# Modicon Libraries – HVAC Library

## User Manual

EIO0000000359.01
05/2021


www.se.com

# Legal Information

# Table of Contents

# Safety Information

## Important Information

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

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

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

**⚠ DANGER**

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

**⚠ WARNING**

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

**⚠ CAUTION**

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

***NOTICE***

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

# Please Note

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

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

# Before You Begin

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

## ⚠WARNING

**UNGUARDED EQUIPMENT**

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

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

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

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

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and

serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

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

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

## Start-up and Test

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

**⚠WARNING**

**EQUIPMENT OPERATION HAZARD**

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

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

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

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

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

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.

- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

## Operation and Adjustments

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

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

# About the Book

## Document Scope

This document describes the function blocks of the Modicon Libraries – HVAC (Heating, Ventilation and Air Condition) Library.

Users are supposed to have a good working knowledge in employment of Control Expert.

## Validity Note

Modicon Libraries – HVAC Library 2021 supports Control Expert V15.0 or later.

This library is compatible with the following hardware platforms:

- Quantum
- M340
- M580

## Related Documents

| Title of Documentation | Reference Number |
|---|---|
| EcoStruxure™ Control Expert Program Languages and Structure Reference Manual | T002683578 |

You can download these technical publications and other technical information from our website at www.se.com/ww/en/download/ .

# Product Related Information

## ⚠WARNING

**LOSS OF CONTROL**

Review potential failure modes of the control paths for critical control functions.

Provide a means to achieve a safe state during and after a path failure.

Provide separate or redundant control paths for critical control functions.

Review the implications of transmission delays or failure of communication links.

Apply local accident prevention and safety regulations and guidelines. [1]

Test each implementation of this library for proper operation before placing it into service.

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

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

The application of this product requires expertise in the design and operation of control systems.

## ⚠WARNING

**UNINTENDED EQUIPMENT OPERATION**

Allow only authorized personnel with such expertise to program, install, alter, and apply this product.

Follow local and national safety codes and standards.

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

Examples in this manual are given for information only.

## ⚠WARNING

**UNINTENDED EQUIPMENT OPERATION**

Adapt examples that are given in this manual to the specific functions and safety requirements ofyour industrial application when you implement them.

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

# General Information

## What's in This Part



## Introduction

This part contains general information on the Modicon Libraries – HVAC Library.

# Block Types and their Applications

## What's in This Chapter



# Overview

This chapter describes the different block types and their applications.

# Block Types

## Block Types

Different block types are used in Control Pro. The general term for the block types is Function and Function Block (FFB).

There are the following types of blocks:

- Elementary Function (EF)
- Elementary Function Block (EFB)
- Derived Function Block (DFB)
- Procedure

**NOTE:** Motion Function Blocks are not available on the Quantum platform.

## Elementary Function

Elementary functions (EF) have no internal status and one output only. If the input values are the same, the output value is the same for the executions of the function, for example the addition of two values gives the same result at every execution.

An elementary function is represented in the graphical languages (FBD and LD) as a block frame with inputs and an output. The inputs are represented on the left and the outputs on the right of the block frame. The name of the function, that is the function type, is shown in the center of the block frame.

The number of inputs can be increased with some elementary functions.

### Elementary Function Block

Elementary function blocks (EFB) have an internal status. If the inputs have the same values, the value on the outputs can have another value during the individual executions. For example, with a counter, the value on the output is incremented.

An elementary function block is represented in the graphical languages (FBD and LD) as a block frame with inputs and outputs. The inputs are represented on the left and the outputs on the right of the block frame. The name of the function block, that is the function block type, is shown in the center of the block frame. The instance name is displayed above the block frame.

### Derived Function Block

Derived function blocks (DFBs) have the same properties as elementary function blocks. They are created by the user in the programming languages FBD, LD, IL and/or ST.

### Procedure

Procedures are functions with several outputs. They have no internal state.

The only difference from elementary functions is that procedures can have more than one output and they support variables of the `VAR_IN_OUT` data type.

Procedures do not return a value.

Procedures are a supplement to IEC 61131-3 and needs to be enabled explicitly.

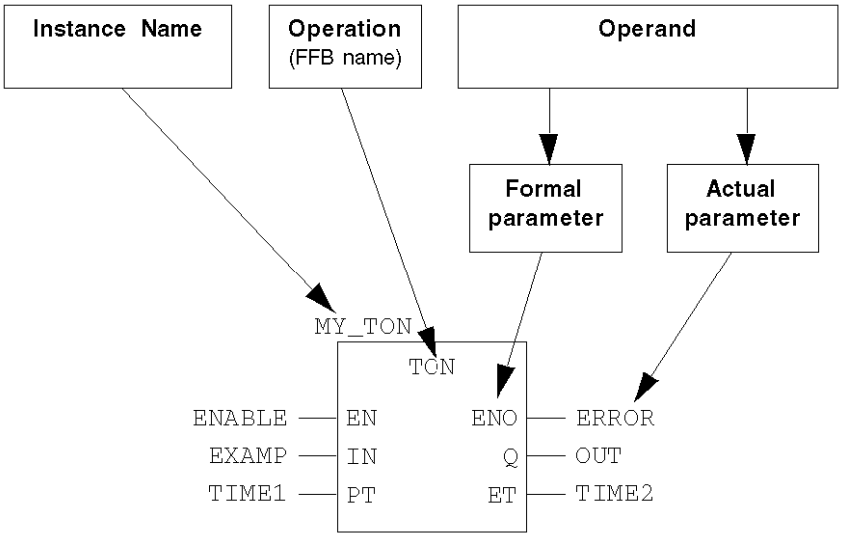
There is no visual difference between procedures and elementary functions.

## FFB Structure

### Structure

Each FFB is made up of an operation (name of the FFB), the operands are required for the operation (formal and actual parameters) and an instance name for elementary/derived function blocks.

Call of a function block in the FBD programming language:



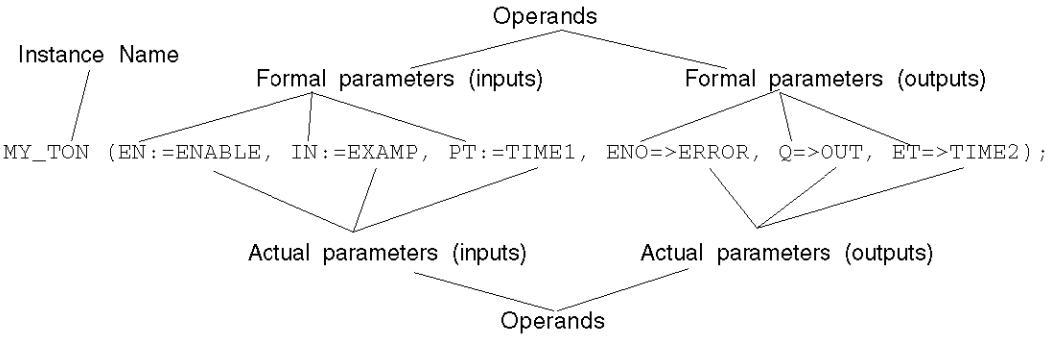

## ⚠CAUTION

**UNEXPECTED APPLICATION BEHAVIOR**

Do not call the same block instance more than once within a controller cycle.

**Failure to follow these instructions can result in injury or equipment damage.**

Formal call of a function block in the ST programming language:



## Operation

The operation determines which function is to be executed with the FFB, e.g. shift register, conversion operations.

## Operand

The operand specifies what the operation is to be executed with. With FFBs, this consists of formal and actual parameters.

## Formal/actual parameters

Inputs and outputs are required to transfer values to or from an FFB. These are called formal parameters.

Objects are linked to formal parameters; these objects contain the current process states. They are called actual parameters.

At program runtime, the values from the process are transferred to the FFB via the actual parameters and then output again after processing.

The data type of the actual parameters has to match the data type of the input/output (formal parameters). The only exceptions are generic inputs/outputs whose data type is determined by the actual parameter. If the actual parameters consist of literals, a suitable data type is selected for the function block.

## FFB Call in IL/ST

In text languages IL and ST, FFBs can be called in formal and in informal form. Details can be found in the *Reference manual*.

Example of a formal function call:

```
out:=LIMIT (MN:=0, IN:=var1, MX:=5);
```

Example of an informal function call:

```
out:=LIMIT (0, var1, 5);
```

**NOTE:** The use of `EN` and `ENO` is only possible for formal calls.

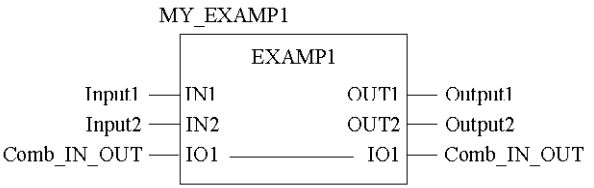
## `VAR_IN_OUT` variable

FFBs are often used to read a variable at an input (input variables), to process it and to output the altered values of the **same** variable (output variables).

This special type of input/output variable is also called a `VAR_IN_OUT` variable.

