Sequential Function Chart (SFC)

TM242

Perfection in Automation
www.br-automation.com

Requirements

Training modules:	TM210 – The Basics of Automation Studio TM246 – Structured Text (ST)
Software	Automation Studio 3.0.90
Hardware	---

TABLE OF CONTENTS

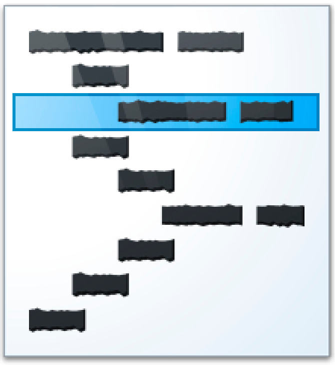
1 INTRODUCTION

Sequential Function Chart is also referred to by its acronym, SFC. SFC is a visual programming language arranged around a series of steps. This not only improves clarity, but also makes it easy to troubleshoot programs.

SFC makes it possible to formulate and implement successive and parallel sequences that arise in the course of software design.

This training module will cover the elements used in the Sequential Function Chart programming language.

Examples are included to provide a deeper level of understanding. The programming that takes place in steps and transitions is written using Ladder Diagram and Structured Text.


Sequential Function Chart (SFC)

Additional information regarding the individual sections covered here can be found in the Automation Studio online help documentation.

> ? Programming \ Programs \ Sequential Function Chart (SFC)
>
> Programming \ Editors \ Graphic editors \ Sequential Function Chart editor

1.1 Training module objectives

In this training module, you will learn...

- ... How the Sequential Function Chart language works
- ... How to use steps and transitions correctly
- ... How to use action steps
- ... Rules for programming
- ... Troubleshooting options

2 SEQUENTIAL FUNCTION CHART

2.1 General information

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 particularly useful whenever program steps and defined transitions to the individual steps are needed. It is even possible to execute parallel sequences within a single program.

Sequential Function Chart properties

Sequential Function Chart is included in the IEC[1] The steps can call actions that are programmed in other programming languages such as Ladder Diagram and Structured Text.

Sequential Function Chart is extremely suitable for use with state machines. Because it supports parallel processing (5.2 "Parallel branches"), it is possible to implement complex designs in a way that is clear and manageable. The visual structure of the program is also advantageous for documentation and troubleshooting sequences.

When is SFC used?

Whenever sequences can be described directly, it is possible to use the SFC programming language to implement them. State machines can be "redrafted" and implemented in SFC; the opposite is usually not possible.

Useful information

GRAFCET (acronym from the French "**GRA**phe **F**onctionnel de **C**ommande **E**tapes/**T**ransitions"), defined in EN 60848, is a standardized language for representing sequences. It is primarily used in automation but can also be found in process engineering.

Specified in IEC 61131-3, Sequential Function Chart is mentioned specifically in the GRAFCET standard as one possible way to implement a GRAFCET plan (*de.wikipedia.org/wiki/GRAFCET*).

Creating large applications

When designing larger applications, it is necessary to consider everything in advance in order to be able to really take advantage of the characteristics that SFC was designed to handle – controlling sequences.

It is recommended that the actual sequence be implemented in the SFC program. Machine sub-functions such as evaluating I/O data, scaling values or controlling actuators should be handled in an additional program module.

[1] The IEC 61131-3 standard is the only valid international standard for programming languages used on programmable logic controllers. This standard also includes Ladder Diagram, Instruction List, Structured Text and Function Block Diagram.

The SFC program can then call these sub-functions via defined interfaces and retrieve the status of each function.

This type of structure is advantageous since a machine sub-function may change (e.g. a new sensor or different drive) but the sequence itself remains unchanged. This kind of division is a good way to allow both the sequence chain and the machine's sub-functions to be tested and expanded as needed.

In the rest of this document, Sequential Function Chart will be referred to by its acronym, SFC.

2.2 The basic function of SFC

The basic components of SFC are called steps and transitions. Steps are shown as rectangles with different types of borders. Transitions are indicated by horizontal lines that intersect the connections between steps.

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.

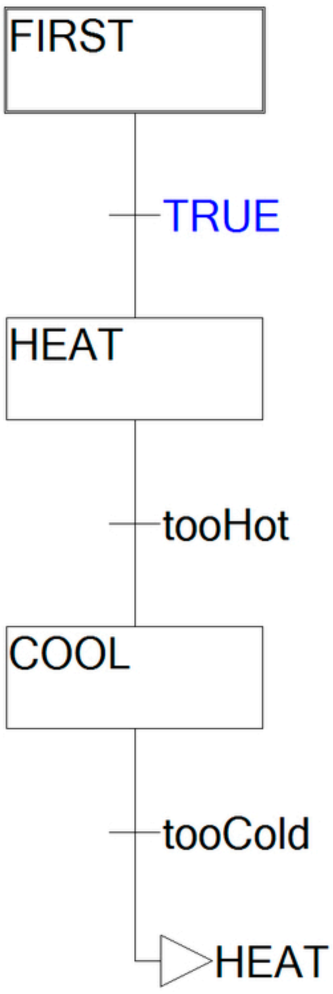
The example shown here illustrates a program with three steps, the associated transitions and a jump.

The first time the SFC program is called, the "FIRST" step is scanned. This step is identified by its double border 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 program then lingers in the "HEAT" step until the "tooHot" transition also becomes 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, on the other hand, are variables of data type BOOL or logical expressions that can be put together in various programming languages.



Sequence with two steps, one jump and an initialization step

A step and a transition form a single unit, with one transition following each step. It is not possible for two or more transitions or steps to appear consecutively.

When moving from one step to the next, the new step is executed only when the next program cycle begins.

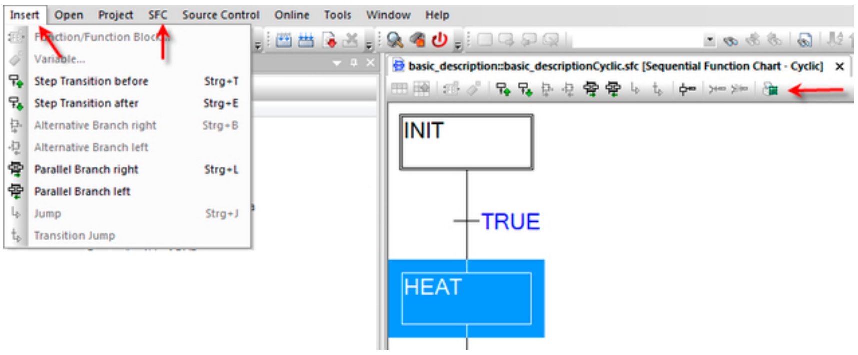
Programming \ Programs \ Sequential Function Chart (SFC)

2.3 Editor functions in SFC

When programming in SFC, it is necessary to insert steps, transitions and jumps.

