL
DoMeasure	✔	IEC type	BOOL
LevelAnalog	✔	IEC type	INT
LevelHigh	✔	IEC type	BOOL
LevelLow	✔	IEC type	BOOL
LevelOk	✔	IEC type	BOOL
LevelPercent	✔	IEC type	USINT

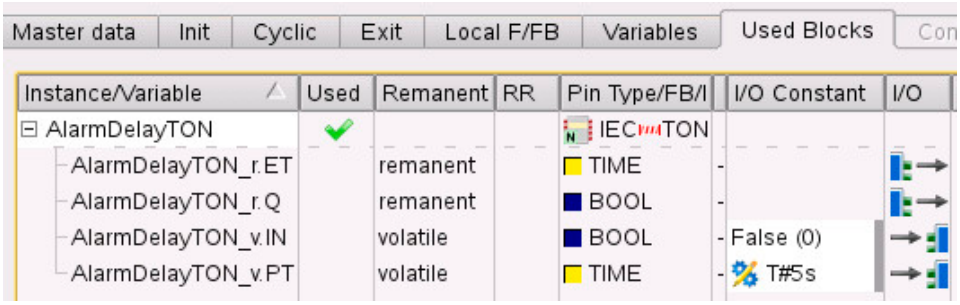
Program code:

```
(*scaling the analog input to percent*)
LevelPercent := INT_TO_USINT((LevelAnalog / 32700)*100);

IF DoMeasure THEN

    CASE LevelPercent OF
        0:
            Alarm := TRUE;
        1..24:
            LevelLow := TRUE;
        25..90:
            LevelOk := TRUE;
        ELSE
            LevelHigh := TRUE;
    END_CASE
ELSE
    Alarm := 0;
    LevelLow := 0;
    LevelOk := 0;
    LevelHigh := 0;
END_IF
```

Table 5: CASE statement for querying values and value ranges

Task: Fill level check part 2

Used Blocks:



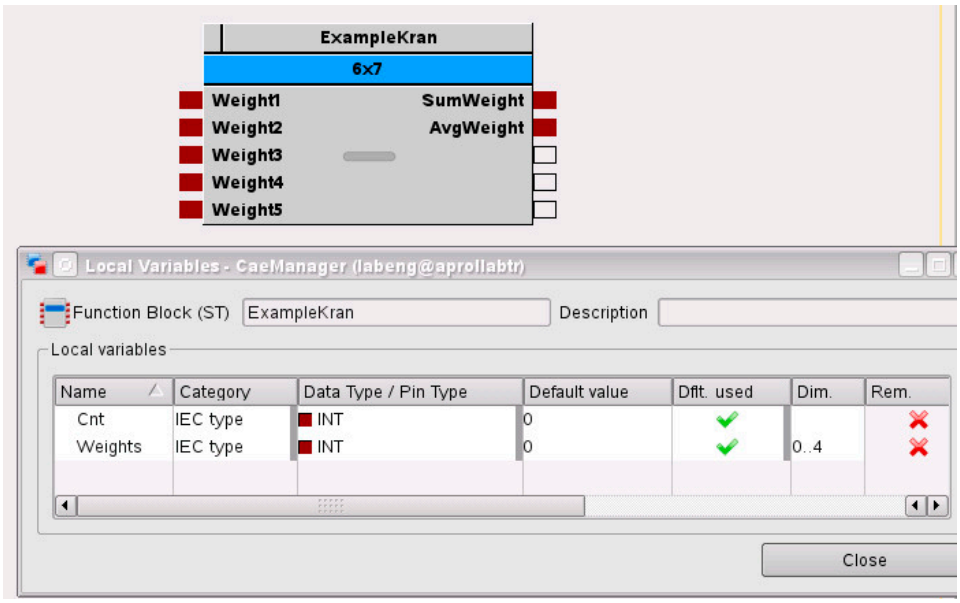
Program code:

```
(*scaling the analog input to percent*)
LevelPercent := INT_TO_USINT((LevelAnalog / 32700)*100);
IF DoMeasure THEN
    CASE LevelPercent OF
        0:
            AlarmDelayTON_v.IN := TRUE;
            IEC61131_3_TON( AlarmDelayTON_v, AlarmDelayTON_r );
            Alarm := AlarmDelayTON_r.Q;
        1..24:
            LevelLow := TRUE;
        25..90:
            LevelOk := TRUE;
        ELSE
            LevelHigh := TRUE;
    END_CASE
ELSE
    Alarm := 0;
    AlarmDelayTON_v.IN := 0;
    IEC61131_3_TON( AlarmDelayTON_v, AlarmDelayTON_r );
    LevelLow := 0;
    LevelOk := 0;
    LevelHigh := 0;
END_IF
```