The input and output variable are linked in the graphic languages (FBD and LD) using a line showing that they belong together.

Function block with VAR_IN_OUT variable in FBD:




Function block with VAR_IN_OUT variable in ST:

```
MY_EXAMP1 (IN1:=Input1, IN2:=Input2, IO1:=Comb_IN_OUT,
           OUT1=>Output1, OUT2=>Output2);
```

Consider the following points when using FFBs with VAR_IN_OUT variables:

- Assign a variable to the VAR_IN_OUT inputs.
- Literals or constants cannot be assigned to VAR_IN_OUT inputs/outputs.

The following additional limitations apply to the graphic languages (FBD and LD):

- When using graphic connections, VAR_IN_OUT outputs can only be connected with VAR_IN_OUT inputs.
- Only one graphical link can be connected to a VAR_IN_OUT input/output.
- Different variables/variable components can be connected to the VAR_IN_OUT input and the VAR_IN_OUT output. In this case the value of the variables/variable component on the input is copied to the output variables/variable component.
- No negations can be used on VAR_IN_OUT inputs/outputs.
- A combination of variable/address and graphic connections is not possible for VAR_IN_OUT outputs.

# EN and ENO

## Description

An EN input and an ENO output can be configured for the FFBs.

If the value of EN is equal to "0" when the FFB is invoked, the algorithms defined by the FFB are not executed and ENO is set to "0".

If the value of EN is equal to "1" when the FFB is invoked, the algorithms defined by the FFB will be executed. After the algorithms have been executed successfully, the value of ENO is

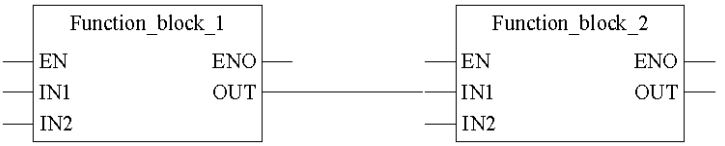
set to "1". In case of a detected error during the execution of these algorithms, ENO is set to "0".

If the EN pin is not assigned a value, when the FFB is invoked, the algorithm defined by the FFB is executed (same as if EN equals to "1"), Please refer to *Maintain output links on disabled EF*.

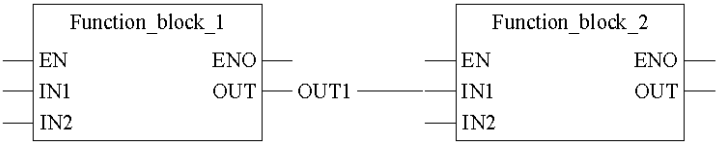
If the algorithms are executed successfully, then value of ENO is set to "1", else ENO is set to "0".

If ENO is set to "0" (caused by EN=0 or a detected error condition during execution or unsuccessful algorithm execution):

- Function blocks
  - EN/ENO handling with function blocks that (only) have one link as an output parameter:



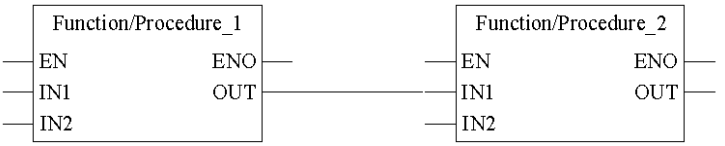

    If EN from FunctionBlock_1 is set to "0", the output connection OUT from FunctionBlock_1 retains the status it had in the last correctly executed cycle.
  - EN/ENO handling with function blocks that have one variable and one link as output parameters:




    If EN from FunctionBlock_1 is set to "0", the output connection OUT from FunctionBlock_1 retains the status it had in the last correctly executed cycle. The variable OUT1 on the same pin, either retains its previous status or can be changed externally without influencing the connection. The variable and the link are saved independently of each other.
- Functions/Procedures

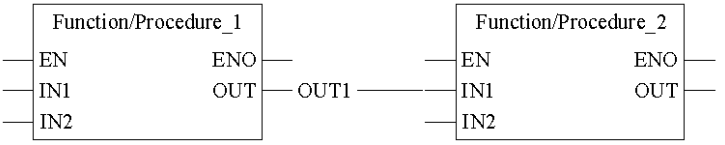
  As defined in IEC61131-3, the outputs from deactivated functions (EN-input set to "0") is undefined. (The same applies to procedures.)

  Here is an explanation of the output status in this case:
  - EN/ENO handling with functions/procedures that (only) have one link as an output parameter:




If EN from Function/Procedure_1 is set to "0", the output connection OUT from Function/Procedure_1 retains the status it had in the previous correctly executed cycle.

- EN/ENO handling with function blocks that have one variable and one link as output parameters:




If EN from Function/Procedure_1 is set to "0", the output connection OUT from Function/Procedure_1 retains the status it had in the last correctly executed cycle. The variable OUT1 on the same pin, either retains its previous status or can be changed externally without influencing the connection. The variable and the link are saved independently of each other.

The output behavior of the FFBs does not depend on whether the FFBs are called up without EN/ENO or with EN=1.

## Conditional/Unconditional FFB Call

"Unconditional" or "conditional" calls are possible with each FFB. The condition is realized by pre-linking the input EN.

- EN connected

  conditional calls (the FFB is only processed if EN = 1)

- EN shown, hidden, and marked TRUE, or shown and not occupied

  unconditional calls (The FFB is processed independently of the value EN)

  **NOTE:** For disabled function blocks (EN = 0) with an internal time function (e.g. DELAY), time seems to keep running, since it is calculated with the help of a system clock and is therefore independent of the program cycle and the release of the block.

**⚠CAUTION**

**UNEXPECTED APPLICATION EQUIPMENT**

Do not disable function blocks with internal time function during their operation.

**Failure to follow these instructions can result in injury or equipment damage.**

## IL and ST

The use of EN and ENO is only possible in the text languages for a formal FFB call, e.g.

```
MY_BLOCK (EN:=enable, IN1:=var1, IN2:=var2,
ENO=>error, OUT1=>result1, OUT2=>result2);
```

Assigning the variables to ENO needs to be done with the operator =>.

With an informal call, EN and ENO cannot be used.

# Availability of the Blocks on Different Hardware Platforms

## What's in This Chapter



# Introduction

You can use the Modicon Libraries –HVAC (Heating, Ventilation and Air Condition) library with Control Expert only.

This chapter contains information on the availability of the function blocks on different hardware platforms.

# Control Expert: Availability of Blocks on Various Hardware Platforms

## Introduction

You can use the Modicon Libraries – HVAC (Heating, Ventilation and Air Condition) library with Control Expert only.

## Related Documents

For further details on how to use EFBs with Control Expert (for example `EN` and `ENO`), please refer to *Control Expert Program Languages and Structure Reference Manual*.

## Block Availability

Block availability by hardware platform can be found in the following table:

| Block Name | Block Type | M340 | Quantum | M580 |
|---|---|---|---|---|
| `CC2_VAC` | EFB | + | + | + |
| `CC3_VAC` | EFB | + | + | + |
| `MC_VAC` | EFB | + | + | + |

| Block Name | Block Type | M340 | Quantum | M580 |
|---|---|---|---|---|
| PI_VAC | EFB | + | + | + |
| SEQ_VAC | EFB | + | + | + |
| SW_VAC | EFB | + | + | + |
| THRS_VAC | EFB | + | + | + |
| UC2_VAC | EFB | + | + | + |
| UC3_VAC | EFB | + | + | + |
| VQ_VAC | EFB | + | + | + |
| WASH_VAC | EFB | + | + | + |

# General Principles

## What's in This Chapter



# Introduction

This chapter provides information about the general principles of the Modicon Libraries – HVAC Library.

# Intended Use

The Modicon Libraries – HVAC Library puts at your disposal an extensive range of function blocks for the implementation of air conditioning systems using the Control Expert programming software.

# Function Blocks

## Function Block Groups

11 function blocks are available, split up as follows:

- 6 basic function blocks (Basic Group, see table below)
- 5 complex function blocks (Complex Group, see table below)

**NOTE:** It is strongly recommended that you familiarizes yourself with the operation of the basic function blocks before reading the complex function block sections.