There are several places from which the SFC editor in Automation Studio can be operated:

- Main menu: Insert
- Main menu: SFC
- Editor toolbar
- Shortcut menu for the SFC program, steps and transitions


Insert
Open
Project
SFC
Source Control
Online
Tools
Window
Help
Function/Function Block
Variable...
Step Transition before Strg+T
Step Transition after Strg+E
Alternative Branch right Strg+B
Alternative Branch left
Parallel Branch right Strg+L
Parallel Branch left
Jump Strg+J
Transition Jump
basic_description::basic_descriptionCyclic.sfc [Sequential Function Chart - Cyclic]
INIT
TRUE
HEAT

Operating the SFC editor using the main menu, the toolbar and shortcut menus

Objects are inserted using the main menu and the toolbar. Object properties can be defined either from the main menu or using the object's shortcut menu. A selected element can be opened by double clicking on it.

It is also possible to zoom in and out of the SFC editor using the Zoom toolbar or the "View" menu in the menu bar.

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ General editor settings

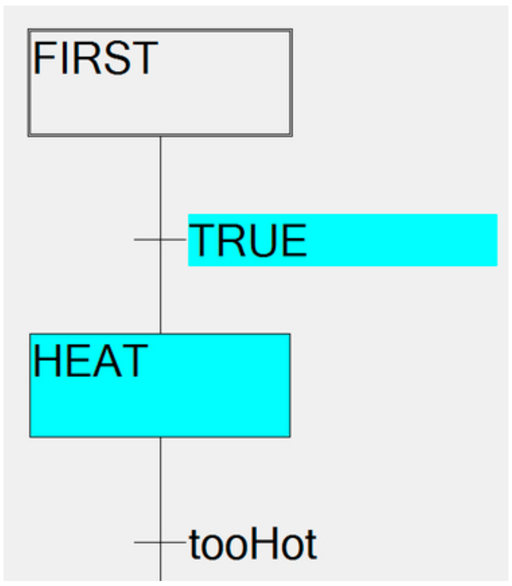
Project management \ The workspace \ Toolbars \ Zoom

Creating your first SFC program

In this exercise, you will create the simple program that was discussed in The basic function of SFC.

1) Insert a new SFC program in the logical view.
   The program should be called "sfc_basic".
2) Insert the step "HEAT".
3) Insert the step "COOL".
4) Name and declare the transitions "tooHot" and "tooCold".
5) Compile the program.
6) Enable monitor mode.
7) Test the program in the variable watch window.
   The program can be tested by manually setting the values of transitions in the variable watch window. When one of the transitions is set, then the SFC program will move to the next active step. Enabling "Powerflow" while monitor mode is active will color active steps and switched transitions.

If Powerflow is enabled, then active steps and transitions are shown in color. Transitions can be set either in the variable watch window or by clicking on it in the SFC program.



The "HEAT" step and the transition after "FIRST" are active.

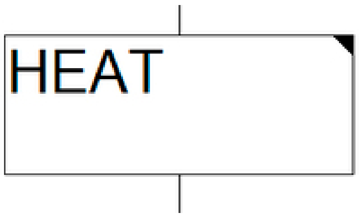
Diagnostics and service \ Diagnostic tool \ Monitors \ Sequential Function Chart editor in monitor mode

3 STEPS

A step can call executable program code or actions. This code is always executed whenever the step is active. A step can be activated, executed and deactivated. Timing can also be monitored for steps.

Steps are represented as rectangles. If a step includes cyclic program code, then a black triangle is shown in its upper right corner.

The contents of a step consist of programs that can be written in one of the IEC programming languages.



Step with executable code

Steps that contain directly executable code are an expansion of the IEC standard. Cyclic actions are usually composed of IEC actions (see Action steps).

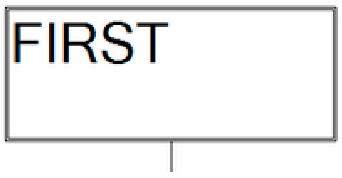
? Programming \ Programs \ Sequential Function Chart (SFC) \ Step

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ Defining SFC cyclic actions

3.1 Initialization steps

The initialization step is executed immediately after the program is started as the first step of the cyclic program. It is important that this not be confused with the initialization subprogram. The initialization step is also executed after a return or after an SFCReset or SFCInit is carried out. When a compound step is active, then the initialization step of the underlying SFC program is activated.

Initialization steps are indicated by a double border.



Initialization step

See also:

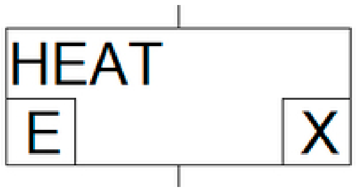
- Compound step
- Using system variables

3.2 Entry and exit actions

Every step except for the initialization step can have an optional entry and exit action. Entry actions are indicated by the letter "E", exit actions by the letter "X". Initialization steps can only have an 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.

These actions can be added or deleted from the step's shortcut menu.



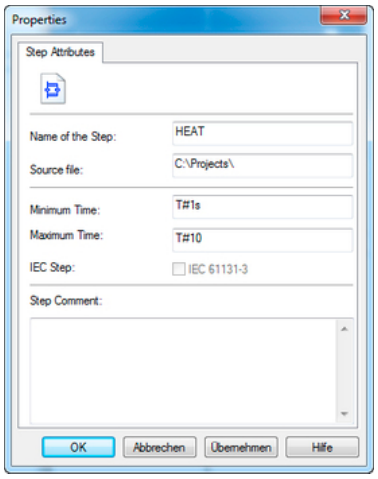
Step with entry and exit action

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ Defining SFC entry actions

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ Defining SFC exit actions

3.3 Monitoring the timing 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. System variables can be used to intercept violations of the configured time (see Using system variables).



Name of the Step: HEAT
Source file: C:\Projects\
Minimum Time: T#1s
Maximum Time: T#10
IEC Step: ☐ IEC 61131-3
Step Comment:

Properties dialog box for a step, minimum and maximum execution time

In addition to time monitoring, the Properties dialog box also has space to add a comment for the step.

This comment and any time values gives are displayed in the step's tooltip.