Table 6: Part 1 extended with the use of TON block

Task: Crane total load

Declaration:



Program code:

```
FUNCTION_BLOCK ExampleKran

    Weights[0] := Weight1;
    Weights[1] := Weight2;
    Weights[2] := Weight3;
    Weights[3] := Weight4;
    Weights[4] := Weight5;

    FOR Cnt := 0 TO 4 DO
        SumWeight := SumWeight + Weights[Cnt];
    END_FOR

    AvgWeight := DINT_TO_INT(SumWeight / 5);

END_FUNCTION_BLOCK
```

Table 7: FOR statement, adding up weights

9.2 Example solution of the SFC programs

Example solution of the dispersion mixer

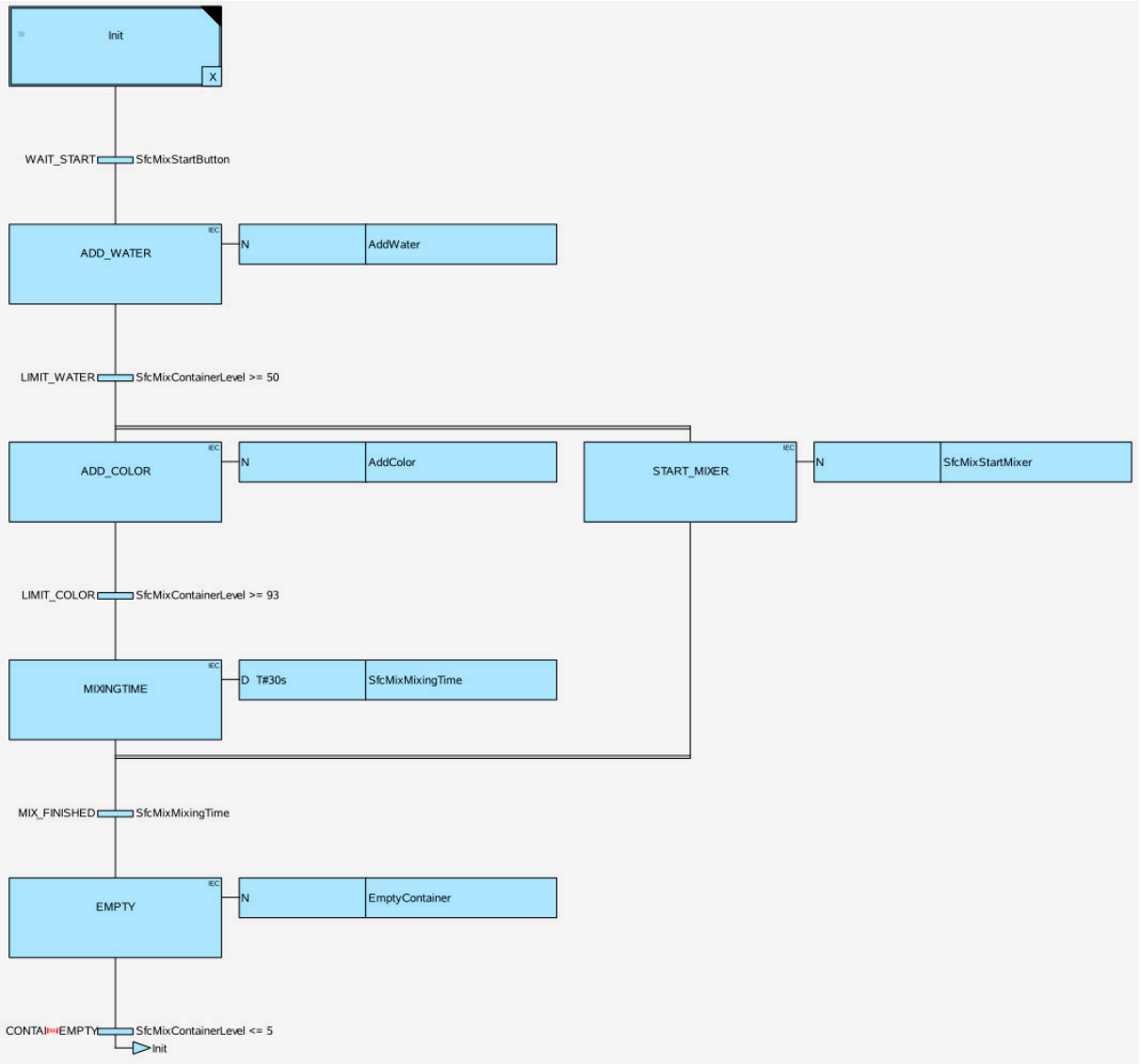


Figure 73: SFC chart Mixer

Exercise solution: Fill control

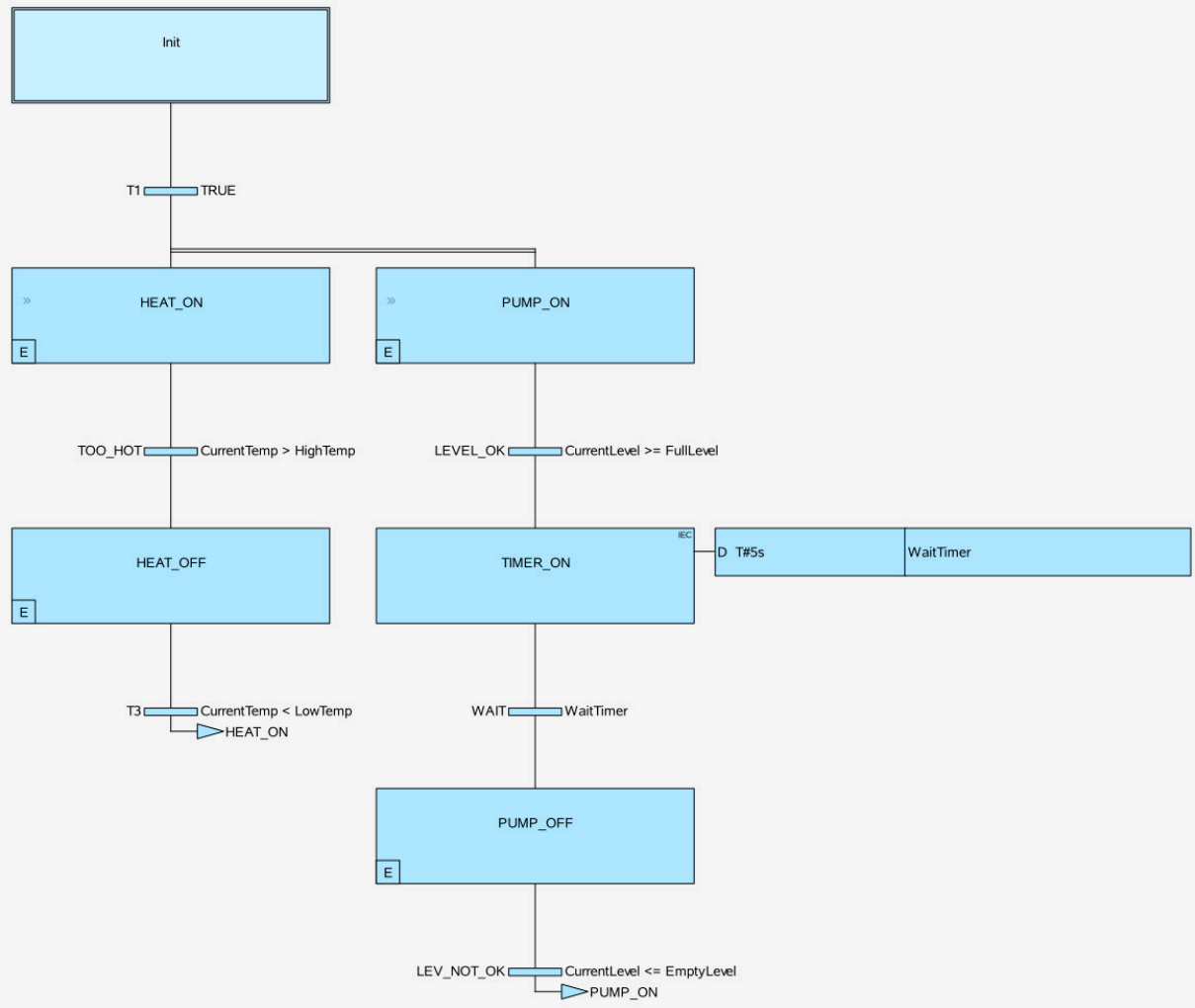


Figure 74: SFC chart Container control

Seminars and Training Modules

The Automation Academy provides targeted training courses for our customers as well as our own employees.
At the Automation Academy, you'll develop the skills you need in no time!
Our seminars make it possible for you to improve your knowledge in the field of automation engineering.
Once completed, you will be in a position to implement efficient automation solutions using B&R technology. This will make it possible for you to secure a decisive competitive edge by allowing you and your company to react faster to constantly changing market demands.

Automation Studio seminars and training modules

Programming and configuration	Diagnostics and service