## Basic Function Blocks

The basic function blocks implement low level functions for basic air conditioning solutions.

| Function Block | Description |
|---|---|
| PI_VAC | PI controller for air conditioning |
| SEQ_VAC | scaling/sequence block for air conditioning |
| SW_VAC | summer/winter setpoint compensation for air conditioning |
| THRS_VAC | threshold switch with hysteresis for air conditioning |
| VQ_VAC | measured value deadband block |
| WASH_VAC | basic washer block for air conditioning |

## Complex Function Blocks

The complex function blocks implement the more complex functions found in controllers which are frequently used in air conditioning systems. They are built using the basic function blocks.

| Function Block | Description |
|---|---|
| CC2_VAC | cascade controller for air conditioning with 2 outputs |
| CC3_VAC | cascade controller for air conditioning with 3 outputs |
| MC_VAC | air mix controller for air conditioning with 1 output |
| UC2_VAC | PI controller for air conditioning with 2 outputs |
| UC3_VAC | PI controller for air conditioning with 3 outputs |

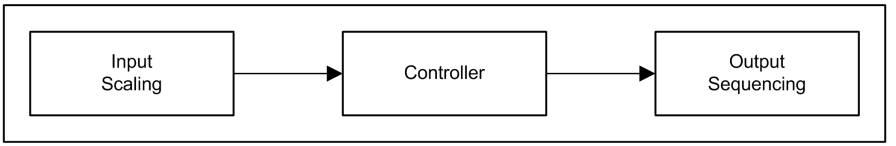
# Scaling and Descaling Within a Control Program

## General Instructions

Basically, a control set up consists of 3 parts:

- the scaling of input variables
- the controller
- the sequencing of output variables

Control set up




”Output Sequencing” is used to take the controller single output, split it into several parts, and to scale the resultant outputs. It can be used for example, to split the controller output amongst multiple actuators.

For the purposes of the Modicon Libraries – HVAC Library 2021, the controller and output sequencing are combined into 1 EFB.

The scaling of inputs can be handled using standard libraries or the Modicon Libraries – HVAC Library 2021 `SEQ_VAC` EFB can be used.

The complex EFBs include the controller plus 1 or more freely parameterizable output sequence blocks. The behavior of a complex EFB can be determined by examining the basic EFBs from which it is built. The complex EFBs are described in this document by referring to their constituent basic EFBs.

The combination of the controller with the output sequencing has been done to simplify the parameterization of the EFB as the output sequence parameters can be assigned at the same time as the controller parameters. Furthermore, the links from the controller output to the sequenced outputs are automatically created with a fixed structure.

## Single Loop Controls

In the case of simple, single loop controls, the purpose of the output sequencing is simply to match the controller output to the process actuators.

For controllers operating in simple P–mode, the controller output will equal zero when the setpoint is equal to the process variable. In this case, the output sequencing has to handle negative controller outputs and convert them into signals for the actuators.

By assigning the appropriate values to the output sequences, it is possible to design a P–controller with a 50% output when the process variable is equal to the setpoint.

The advantage of such an approach is that on startup, the controller output will drive a heating action which will reduce the chances of icing up in winter conditions.

However, where the possibility of icing up is high, this mechanism should not be depended upon for freeze protection as unfavorable dynamics could result in the heating control valve closing. In this case one should control the temperature manually by opening the heating control valve 100% on start up for a specific period of time before putting the controller into automatic and starting the fan.

| ⚠ CAUTION |
| --- |
| **UNINTENDED EQUIPMENT BEHAVIOR**<br><br>During winter conditions, control the temperature manually by opening the heating control valve 100% on start up for a specific period of time before putting the controller into automatic and starting the fan.<br><br>**Failure to follow these instructions can result in injury or equipment damage.** |

Where a project uses multiple controllers that have the same behavior, the user should define a consistent naming convention for the Control Expert project.

For example:

Y1 / Y2 / Y3 = Heat / Air Mix / Cool.

Where a controller output is divided into multiple process outputs using the sequence functions, one may have to take into account the fact that different process actuators have different gains. As a result, the controller gain may vary depending on the position of its output, i.e., whether the output is driving, a hot water control valve, an air mixing damper or humidity control device.

The different actuator gains can be compensated by using the appropriate sequence layout.

## Multi-Loop Controls

As well as single loops directly connected to process outputs, cascaded loops where the output of 1 controller is connected as the setpoint of another controller are often required for HVAC systems.

The use of cascaded controllers not only gives improved dynamics, it also results in a clearer and easier to understand layout of the HVAC controls.

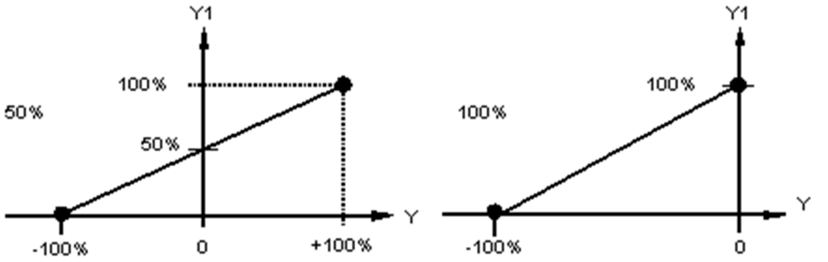
The output sequence functions allow the user to design a wide variety of control options for his HVAC strategy.

Examples of output sequencing are given in figure below.

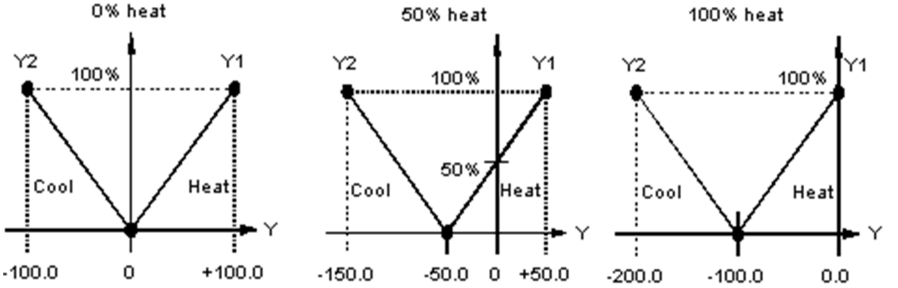
Y refers to the PI controller output, while Y1, Y2 and Y3 refer to the final outputs from the output sequencing which are either sent directly to the process actuators or to other programming controllers/functions.

**Examples of sequences with direct output of output values as manipulated variables:**

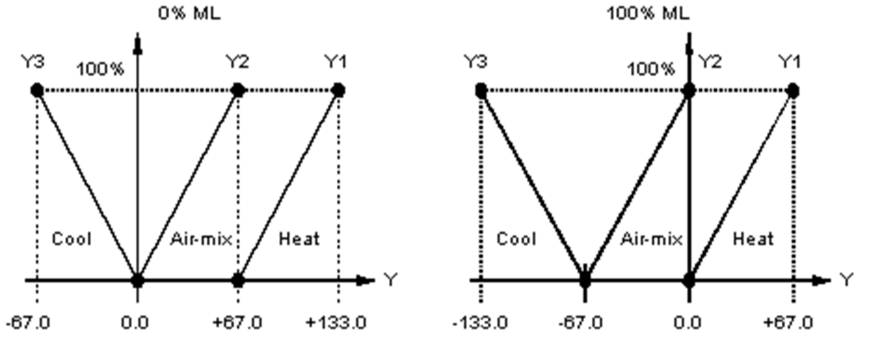
Single manipulated variable




Double manipulated variable




Triple manipulated variable



# EFB Descriptions

## What's in This Part



## Introduction

This part contains the EFB descriptions.

These modules do not reflect any specific installation.

# ⚠WARNING

**MISAPPLICATION OF FUNCTION BLOCKS**

Before adopting these function blocks for use in a specific application, you must:

- Conduct a safety analysis for the application and equipment installed.
- Verify that the selected function blocks are appropriate for the equipment or function in the installation.
- Supply appropriate parameters, particularly for limits.
- Check that all sensors and actuators are compatible with the selected function blocks.
- Verify that all functions of the selected function blocks work properly during verification and commissioning.
- Provide independent paths for critical control functions (emergency stop, over-limit conditions, etc.) according to the safety analysis and applicable codes, and regulations.

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

# CC2_VAC: Cascade Controller for Air Conditioning with 2 Outputs

## What's in This Chapter



# Introduction

This chapter describes the `CC2_VAC` function block.

# Brief Description

## Function Description

The `CC2_VAC` block is a cascade controller used to provide temperature or humidity control of the inlet air to a room. It consists of a P–only outer loop which uses the room temperature/humidity as process variable and an inner PI loop that controls the temperature/humidity of the inlet air supplying the room.

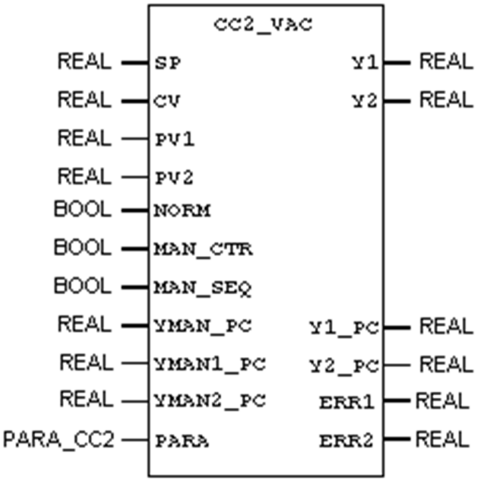
The EFB has a fixed structure where the setpoint of the P–controller is fed forward and added to the P–controller output to form the setpoint of the PI–controller. The output of the PI–controller has 2 output sequences, Y1 and Y2.