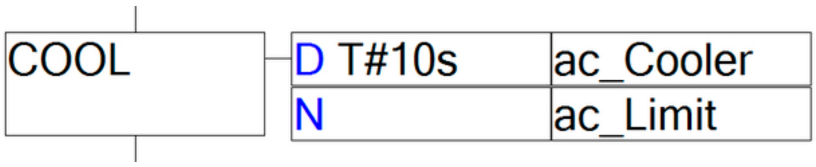
3.4 Action steps

Action steps can also be referred to as IEC steps. Action steps themselves do not contain any program code; they can only contain the actions associated with the action step. These actions are executed when the step is active.

There are two different types of action blocks:

- Boolean actions
- Actions with program code

In addition, qualifiers can be defined to determine how the actions should be called. For example, it is possible to specify that an action should be delayed by a certain amount of time when the step is activated.



Step with delayed action and a cyclic action

The content of an IEC action with program code is programmed in a separate program module, e.g. in Structured Text. IEC actions can be used multiple times, therefore making it possible to associate them with several steps with different qualifiers.

With Boolean actions, a variable of data type BOOL is connected in addition to the qualifier. The value of the variable is set to TRUE whenever the action is executed.

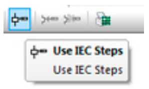
For a complete list of possible qualifiers, please refer to the help documentation.

Overview of important qualifiers:

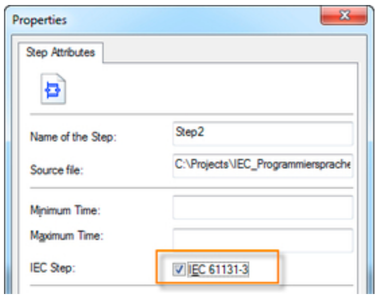
- Calls an action cyclically (N)
- Limits an action temporally (L)
- Delays an action (D)
- Sets an action conditionally (S)
- Resets an action (R)

Activating IEC steps

A setting in the toolbar makes it possible to configure whether a step with cyclic program code or an IEC step with IEC actions should be inserted. Steps that have already been placed can be reconfigured in the step's Properties dialog box.



Default setting "Use IEC steps"


Properties

Step Attributes

Name of the Step: Step2

Source file: C:\Projects\IEC_Programmiersprache

Minimum Time:

Maximum Time:

IEC Step: ☑ IEC 61131-3

Step property "IEC 61131-3"

Up to ten different actions with any variety of qualifiers can be associated with a single action step.

In SFC, action blocks are executed according to the logic of the "final scan".

For this reason, actions with the qualifier "P" are always called twice.

The scan in which the action is currently being processed can be seen with the action's instance structure (3.4.2 "Evaluating an action's 'final scan'").

In addition, deactivated actions are called at the end of the program in alphabetical order; newly active actions are then called, also in alphabetical order. This should be taken into consideration when constructing the sequential program.

Programming \ Programs \ Sequential Function Chart (SFC) \ Action step

Programming \ Actions

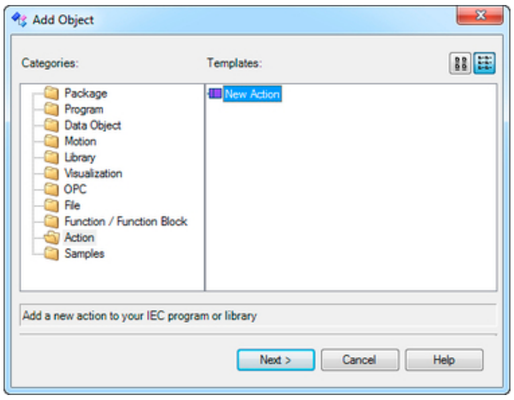
Programming \ Actions \ Action block - Qualifiers

Programming \ Actions \ Using actions in the various programming languages

3.4.1 Creating and associating actions

Creating an action with program code

An action containing program code must first be created before it can be associated with a step. A new action is added using the wizard in the logical view.



Add a new action to your IEC program or library

Inserting a new action in the Logical View

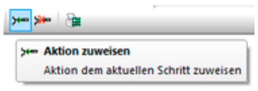
Project Management \ Logical view \ Wizards in the logical view

Creating a Boolean action

The only thing needed for a Boolean action is a local or global variable of data type BOOL. This must be declared in the variable declaration window before the action can be associated with a step.

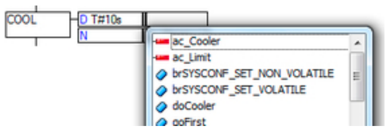
Associating an action with a step

Actions can be associated with a step using the "Associate action" toolbar icon. The step must be selected first.



"Associate action" icon

Qualifiers (N, D, P, etc.) as well as the actual action can then be assigned in the action associated with the step. A list of possible actions (with code or Boolean) can be opened up with the key combination <**CTRL**> + <**space bar**>.



Displaying all actions and Boolean variables with <CTRL> + <space bar>

It is possible to use an action with different qualifiers for different steps. However, this is not always a good idea.

For example, if an action is set with the "S" qualifier, then one can assume the qualifier "R", which affects the same action.

But if this action is also called with the qualifier "N", then the action will already be reset when the step becomes inactive. The "R" qualifier therefore no longer has any effect on the action since it has already been reset.

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ Using action blocks

Programming \ Editors \ General operations \ SmartEdit

3.4.2 Evaluating an action's 'final scan'

Depending on the program constellation, it may be necessary to only call an action once. In these cases, the final scan can be queried in actions with program code using the action's instance structure.

An action with the name "ac_Once" and qualifier "P" should only be called once. Because of this, the final scan is queried in the action. The programming language used here is Structured Text (ST).

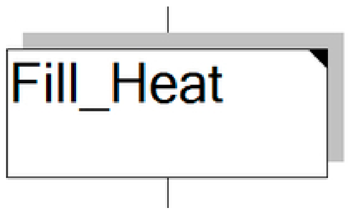
Program code

```
ACTION ac_Once:
    IF  ac_Once.x AND NOT ac_Once._x THEN
        (*last scan*)
    ELSE
        OnlyOnce :=  TRUE;
    END_IF ;
END_ACTION
```

3.5 Compound step

A compound step is an SFC step that can contain another SFC program as cyclic program code. These steps are indicated by a black triangle in their upper right corner and a gray shadow.

This makes it possible, for example, to integrate smaller sequences into a larger, more complex sequence. The underlying SFC program includes an initialization step as well as all of the other possibilities available in the main program.



Compound step

Programming \ Programs \ Sequential Function Chart (SFC) \ Compound step

Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ Using several SFC layers

4 TRANSITIONS

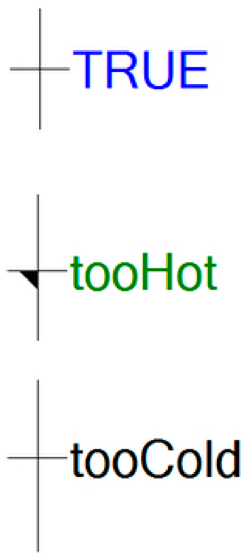
Each step has a transition that can be used to render it inactive. If a transition is active, then the associated step becomes inactive and the following step is activated in the next cycle.