SEM210 – Basics SEM246 – IEC 61131-3 programming language ST* SEM250 – Memory management and data storage SEM410 – Integrated motion control* SEM441 – Motion control: Electronic gears and cams** SEM480 – Hydraulics** SEM1110 – Axis groups and path-controlled movements** SEM510 – Integrated safety technology* SEM540 – Safe motion control*** SEM610 – Integrated visualization*	SEM920 – Diagnostics and service for end users SEM920 – Diagnostics and service with Automation Studio SEM950 – POWERLINK configuration and diagnostics* If you don't happen to find a seminar on our website that suits your needs, keep in mind that we also offer customized seminars that we can set up in coordination with your sales representatives: SEM099 – Individual training day Please visit our website for more information****.****: **www.br-automation.com/academy**

Overview of training modules	
TM210 – Working with Automation Studio TM213 – Automation Runtime TM223 – Automation Studio Diagnostics TM230 – Structured Software Generation TM240 – Ladder Diagram (LD) TM241 – Function Block Diagram (FBD) TM242 – Sequential Function Chart (SFC) TM246 – Structured Text (ST) TM250 – Memory Management and Data Storage TM400 – Introduction to Motion Control TM410 – Working with Integrated Motion Control TM440 – Motion Control: Basic Functions TM441 – Motion control: Electronic gears and cams TM1110 – Integrated Motion Control (Axis Groups) TM1111 – Integrated Motion Control (Path Controlled Movements) TM450 – Motion Control Concept and Configuration TM460 – Initial Commissioning of Motors TM500 – Introduction to Integrated Safety TM510 – Working with SafeDESIGNER TM540 – Integrated Safe Motion Control	TM600 – Introduction to Visualization TM610 – Working with Integrated Visualization