The EFB provides the following:

- winter/summer setpoint compensation as per DIN 1946 pert 2
- 4 quadrant operation of the output sequence scaling
- display of output variables as percentages
- upper and lower limits on outputs
- presetting the controllers' gains in the form of GAIN or PROP with the possibility of using negative values for switching the control direction
- operation with Anti–Windup–Reset (AWR)
- manual adjustment of either the PI–controller output or the individual sequence outputs (Y1 and Y2) using percentages
- When the controller output is manually adjusted, the EFB tracks the I contribution in order to provide bumpless switching back to automatic mode.

- display of the P and PI controller deviations (SP–PV)

## Representation




## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP | REAL | setpoint |
| CV | REAL | command variable |
| PV1 | REAL | actual value P controller |
| PV2 | REAL | actual value PI controller |
| NORM | BOOL | basic setting |
| MAN_CTR | BOOL | MANUAL for controller |
| MAN_SEQ | BOOL | MANUAL for sequences |
| YMAN_PC | REAL | total manual manipulated variable as a % for controller |
| YMAN1_PC | REAL | individual manual manipulated variable as a % for 1 sequence |
| YMAN2_PC | REAL | individual manual manipulated variable as a % for 2 sequence |
| PARA | PARA_CC2 | parameter structure |

### Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y1 | REAL | manipulated variable / output variable 1 |
| Y2 | REAL | manipulated variable / output variable 2 |
| Y1_PC | REAL | manipulated variable / output variable 1 as a % |
| Y2_PC | REAL | manipulated variable / output variable 2 as a % |
| ERR1 | REAL | control difference P controller |
| ERR2 | REAL | control difference PI controller |

## Type Description

### PARA_CC2

| Parameter | Data Type | Meaning |
|---|---|---|
| SP_SPCV | BOOL | setpoint / setpoint + command variable |
| **P Controller** | | |
| YMAX1 | REAL | upper limit manipulated variable |
| YMIN1 | REAL | lower limit manipulated variable |
| GAIN1 | REAL | controller gain |
| PROP1 | REAL | proportional value |
| PREF1 | REAL | proportional value reference |
| **PI Controller** | | |
| YMAX2 | REAL | upper limit manipulated variable |
| YMIN2 | REAL | lower limit manipulated variable |
| GAIN2 | REAL | controller gain |
| PROP2 | REAL | proportional value |
| PREF2 | REAL | proportional value reference |
| TI2 | TIME | reset time |
| **1. Sequence** | | |
| X1_1 | REAL | 1. Abscissa value |
| Y1_1 | REAL | 1. Ordinate value |
| X2_1 | REAL | 2. Abscissa value |

| Parameter | Data Type | Meaning |
|---|---|---|
| Y2_1 | REAL | 2. Ordinate value |
| **2. Sequence** | | |
| X1_2 | REAL | 1. Abscissa value |
| Y1_2 | REAL | 1. Ordinate value |
| X2_2 | REAL | 2. Abscissa value |
| Y2_2 | REAL | 2. Ordinate value |

# Detailed Description

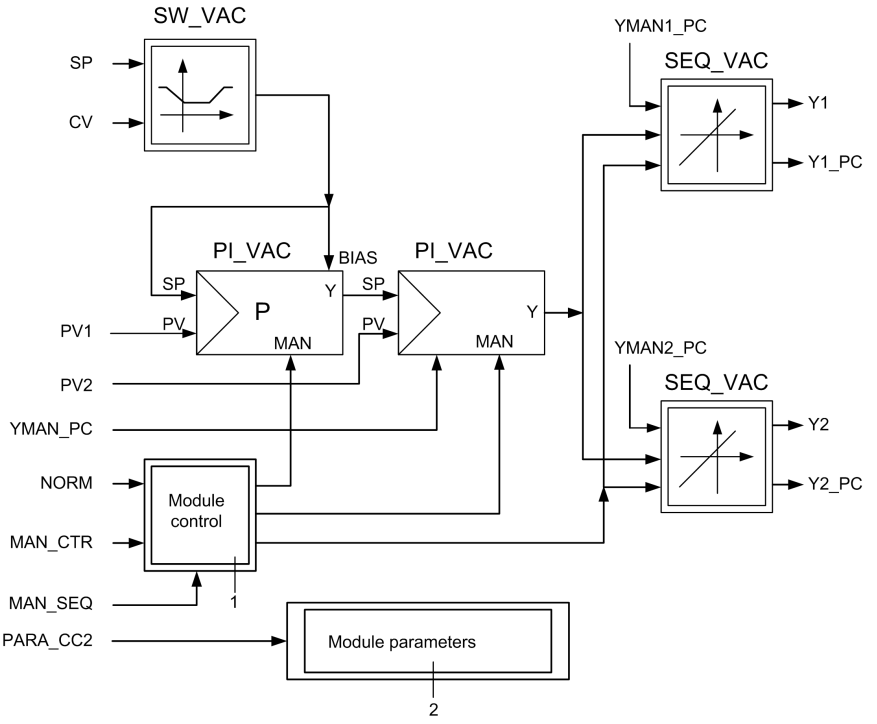
## EFB Structure

A block diagram representation of the CC2_VAC complex function block is shown in figure below. It is made up of the following basic function blocks:

- SW_VAC, page 73
- PI_VAC, page 62
- SEQ_VAC, page 67

Please refer to the respective individual basic function block description for detailed information.

Controller structure




**1** Module control

**2** Module parameters

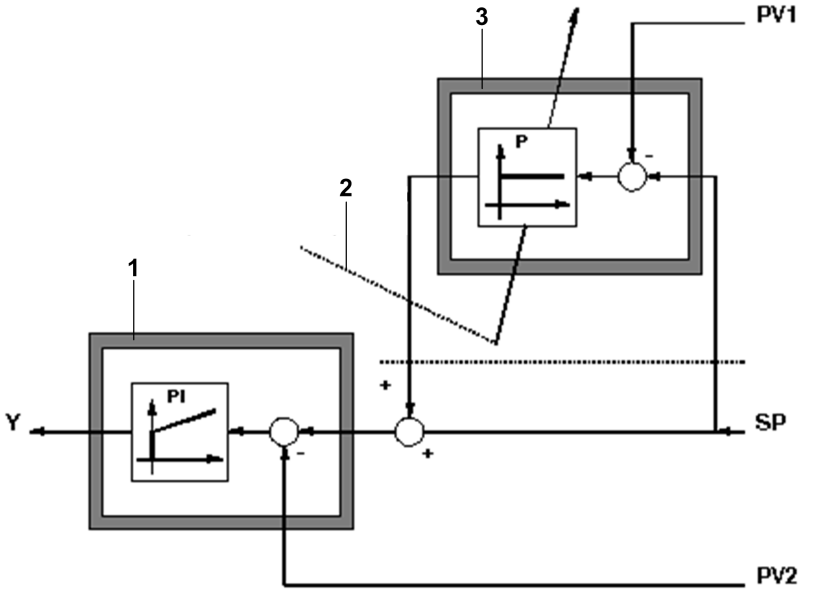
While the CC2_VAC function block can be used for both temperature or humidity control, the remainder of this description refers to temperature control only.

## Basic Operation

The setpoint for the CC2_VAC function block is fed to the P–controller via the summer/winter compensation block SW_VAC, page 73.

The output of the SW_VAC block is fed not only to the setpoint of the room P–controller, but also to its BIAS input. In this way, the P–controller setpoint is added to the P–controller output (which is equal to the P–controller deviation amplified by the controller gain), and the result is fed as setpoint to the inlet air PI–controller. This is shown in figure below.

Diagram of setpoint formation




**1** Inlet air

**2** P contribution adjustable from 0 to the point of instability

**3** Room air

**PV1** Room temperature

**SP** Direct setpoint or command variable

**PV2** Inlet air temperature

In order to understand the operation of this PPI controller, consider the case of a room with no temperature gains or losses. In this case, the room temperature PV1 would be equal to the PPI controller setpoint, and the inlet air temperature PV2 would be equal to the room temperature PV1.

If we now consider the case of a heat source in the room, the room air temperature PV1 will rise to a value greater than the setpoint. As a result, the output of the P–controller, equal to controller deviation (SP–PV1) multiplied by the GAIN, would trim the setpoint to the PI–controller resulting in a lowering of the inlet air temperature to offset the heat source in the room. Because the inlet air dynamics are much faster than the room air dynamics, a steady state condition will be reached under fluctuating room conditions using this cascaded approach.

A simple PI–controller using the room temperature as process variable would not achieve satisfactory control because of the **dead time** between the adjustment of the inlet air temperature and a reaction in the room air temperature.

PPI control algorithm will result in a steady room temperature, there will be a small offset from the PPI controller setpoint, i.e., PV1 will not equal SP. The size of the offset will depend on the disturbance conditions in the room. If an application requires tight control of temperature to an absolute value, the `CC2_VAC` function block should not be used. However, for applications that do not require tight absolute control, the PPI controller provides a simple, steady and easy–to–use control algorithm.