A transition can have one of two values: "TRUE" or "FALSE". If the value is "TRUE", then the transition is active.

The value of a transition can be received from a constant, a variable or a bit of program code that returns a Boolean value.

Transitions that are based on underlying program code are indicated by a black triangle. The text of a transition is shown in green and also serves as a comment.

Two transitions may never be placed directly next to one another.



Transition with constant, program code and variable

Moving to a different step by switching a transition doesn't take place in the same program cycle. The newly active step always becomes active in the next cycle.

If several transitions are active in a chain of steps, then the SFC program looks for a stable state. The step with a subsequent inactive transition is executed cyclically. The entry and exit actions are called for the steps being scanned.

Programming \ Programs \ Sequential Function Chart (SFC) \ Transition / Transition condition

4.1 Transitions with program code

Transitions can also contain program code. The only condition is that the program code returns a result when called once. This return value must be of data type BOOL; it represents the switching state of the transition.

It is possible to combine logical operators (AND, OR, XOR, etc.). It's also possible to call functions.

The following programming languages can be used for transitions with program code:

- Instruction List
- Ladder Diagram
- Structured Text
- Function chart
- Continuous Function Chart
- B&R Automation Basic

Program code for a transition can be inserted either by simply clicking on the transition or from its shortcut menu. A dialog box opens up to select the programming language.

WARNING!

Only functions may be used in transitions. Whatever expression is used in the transition must return a clearly defined result in the first call. The compiler returns an error message if attempting to call a function block:

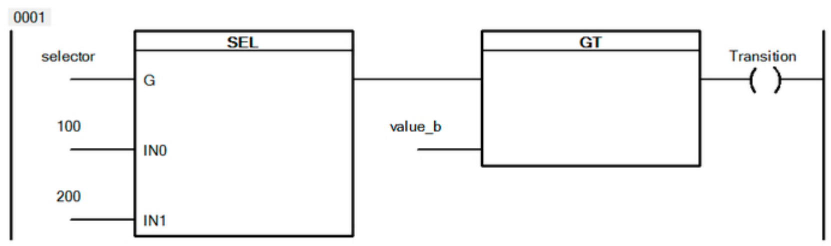
```
Error 1453: A transition is not allowed to cause side effects (assignments, function block calls, etc.).
```

4.1.1 Transitions in Ladder Diagram

A transition can be programmed as a network in Ladder Diagram. The transition is given at the end of the network. When a transition is created in Ladder Diagram, a network is created automatically with the transition placed in the output position.

The "Transition" identifier is to be used and is pre-declared by the system.

A transition in Ladder Diagram will look something like this:



Transition with two functions. The "selector" variable determines the preselected value for the subsequent comparison.

Programming \ Programs \ Ladder Diagram (LD)

4.1.2 Transitions in Structured Text

In Structured Text, transitions are defined as an expression without assignment operators. Using logical operators, comparison operators and functions is permissible.

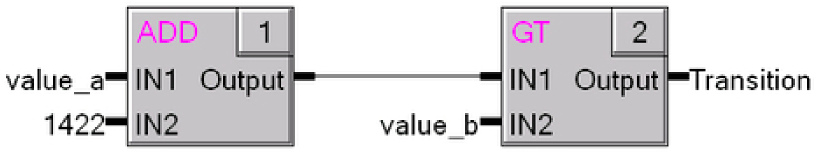
The following statement is TRUE if the values being compared are unequal and the maximum of both values does not exceed 100.

```
(value_a <> value_b) AND (MAX(value_a,value_b)< 100)
```

Programming \ Programs \ Structured Text (ST)

4.1.3 Transitions in Function Block Diagram and CFC

In Function Block Diagram (FBD) and Continuous Function Chart (CFC), transitions can be placed in a function chart network. The result is then assigned to the "Transition" variable. This variable already exists in the system and doesn't have to be declared.



Addition with "greater than" comparison results in the transition

Programming \ Programs \ Function Block Diagram (FBD)

4.1.4 Transitions in Instruction List

Transitions can be executed in the Instruction List (IL) language without an assignment operator. When called, the expression must return a Boolean result.

The transition below is set to TRUE if "value_a" is greater than 200.

```
(* Transition is TRUE if *)
(* value_a > 200         *)
LD      value_a
GT      200
```

Programming \ Programs \ Instruction List (IL)

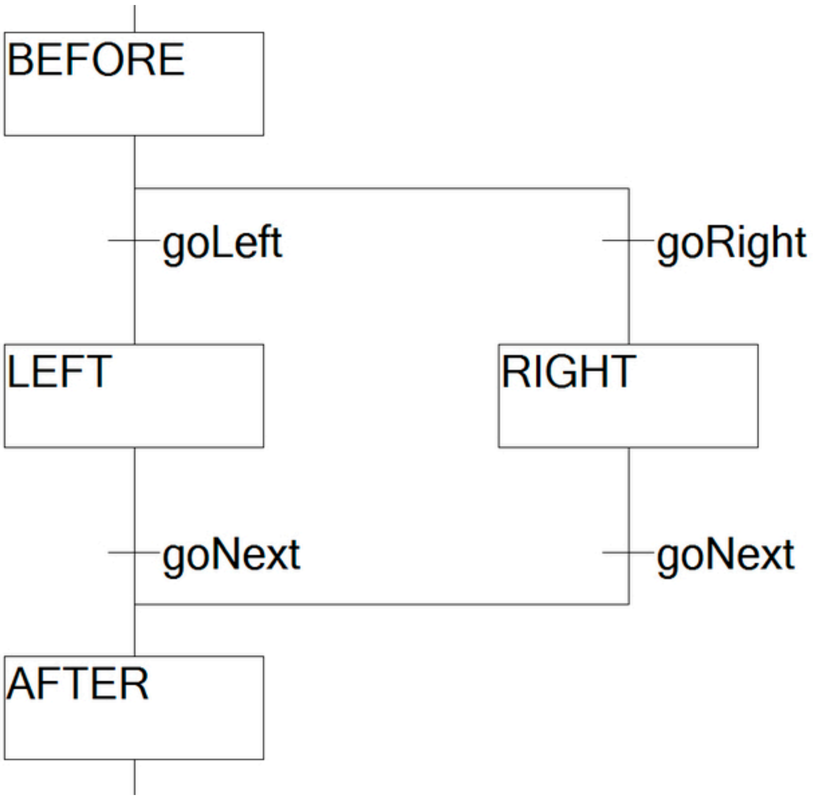
5 ALTERNATIVE AND PARALLEL BRANCHES

In SFC, individual steps don't always have to be executed one right after the other. To provide more flexibility, parallel and alternative branches are permitted.