TM630 – Visualization Programming Guide TM640 – Alarm System, Trends and Diagnostics TM670 – Visual Components Advanced TM920 – Diagnostics and service TM923 – Diagnostics and Service with Automation Studio TM950 – POWERLINK Configuration and Diagnostics TM280 – Condition Monitoring for Vibration Measurement TM480 – The Basics of Hydraulics TM481 – Valve-based Hydraulic Drives TM482 – Hydraulic Servo Pump Drives TM490 – Printing Machine Technology In addition to a printed version, our training modules are also available on our website for download as electronic documents (login required): Visit our website for more information: **www.br-automation.com/academy**

Process control seminars and training modules

Process control standard seminars	Process control training modules
SEM841 – Process Control Training: Basic 1 SEM842 – Process Control Training: Basic 2 SEM890 – Advanced Process Control Solutions	TM800 – APROL System Concept TM811 – APROL runtime system TM812 – APROL Operator Management TM813 – APROL XML Queries and Audit Trail TM830 – APROL Project Engineering TM890 – The Basics of LINUX Visit our website for more information: **www.br-automation.com/academy**

* SEM210 - Basics is a prerequisite for this seminar.
** SEM410 - Integrated motion control is a prerequisite for this seminar.
*** SEM410 - Integrated motion control and SEM510 - Integrated safety technology are prerequisites for this seminar.
**** Our seminars are listed in the Academy\Seminars area of the website.
**** Seminar titles may vary by country. Not all seminars are available in every country.