The PI–controller setpoint can be maintained within minimum and maximum limits by using the P–controller output limits YMAX1 and YMIN1. This facility can be used to avoid large swings in the inlet air temperature.

## Manual Operation

There are 3 possible modes of manual operation which are specified by setting the mutually exclusive NORM, MAN_CTR and MAN_SEQ parameters.

- **NORM mode (NORM = 1)**

  The output of the PI–controller output is set to zero. The zero value is fed to the output sequences that are active in NORM mode. As a result the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters. In the case where the controller output limits YMIN2 and YMAX2 have been set to values that do not enclose zero, the output will be set to the value of YMIN2 and YMAX2 that is closest to zero. In other words:

  Y = YMAX2 if YMAX2 < 0 AND YMIN2 < 0,

  Y = YMIN2 if YMAX2 > 0 AND YMIN2 > 0.

  Again, the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters.

- **MAN_CTR mode (MAN_CTR=1)**

  The output of the PI controller is set equal to the user–specified parameter YMAN_PC. Again, the output sequences are active in this mode and therefore the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters.

- **MAN_SEQ mode (MAN_SEQ=1)**

  The outputs Y1 and Y2 are set to the values specified by the parameters YMAN1_PC and YMAN2_PC, i.e., Y1_PC = YMAN1_PC and Y2_PC = YMAN2_PC.

In MAN_CTR mode, the I contribution of the PI–controller is tracked so that a bumpless transfer back to automatic mode may be carried out. In NORM and MAN–SEQ modes, the I contribution is set to zero.

## Output Parameters

The CC2_VAC outputs Y1 and Y2 are available in real or percentage form (Y1, Y2, Y1_PC and Y2_PC).

The percentage values specified for the manual control of outputs (YMAN_PC, YMAN1_PC and YMAN2_PC) refer to the specified range of the appropriate output. For example, the variable YMAN_PC sets the total output of the PI_VAC basic function block. The range of this output is specified by the parameters YMIN (0%) and YMAX (100%).

The variables YMAN1_PC and YMAN2_PC set the outputs of the 2 SEQ_VAC blocks, the ranges of which are specified by their corresponding ordinate vales Y1..Y2 (where Y1 and Y2 refer to the SEQ_VAC ordinates, not the actual outputs Y1 and Y2 of the 2 SEQ_VAC blocks).

**NOTE:** The smaller value of Y1 and Y2 is equated to 0% and the greater value to 100%. In other words, the percentage value is related to the size of the output variable Y irrespective of its direction:

- Y1 < Y2 –> 0%..100% = Y1..Y2
- Y1 > Y2 –> 0%..100% = Y2..Y1

For more information on the operation of the SEQ_VAC block, please refer to the respective description, page 67.

# CC3_VAC: Cascade Controller for Air Conditioning with 3 Outputs

## What's in This Chapter



# Introduction

This chapter describes the CC3_VAC function block.

# Brief Description

## Function Description

The CC3_VAC block is a cascade controller used to provide temperature or humidity control of the inlet air to a room. It consists of a P–only outer loop which uses the room temperature/humidity as process variable and an inner PI loop that controls the temperature/humidity of the inlet air supplying the room.

The EFB has a fixed structure where the setpoint of the P–controller is fed forward and added to the P–controller output to form the setpoint of the PI–controller. The output of the PI–controller has 3 output sequences, Y1, Y2 and Y3. The operation of CC3_VAC is identical to CC2_VAC, the only difference being the additional output.

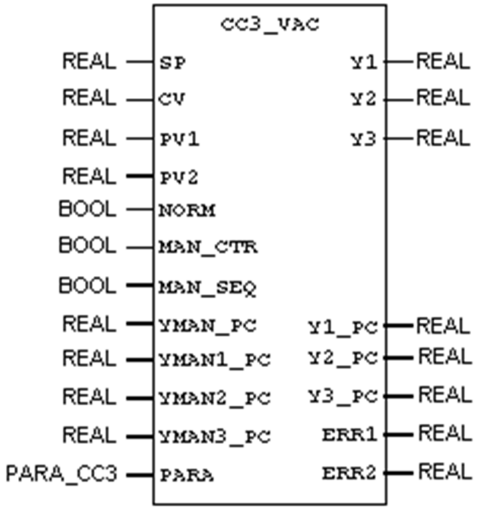
The EFB provides the following:

- winter/summer setpoint compensation as per DIN 1946 part 2
- 4 quadrant operation of the output sequence scaling
- display of output variables as percentages
- upper and lower limits on outputs
- presetting the controllers gains in the form of GAIN or PROP with the possibility of using negative values for switching the control direction
- operation with Anti–Windup–Reset (AWR)
- manual adjustment of either the PI–controller output or the individual sequence outputs (Y1 and Y2) using percentages

  When the controller output is manually adjusted, the EFB tracks the I contribution in order to provide bumpless switching back to automatic mode.

- display of the P and PI controller deviations (SP–PV)

## Representation




## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP | REAL | setpoint |
| CV | REAL | command variable |
| PV1 | REAL | actual value P controller |
| PV2 | REAL | actual value PI controller |
| NORM | BOOL | basic setting |
| MAN_CTR | BOOL | MANUAL for controller |
| MAN_SEQ | BOOL | MANUAL for sequences |
| YMAN_PC | REAL | total manual manipulated variable as a % for controller |
| YMAN1_PC | REAL | individual manual manipulated variable as a %, 1st sequence |
| YMAN2_PC | REAL | individual manual manipulated variable as a %, 2nd sequence |

| Parameter | Data Type | Meaning |
|---|---|---|
| YMAN3_PC | REAL | individual manual manipulated variable as a %, 3rd sequence |
| PARA | PARA_CC3 | parameter structure |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y1 | REAL | manipulated variable / output variable 1 |
| Y2 | REAL | manipulated variable / output variable 2 |
| Y3 | REAL | manipulated variable / output variable 3 |
| Y1_PC | REAL | manipulated variable / output variable 1 as a % |
| Y2_PC | REAL | manipulated variable / output variable 2 as a % |
| Y3_PC | REAL | manipulated variable / output variable 3 as a % |
| ERR1 | REAL | control difference P controller |
| ERR2 | REAL | control difference PI controller |

## Type Description

PARA_CC3

| Parameter | Data Type | Meaning |
|---|---|---|
| SP_SPCV | BOOL | setpoint / setpoint + command variable |
| **P controller** | | |
| YMAX1 | REAL | upper limit manipulated variable |
| YMIN1 | REAL | lower limit manipulated variable |
| GAIN1 | REAL | controller gain |
| PROP1 | REAL | proportional value |
| PREF1 | REAL | proportional value reference |
| **PI Controller** | | |
| YMAX2 | REAL | upper limit manipulated variable |
| YMIN2 | REAL | lower limit manipulated variable |
| GAIN2 | REAL | controller gain |
| PROP2 | REAL | proportional value |

| Parameter | Data Type | Meaning |
|---|---|---|
| PREF2 | REAL | proportional value reference |
| TI2 | TIME | reset time |
| **1. Sequence** | | |
| X1_1 | REAL | 1. abscissa value |
| Y1_1 | REAL | 1. ordinate value |
| X2_1 | REAL | 2. abscissa value |
| Y2_1 | REAL | 2. ordinate value |
| **2. Sequence** | | |
| X1_2 | REAL | 1. abscissa value |
| Y1_2 | REAL | 1. ordinate value |
| X2_2 | REAL | 2. abscissa value |
| Y2_2 | REAL | 2. ordinate value |
| **3. Sequence** | | |
| X1_3 | REAL | 1. abscissa value |
| Y1_3 | REAL | 1. ordinate value |
| X2_3 | REAL | 2. abscissa value |
| Y2_3 | REAL | 2. ordinate value |

# Detailed Description

## EFB Structure

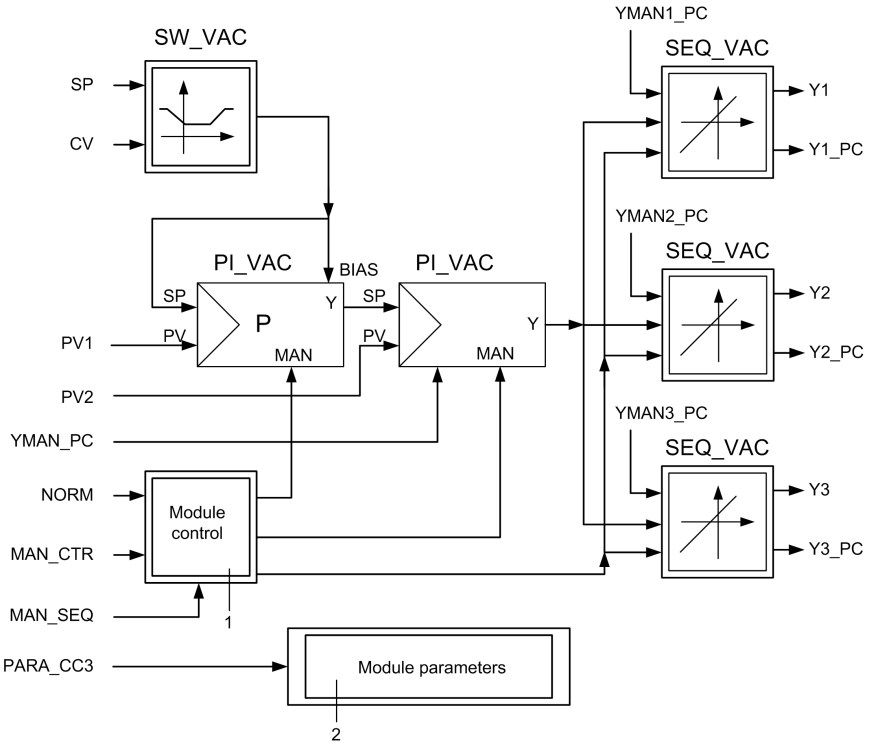
A block diagram representation of the `CC3_VAC` complex function block is shown in figure below.