5.1 Alternative branches

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. Because of this, the path divides before the alternatives themselves. Transitions are evaluated from left to right in this process.

The alternative branch can either return to the "main" path or end in a jump.

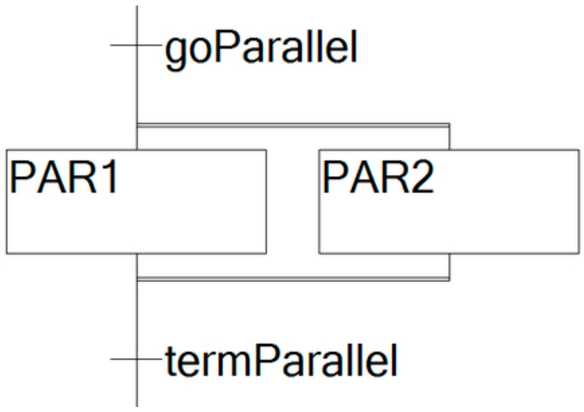


Alternative branch fed back via the "goNext" transition

Programming \ Programs \ Sequential Function Chart (SFC) \ Alternative branch

5.2 Parallel branches

Dividing programs up into several steps considerably increases their clarity. If steps have to be executed simultaneously, then a parallel branch can be created in SFC. The parallel steps are activated and deactivated with a common transition.



Two parallel steps

Parallel steps are executed from left to right.

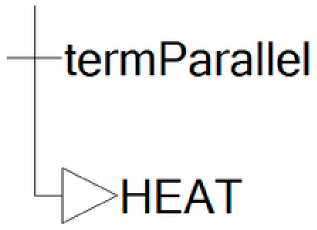
Programming \ Programs \ Sequential Function Chart (SFC) \ Parallel branch

6 JUMPS

Jumps are a way to make a sequence of steps a bit more flexible. A jump is activated by a transition. When active, a jump can transfer control to any of the other steps in the program.

An SFC program sequence always ends with a jump, which usually leads back to the sequence of steps. Alternative and parallel branches can also end in a jump.

Jumps to transitions are not allowed.



Jump to the "HEAT" step

Programming \ Programs \ Sequential Function Chart (SFC) \ Jump

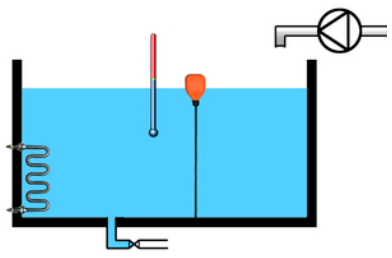
Programming \ Editors \ Graphic editors \ Sequential Function Chart editor \ Inserting SFC elements

7 EXAMPLES

7.1 Application example: Fill control

The fill level and temperature of the water in an aquarium need to be controlled. In this example, there are two processes that can be handled separately from one another.

A heating element is used to increase the temperature until it reaches the setpoint. Once the setpoint has been reached, the heating element is turned back off.



Schematics of the exercise

Control of the fill level is handled using a pump and a floating switch. Once the floating switch detects that the fill level is too low, a pump will fill the water in the container until the proper fill level is achieved. A buffer time will be added to reduce the switching frequency of the pump when waves occur.

Implementation in SFC

Since we are dealing with two independent sequences, we can develop a solution that uses parallel branches that are executed concurrently. 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

This exercise should be solved in an SFC program using a parallel branch. Each branch will include the states that were described in the state diagram.

1) Create an SFC program.
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 (GT / LT function blocks, Ladder Diagram programming language).
5) Program the actions and switching commands.

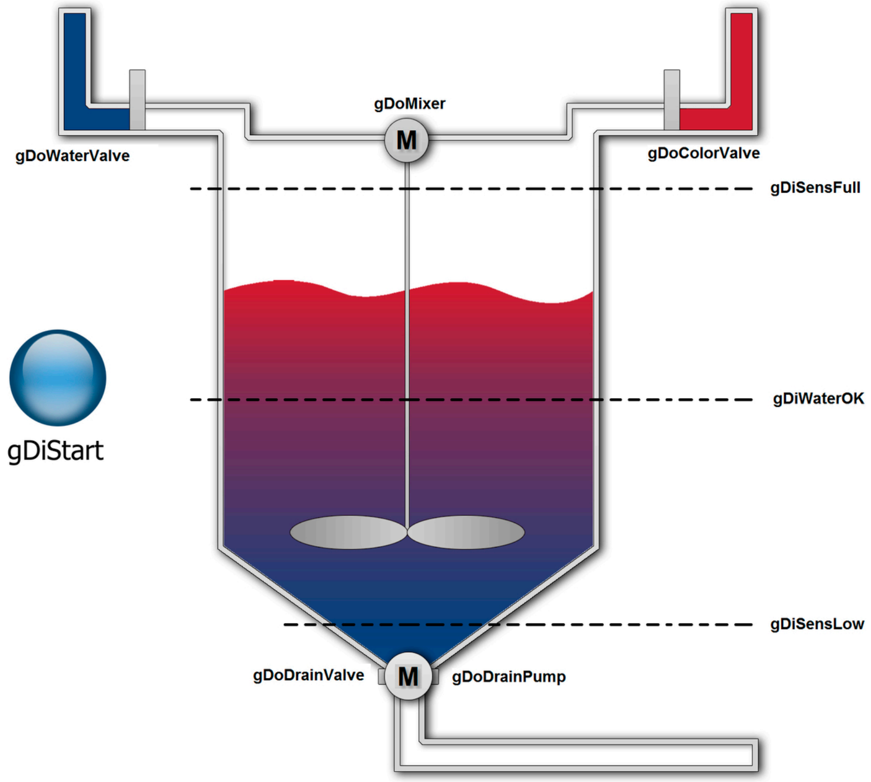
   The switch-on and switch-off commands for the heating element and the pump should be implemented using the steps' entry and exit actions.
6) Use an IEC action for the time delay.

   The transition that deactivates the waiting step should be switched with an action that has the "D" qualifier. The delay time should be set to 5 seconds.
7) Test the sequences by setting the transitions in the variable watch window.

One possible solution can be found in the appendix (see Exercise solution: Fill control).

7.2 Application example: Mixer control

Here we are going to configure a mixing system that mixes water and paint into a dispersion.



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 is added to the container until the **"gDiWaterOK"** sensor reacts.
- The stirring unit **(gDoMixer)** starts and paint is added until the **"gDiSensFull""** sensor reacts.
- 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 reacts.
- The starting situation is restored.