It is made up of the following basic function blocks:

- `SW_VAC, page 73`
- `PI_VAC, page 62`
- `SEQ_VAC, page 67`

Please refer to the respective individual basic function block description for detailedinformation.

Controller structure



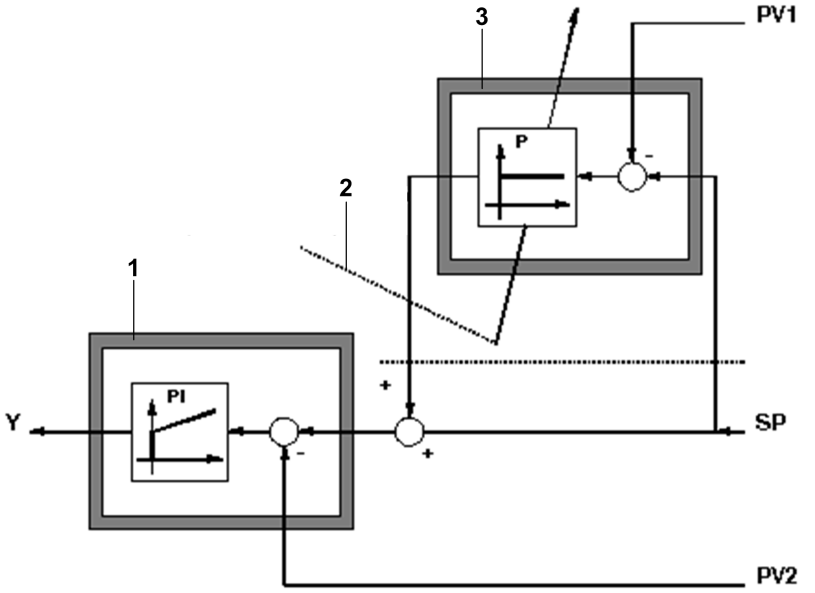

While the CC3_VAC function block can be used for both temperature or humidity control, the remainder of this description refers to temperature control only.

## Basic Operation

The setpoint for the CC3_VAC function block is fed to the P–controller via the summer/winter compensation block SW_VAC, page 73

The output of the SW_VAC block is fed not only to the setpoint of the room P–controller, but also to its BIAS input. In this way, the P–controller setpoint is added to the P–controller output (which is equal to the P–controller deviation amplified by the controller gain), and the result is fed as setpoint to the inlet air PI–controller. This is shown in figure below.

Diagram of setpoint formation




**1** Inelt air

**2** P contribution adjustable from 0 to the point of instability

**3** Room air

**PV1** Room temperature

**SP** Direct setpoint or command variable

**PV2** Inlet air temperature

In order to understand the operation of this PPI controller, consider the case of a room with no temperature gains or losses. In this case, the room temperature PV1 would be equal to the PPI controller setpoint, and the inlet air temperature PV2 would be equal to the room temperature PV1. If we now consider the case of a heat source in the room, the room air temperature PV1 will rise to a value greater than the setpoint.

As a result, the output of the P–controller, equal to controller deviation (SP–PV1) multiplied by the GAIN, would trim the setpoint to the PI–controller resulting in a lowering of the inlet air temperature to offset the heat source in the room. Because the inlet air dynamics are much faster than the room air dynamics, a steady state condition will be reached under fluctuating room conditions using this cascaded approach.

A simple PI–controller using the room temperature as process variable would not achieve satisfactory control because of the **dead time** between the adjustment of the inlet air temperature and a reaction in the room air temperature.

PPI control algorithm will result in a steady room temperature, there will be a small offset from the PPI controller setpoint, i.e., PV1 will not equal SP. The size of the offset will depend on the disturbance conditions in the room. If an application requires tight control of temperature to an absolute value, the `CC3_VAC` function block should not be used. However, for applications that do not require tight absolute control, the PPI controller provides a simple, steady and easy–to–use control algorithm.

The PI–controller setpoint can be maintained within minimum and maximum limits by using the P–controller output limits YMAX1 and YMIN1. This facility can be used to avoid large swings in the inlet air temperature.

## Manual Operation

There are 3 possible modes of manual operation which are specified by setting the mutually exclusive NORM, MAN_CTR and MAN_SEQ parameters.

- **NORM mode (NORM = 1)**

  The output of the PI–controller output is set to zero. The zero value is fed to the output sequences that are active in NORM mode. As a result the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters. In the case where the controller output limits YMIN2 and YMAX2 have been set to values that do not enclose zero, the output will be set to the value of YMIN2 and YMAX2 that is closest to zero. In other words:

  Y = YMAX2 if YMAX2 < 0 AND YMIN2 < 0,

  Y = YMIN2 if YMAX2 > 0 AND YMIN2 > 0.

  Again, the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters.

- **MAN_CTR mode (MAN_CTR=1)**

  The output of the PI controller is set equal to the user–specified parameter YMAN_PC. Again, the output sequences are active in this mode and therefore the actual values of Y1, Y2 and Y3 that are output to the control actuators will depend on the output sequence parameters.

- MAN_SEQ mode (MAN_SEQ=1)

  The outputs Y1, Y2 and Y3 are set to the values specified by the parameters YMAN1_PC, YMAN2_PC and YMAN3_PC, i.e., Y1_PC = YMAN1_PC, Y2_PC = YMAN2_PC and Y3_PC = YMAN3_PC.

In MAN_CTR mode, the I contribution of the PI–controller is tracked so that a bumpless transfer back to automatic mode may be carried out. In NORM and MAN–SEQ modes, the I contribution is set to zero.

## Output Parameters

The CC3_VAC outputs Y1, Y2 and Y3 are available in real or percentage form (Y1, Y2, Y3, Y1_PC, Y2_PC and Y3_PC).

The percentage values specified for the manual control of outputs (YMAN_PC, YMAN1_PC, YMAN2_PC and YMAN3_PC) refer to the specified range of the appropriate output. For example, the variable YMAN_PC sets the total output of the PI_VAC basic function block. The range of this output is specified by the parameters YMIN (0%) and YMAX (100%).

The variables YMAN1_PC, YMAN2_PC and YMAN3_PC set the outputs of the 2 SEQ_VAC blocks, the ranges of which are specified by their corresponding ordinate vales Y1..Y2 (where Y1 and Y2 refer to the SEQ_VAC ordinates, not the actual outputs Y1, Y2 and Y3 of the 2 SEQ_VAC blocks).

**NOTE:** The smaller value of Y1 and Y2 is equated to 0% and the greater value to 100%. In other words, the percentage value is related to the size of the output variable Y irrespective of its direction:

- Y1 < Y2 –> 0%..100% = Y1..Y2
- Y1 > Y2 –> 0%..100% = Y2..Y1

For more information on the operation of the SEQ_VAC block, please refer to the respective description, page 67.

# MC_VAC: Air Mix Controller for Air Conditioning with 1 Output

## What's in This Chapter



# Introduction

This chapter describes the MC_VAC function block.

# Brief Description

## Function Description

The MC_VAC block is a controller designed for applications that require switching of the controller direction, and maximum/minimum selection of outputs. Examples of such applications would be air mixing controls where the control direction will depend on the relative values of the outside air temperature and return air temperature, or heat exchanger controls used in energy conservation schemes.

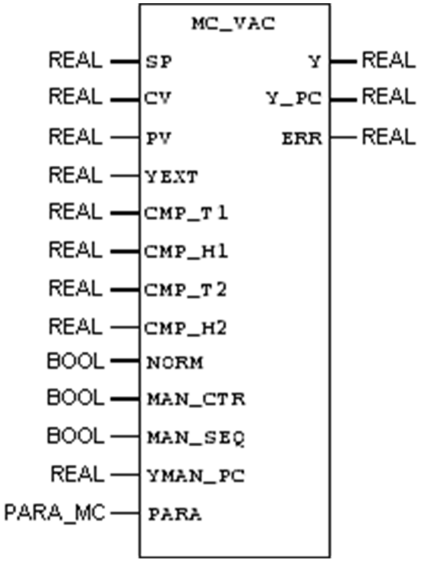
The EFB provides the following:

- winter/summer setpoint compensation as per DIN 1946 part 2
- 4 quadrant operation of the output sequence scaling
- display of output variables as percentages
- upper and lower limits on outputs
- presetting the controllers' gains in the form of GAIN or PROP with the possibility of using negative values for switching the control direction
- control reversal using comparison inputs with built–in hysteresis
- selection of operating mode as a "Direct Controller" or "Auxiliary Controller" with max/min selection
- operation with Anti–Windup–Reset (AWR)
- manual adjustment of either the PI–controller output or the individual sequence outputs (Y1 and Y2) using percentages

When the controller output is manually adjusted, the EFB tracks the I contribution in order to provide bumpless switching back to automatic mode.

- display of the P and PI controller deviations (SP–PV)

## Representation




## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP | REAL | setpoint |
| CV | REAL | command variable |
| PV | REAL | actual value |
| YEXT | REAL | manipulated variable of higher–level controller in the case of cascade mode |
| CMP _T1 | REAL | reference value T1 |
| CMP_H1 | REAL | reference value H1 |
| CMP_T2 | REAL | reference value T2 |
| CMP_H2 | REAL | reference value H2 |
| NORM | BOOL | basic setting |

| Parameter | Data Type | Meaning |
|---|---|---|
| MAN_CTR | BOOL | MANUAL for controller |
| MAN_SEQ | BOOL | MANUAL for sequence |
| YMAN_PC | REAL | total manual manipulated variable as a % or sequence manual manipulated variable as a % |
| PARA | PARA_MC | parameter structure |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y | REAL | manipulated variable / output variable |
| Y_PC | REAL | manipulated variable / output variable as a % |
| ERR | REAL | system deviation |

## Type Description

PARA_MC

| Parameter | Data Type | Meaning |
|---|---|---|
| CASC | BOOL | direct / cascade mode |
| SP_SPCV | BOOL | setpoint / setpoint + command variable |
| MIN | BOOL | max/min switch (max=0) |
| **PI Controller** | | |
| YMAX | REAL | upper limit of manipulated variable (YAU ever 0) |
| GAIN | REAL | controller gain |
| PROP | REAL | proportional value |
| PREF | REAL | proportional value reference |
| TI | TIME | reset time |
| **Output sequence** | | |
| X1 | REAL | 1. abscissa value |
| Y1 | REAL | 1. ordinate value |
| X2 | REAL | 2. abscissa value |
| Y2 | REAL | 2. ordinate value |

# Detailed Description

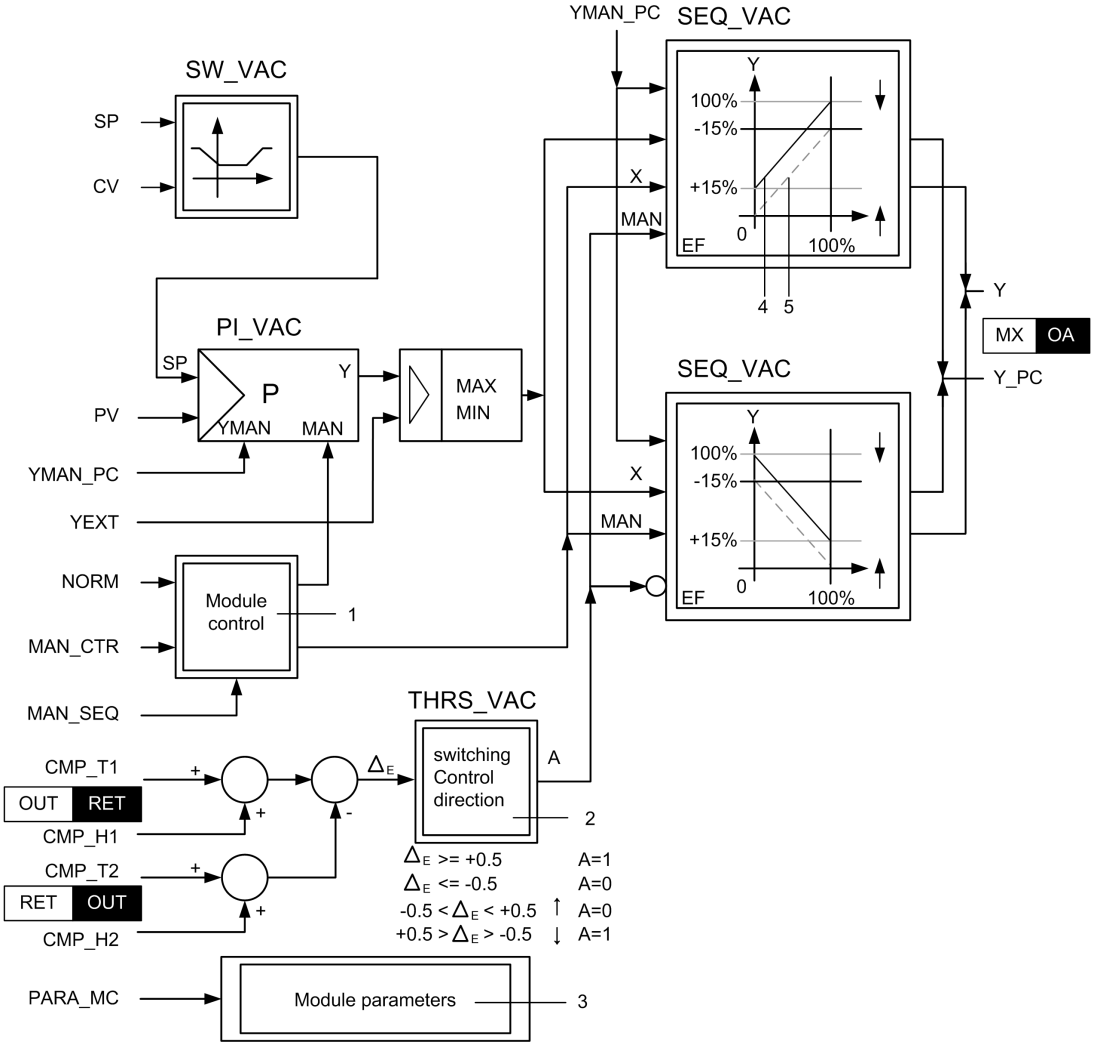
## EFB Structure

A block diagram representation of the MC_VAC complex function block is shown in figure below.

It is made up of the following basic function blocks:

- SW_VAC, page 73
- PI_VAC, page 62
- THRS_VAC, page 77
- SEQ_VAC, page 67

## Controller structure




**1** Module control

**2** Switching control direction

**3** Module parameters

**4** Unbroken line = outside air damper activated

**5** Broken line = mixed air damper activated

**MX** Mixed air damper

**OA** Outside air damper

**OUT** Outside air temperature/humidity

**RET** Return air temperature/humidity

Please refer to the respective individual basic function block description for detailed information.

## Setpoint

The setpoint for the MC_VAC function block is fed to the PI–controller via the summer/winter compensation block SW_VAC, page 73.

The PI–controller setpoint can be maintained within minimum and maximum limits by using the P–controller output limits YMAX1 and YMIN1. This facility can be used to avoid large swings in the inlet air temperature as a result of unusual room temperature fluctuations or a badly adjusted P–controller.

## Control Direction

The function block has 4 reference variables CMP–T1, CMP_H1, CMP_T2 andCMP_H2. CMP_T1 is added to CMP_H1 and CMP_T2 is added to CMP_H2.

The resulting additions are then compared with one another, and the result of the comparison is fed to the THRS_VAC block to provide switching with a built in hysteresis of 1.0. The output of the THRS_VAC block is used to ”reverse” the action of the output sequences.

The user is free to choose what inputs are connected to the reference variables. In general though, temperature comparisons should use CMP_T1 and CMP_T2 while humidity comparisons should use CMP_H1 and CMP_H2. If temperature comparison only is required, the variables CMP_H1 and CMP_H2 can be set to zero.

The comparison inputs can be used to compare the heat content of 2 air streams. The heat content of air can be calculated as follows:

$Q = Q_a + Q_{sw} + Q_{lw}$

Where

$Q_a$ = sensible heat content of air (kJ)

$Q_{sw}$ = sensible heat content of water vapor (kJ)

$Q_{lw}$ = latent heat content of water vapor (kJ)

Therefore

$Q = m * C_a * T_a + m * x * C_w * T^a + m * x * h_w$

Where

m = mass of air (kg)

$C_a$ = specific heat capacity of air = 1.006 kJ/(kg*K)

$T_a$ = temperature of air (deg C)

x = absolute humidity (g/kg)

$C_w$ = specific heat capacity of water vapor = 1.92 kJ/(kg*K)

$h_w$ = specific enthalpy of water vapor = 2500 kJ/kg = 2.5 kJ/g

Bearing in mind that $C_a$ is approximately equal to 1 and that $h_w$ is much greater than $C_w*T_a$, the equation can be simplified to:

$Q = m * T_a + m * h_w * x$

Therefore, the enthalpy H of the air can be calculated as:

$H = Q / m = T_a + h_w* x = T_a + 2.5x$

Therefore by scaling the air temperature in degrees Celsius and by multiplying the absolute humidity (scaled in g/kg) by 2.5 the resultant addition gives a good indication of the enthalpy content of the air in kJ/kg. In this case, the built in hysteresis corresponds to 1.0 kJ/kg.