Implementing the mixer control

The mixing procedure just described should now be programmed. To do so, carry out the following steps:

1) Sketch out the necessary SFC.
   - Define the steps.
   - Define the transitions.
   - Define the actions.
2) Determine which steps need an IEC action.

3) Covert these requirements into an SFC program.

The stirring unit will be active over several steps.

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

- Turning the stirring unit on and off in the steps' entry/exit action.
- Stirring unit is a parallel step to the other steps.
- The stirring unit is turned on with an action with the "S" qualifier and turned back off at the end of the sequence with an action with the "R" qualifier.

One possible solution can be found here: Exercise solution: Mixer control.

8 USING SYSTEM VARIABLES

SFC system variables are variables that the system already "knows" and that have already been given names. These variables can be declared and used in the program. System variables contain information about the current state of the SFC program and can influence how the program is executed.

For example, they can be used to determine which step the program is currently executing as well as to stop or reset the program. Steps that violate their time limits are made known.

A complete list of system variables can be found in the Automation Studio online help documentation.

Procedure for using system variables:

- Select the necessary variable from the help system.
- Declare the system variable with the given data type.
- Use the system variable in the program.
- Display the variable in the variable watch window.

If system variables are not used in the program code, then the compiler will not store them in memory. It is then not possible for them to be shown in the variable watch window.

Programming \ Programs \ Sequential Function Chart (SFC) \ Sequential Function Chart - System variables

8.1 Usage in applications

The following will describe a few cases where SFC system variables can be used in applications. Some of these examples will also improve your understanding of using the different diagnostic tools for SFC programs.

In order to use system variables, they must first be declared and added somewhere in the SFC program's code. If the system variables should merely be tracked or modified in the variable watch window for testing purposes, then they can be integrated in the initialization step (Structured Text) with the following statement:

```
SFCInit;
```

Resetting the SFC program

The two system variables "SFCInit" and "SFCReset" can be used to reset the SFC program. If the "SFCInit" variable is located in the program, then "SFCReset" is not evaluated. In both cases, the program is reset and the initialization step called.

With "SFCReset", the initialization step is called cyclically as long as the variable remains TRUE.

With "SFCInit", the initialization step is called once when there is a falling edge of the variable.

When setting the variables "SFCInit" and "SFCReset", the exit actions of active steps are not called. Setting these variables in essence aborts the program.

Task: Reset the SFC program

The following exercises can be applied to the mixer control example program.

1) Declare the variable "SFCInit" with data type BOOL.
2) Use the variable in the program code.
3) Start the program sequence.
4) Set the system variable in the variable watch window.
5) Monitor the program's response.
6) Remove the "SFCInit" variable from the program.
7) Declare and use "SFCReset".
8) Compare the different program responses.

Determining the current program step

The program step currently being executed can be read using the "SFCCurrentStep" system variable. Its data type is STRING.

With parallel steps, the name of the branch furthest to the right is saved.

Task: Determine the current program step

Determine the step currently being executed in your program.

Using time monitoring

In SFC, steps can be configured with a minimum and maximum "buffer time". This buffer time begins as soon as the step becomes active. The time is then reset when the step has been deactivated. By default, time monitoring is disabled for the steps.

System variable	Data type	Function
SFCEnableLimit	BOOL	Enables time monitoring for steps
SFCError	BOOL	Indicates whether an error is present
SFCErrorPOU	STRING	Indicates the program where the error occurred
SFCCurrentStep	STRING	Indicates the step that caused the error
SFCQuitError	BOOL	Acknowledges errors, clears the string variables

Table: System variables for detecting timing violations

Task: Using time monitoring

Expand your program so that some of its steps have a maximum time of one minute. Evaluate the timing violations in the variable watch window.

1) Configure the maximum times for some of the steps.
2) Declare the necessary system variables.
3) Enable time monitoring.
4) Force a timing violation.
5) Evaluate the step that causes the error.

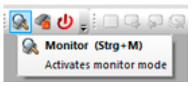
When monitor mode is enabled, a step's running time is shown in its tooltip.

9 DIAGNOSTIC FUNCTIONS

Diagnostic tools are necessary to ensure that programming languages are used efficiently. This section will provide a brief list of the troubleshooting tools that can be used for SFC programs.

9.1 Monitor mode in SFC

As a central component, monitor mode enables the use of many diagnostic tools. It can be enabled with the monitor mode icon in the toolbar.



Enabling monitor mode

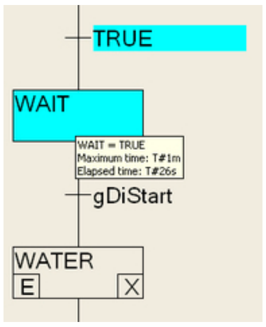
Monitor mode is used to enable the following diagnostic tools:

- Powerflow
- Variable watch
- Debugger (9.3 "Debugger")

Powerflow

Powerflow can be enabled whenever monitor mode is active. It shows all active steps, transitions and actions in color. This makes it easy to determine the step currently being executed in the SFC program as well as which transitions are being switched.

The image shown here illustrates Powerflow when it is active. Active steps and transitions are shown in color. The step's tooltip displays information such as the status of the step as well as the time that has elapsed if time monitoring is enabled.

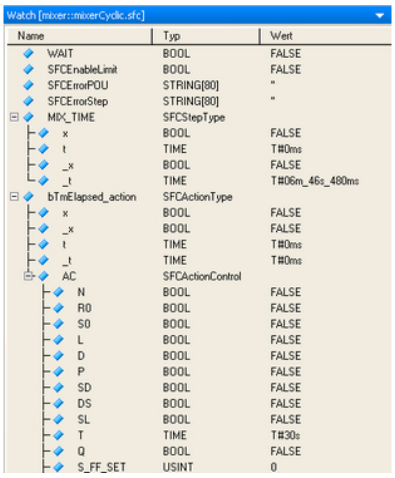


Powerflow in monitor mode with time monitoring enabled for the step

Diagnostics and service \ Diagnostic tool \ Monitors \ Sequential Function Chart editor in monitor mode

Variable watch

The variable watch feature makes it possible to observe not just the variables used in the program, but also the SFC system variables and instance structures of steps and actions. Variables can also be set/modified here.


Watch [mixer::mixerCyclic.sfc]

Name	Typ	Wert
WAIT	BOOL	FALSE
SFCEnableLimit	BOOL	FALSE
SFCErrorPOU	STRING[80]	''
SFCErrorStep	STRING[80]	''
MIX_TIME	SFCStepType	
x	BOOL	FALSE
t	TIME	T#0ms
_x	BOOL	FALSE
_t	TIME	T#06m_46s_480ms
bTmElapsed_action	SFCActionType	
x	BOOL	FALSE
_x	BOOL	FALSE
t	TIME	T#0ms
_t	TIME	T#0ms
AC	SFCActionControl	
N	BOOL	FALSE
R0	BOOL	FALSE
S0	BOOL	FALSE
L	BOOL	FALSE
D	BOOL	FALSE
P	BOOL	FALSE
SD	BOOL	FALSE
DS	BOOL	FALSE
SL	BOOL	FALSE
T	TIME	T#30s
Q	BOOL	FALSE
S_FF_SET	USINT	0

Displaying variables in the variable watch window

Diagnostics and service / Diagnostic tools / Variable watch

9.2 SFC variables

System variables

System variables (Using system variables) can be used to determine the status of a program. For example, the SFC program can be paused and then continued in "Step into" mode, or the SFC program can be reset for testing purposes.

The possibilities include the following:

- Determining the active step
- Determining whether transitions are active
- Enabling / disabling time monitoring
- Determining whether time monitoring has been triggered
- Pausing the SFC program
- Continuing the SFC program in "Step into" mode
- Resetting the SFC program

Programming \ Programs \ Sequential Function Chart (SFC) \ Sequential Function Chart - System variables

Instance variables of steps, actions and transitions

The compiler implicitly generates instance variables for each step and action. These instance variables can then be inserted into the variable watch window and observed.

The step variable, for example, can be used to determine whether the step is currently active or not. The name of the step variable is exactly the same as the step itself and is of data type BOOL or SFCStepType, depending on the type of step. If the step is active, then its step variable will be set to TRUE.

Instance variables for actions can be used e.g. to determine the final scan (3.4.2 "Evaluating an action's 'final scan'"). The data type is SFCActionType.

WARNING!

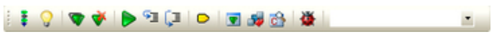
Instance variables for steps and Boolean actions are generated automatically by Automation Studio when the program is compiled. It is not recommended to manipulate these instances to affect the sequence. This renders programs unclear and prone to errors.

9.3 Debugger

As with textual programming languages, the debugger is also supported in SFC. It allows breakpoints to be assigned to steps. If the SFC program sequence scans through a step with a breakpoint, then the SFC program is paused.

Once the debugger is closed, the SFC continues cyclic execution.

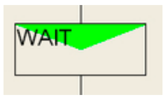
All actions can be controlled using the debugger toolbar and message window.


The debugger toolbar

Setting breakpoints

When monitor mode is enabled, it is possible to set several breakpoints for steps using the debugger toolbar. The debugger is not yet active during this process.

Breakpoints that have already been set are indicated by a green triangle in the step.

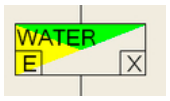


A set breakpoint

Enabling the debugger

The debugger can then be enabled by clicking on the respective icon in the debugger toolbar. When the program is running, it will pause when it reaches one of these breakpoints.

The step with the breakpoint where the program has been paused is indicated by a yellow triangle.



Breakpoint reached in a program

Diagnostics and service \ Diagnostics tools \ Debugger

Task: Test the debugger

Set breakpoints in your last SFC program and test debugger functions.

1) Open the program.
2) Start monitor mode.
3) Set the breakpoints.
4) Enable the debugger.
5) Observe the output in the message window.

10 SUMMARY

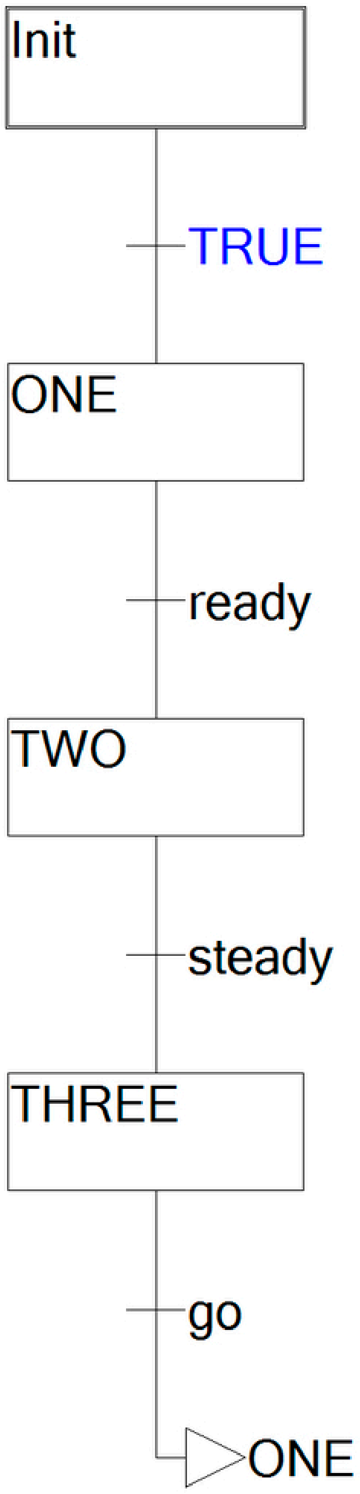
With its visual interface, the Sequential Function Chart programming language is very well structured and straightforward. It provides an excellent way to represent state diagram and state machine programs in the form of steps and transitions.

Because programs are executed sequentially and a range of diagnostic tools are available, maintaining and troubleshooting SFC programs is very easy.

Other processes of a control application can also be called from the SFC program over defined interfaces. This makes the sequence and the subfunctions called clear and easy to test.

When combined with other IEC programming languages like Ladder Diagram or Structured Text, SFC becomes a very powerful language.

Action blocks that use IEC actions with various qualifiers round out the range of functions available in SFC.

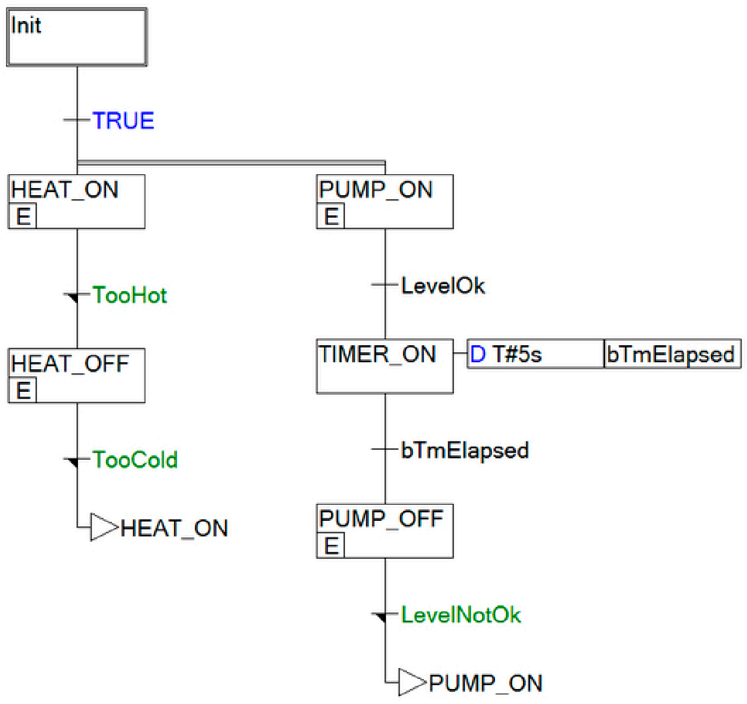


The SFC programming language

11 APPENDIX: EXERCISE SOLUTIONS

11.1 Exercise solution: Fill control

The exercise from Application example: Fill control can be solved like this:



Possible solution - Implementation with parallel branches

The main branch of the parallel branches corresponds to the two state diagrams.

The transitions between "HEAT_ON" and "HEAT_OFF" contain program code that compares the set temperature with the actual temperature.

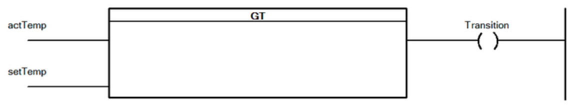
The switching commands for turning the heating unit on and off are implemented in the entry actions of the heating steps.

The pump branch uses an IEC action with the "D" qualifier for the delay time and only switches the transition for the next step in its action.

The entry action of "PUMP_OFF" resets the transition of the elapsed time as well as the output for the pump.

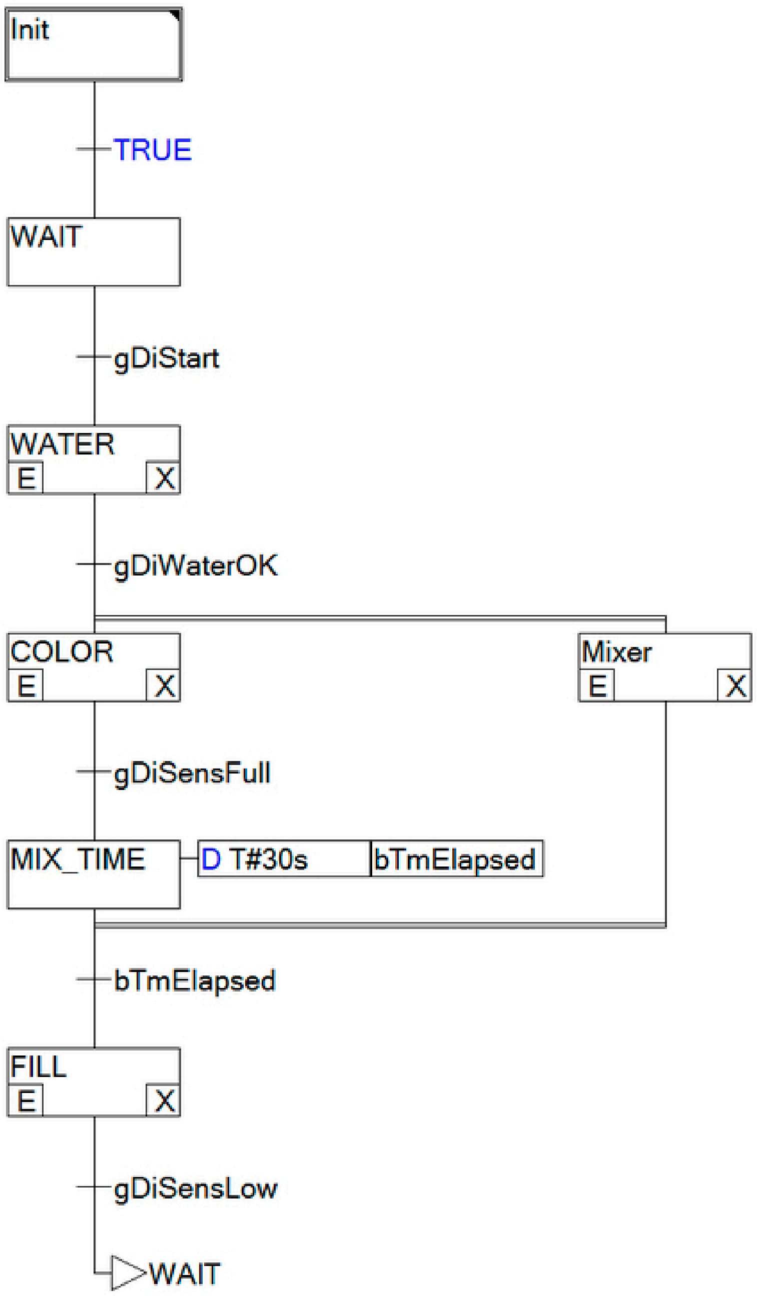
Contents of the transitions

The "TooHot" transition could include something like a Ladder Diagram that compares the setpoint with the actual temperature value. If the actual temperature is greater than the set temperature, then the transition is switched.



Transition "TooHot" in Ladder Diagram

11.2 Exercise solution: Mixer control



Possible solution for the mixer control system

The mixer control system sequence (Application example: Mixer control) can be solved as shown in the image provided here.

One step for controlling the mixer can be used as a parallel branch for filling the paint and handling the mixing time.

The entry and exit actions can be used to switch the actuators such as the valves and motors, respectively.

To handle time monitoring, the Boolean action "bTmElapsed" is called with the qualifier "D T#30s" and used as the transition to the next step.

TRAINING MODULES

TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram
TM241 – Function Block Diagram (FBD)
TM242 – Sequential Function Chart (SFC)
TM246 – Structured Text
TM250 – Memory Management and Data Storage
TM261 – Closed Loop Control with LOOPCONR
TM400 – The Basics of Drive Technology
TM410 – ASiM Basis
TM440 – ASiM Basic Functions
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors
TM500 – Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER
TM540 – ASiST SafeMC
TM600 – The Basics of Visualization
TM630 – Visualization Programming Guide
TM640 – ASiV Alarms, Trends and Diagnostics
TM670 – Visual Components Advanced
TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM890 – The Basics of LINUX

TM242TRE.00-ENG / V1.0.1
©2011 by B&R, All rights reserved.
All registered trademarks are the property of their respective companies.
Technical changes reserved.

www.br-automation.com