In cases where an accurate value for hysteresis is not necessary, the absolute humidity can be substituted by its corresponding dew point temperature. This can be obtained by the use of `WASH_VAC`, feeding the measured relative humidity to SP_RH and the measured temperature $T_a$ to SP_RT. The resulting SP_TW replaces in this case the term 2.5x.

$H = T_a$ + SP_TW

SP_TW represents a nonlinear but fixed function or the unknown absolute humidity. Because both values of H, from RETURN AIR and OUTSIDE AIR, are calculated in the same way, the comparison of both values does not need to take care that these calculated values themselves are nonlinear to the true H values.

During the first execution cycle of the function block, the hysteresis is initialized on the assumption that CMP_T1 < CMP_T2. If after the first calculation cycle the comparison result is within the hysteresis band, the initial switching status is retained.

## Output Sequences

**NOTE:** Only 1 output sequence is specified by the user. The function block will then calculate the reversed sequence when the control direction is switched. The sequence specified needs to have an increasing slope, i.e., Y2>Y1.

In order to understand how the output sequencing works, the example of temperature control using a mixing air damper can be used.

For a mixing air controller, there are normally 2 dampers – a control damper for mixing the return air with the outside air and a control damper on the outside air inlet, see example applications, page 57.

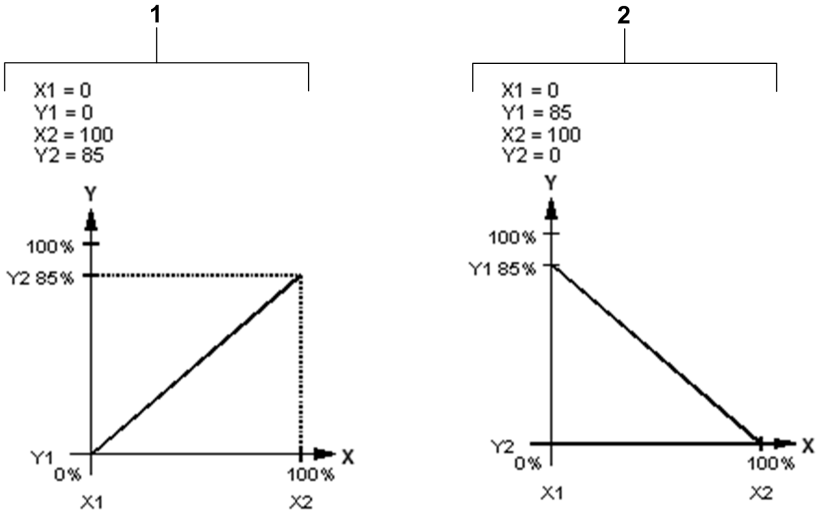
The control action on 1 damper is the reverse of the other. In practical terms, the controller output is sent to 1 damper only, the other damper being controlled mechanically or electrically. It therefore follows that the output sequences will depend on which damper is being controlled directly.

Also, a minimum outside air flow is commanded by the program to allow fresh air into the room.

Assuming an outside air minimum of 15%, the output sequences would look like the following.

If the air mixer damper is controlled, the sequences would be:

**Air Mixing Damper**

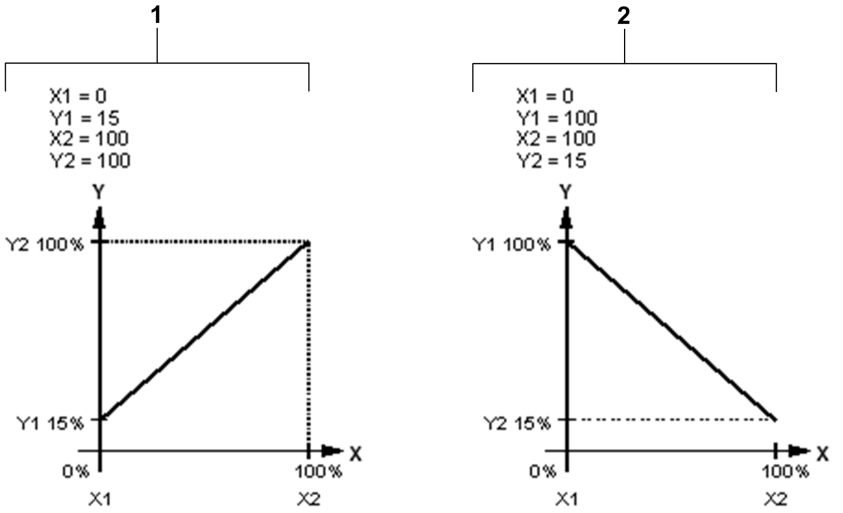



**1** Sequence 1, outside air temperature -> return air temperature

**2** Sequence 2, outside air temperature <- return air temperature

If the outside air damper is controlled, the sequences would be:

**Outside Air Damper**




**1** Sequence 1, outside air temperature <- return air temperature

**2** Sequence 2, outside air temperature -> return air temperature

The reference values CMP_T1 and CMP_T2 will determine which of the sequences are activated as follows:

CMP_T1 > CMP_T2 => Sequence 1 is active

CMP_T1 < CMP_T2 => Sequence 2 is active

In order to get the desired control, the comparison inputs have to be connected as follows.

- If the air mixer damper is controlled:

  CMP_T1 = Outside air temperature

  CMP_T2 = Return air temperature
- If the outside air damper is controlled:

  CMP_T1 = Return air temperature

  CMP_T2 = Outside air temperature

## Direct and Auxiliary Control

Direct or auxiliary control is selected using the CASC parameter.

Direct control is specified by setting CASC = 0. In this mode, the Max/Min selection is disable and the external input YEXT is ignored.

Auxiliary control is specified by setting CASC = 1. In this mode, the Max/Min selection is enabled and the external input YEXT is compared to the PI controller output.

## Manual Operation

There are 3 possible modes of manual operation which are specified by setting the mutually exclusive NORM, MAN_CTR and MAN_SEQ parameters.

- **NORM mode (NORM = 1):**

  The output of the PI–controller output is set to zero. However, the actual value forwarded to the output sequences depends on whether the controller is set up in ”Direct” or ”Auxiliary mode. In ”Direct” mode the PI–controller output is fed through the Max/Min block to the output sequence. In ”Auxiliary” mode, the value passed through to the output sequence depends on the result of the comparison between the controller output and the value of YEXT which is set by the higher level controller. Therefore, in order to operate in true NORM mode, both the Auxiliary controller and the higher level controller has to be set to NORM. The actual value that is output to the control actuator depends on the output sequence parameters.

- **MAN_CTR mode (MAN_CTR=1):**

  The output of the PI controller is set equal to the user–specified parameter YMAN_PC. However, the YEXT value coming from the higher level controller may override the YMAN_PC value. Again, the output sequence is active in this mode and therefore the actual value of Y that is output to the control actuator will depend on the output sequence parameters and the control direction.

- **MAN_SEQ mode (MAN_SEQ=1):**

  The sequence output Y is set to the value specified by the parameter YMAN_PC, i.e., Y_PC = YMAN_PC. When in MAN_SEQ mode, switching the control direction has no effect on the output.

In MAN_CTR mode, the I contribution of the PI–controller is tracked so that a bumpless transfer back to automatic mode may be carried out. In NORM and MAN–SEQ modes, the I contribution is set to zero.

## Output Parameters

The MC_VAC output Y is available in real or percentage form (Y, and Y_PC).

The percentage values specified for the manual control of outputs (YH_PC, and YMAN_PC) refer to the specified range of the appropriate output. For example, the variable YH_PC sets the total output of the PI_VAC basic function block. The range of this output is specified by the parameters YMIN (0%) and YMAX (100%).

The variable YMAN_PC sets the output of the SEQ_VAC block, the range of which is specified by the corresponding ordinate vales Y1..Y2 (where Y1 and Y2 refer to the SEQ_VAC ordinates).

**NOTE:** The smaller value of Y1 and Y2 is in principle equated to 0% and the greater value to 100%. In other words, the percentage value is related to the size of the output variable Y irrespective of its direction:

- Y1 < Y2 –> 0%..100% = Y1..Y2
- Y1 > Y2 –> 0%..100% = Y2..Y1

For more information on the operation of the SEQ_VAC block, please refer to the respective description, page 67.

# Example Applications

## General

You will find 2 examples for using the MC_VAC block:

- Air Mixing
- Heat Recovery Exchangers

**NOTE:** In both of the examples, select control actuators for proper operation and to avoid freezing of water coils. Hardwired antifreeze controls should also be used to override the controller controls and if necessary shutdown the make up air handling unit to safeguard the coils.

| ***NOTICE*** |
| --- |
| **UNEXPECTED BEHAVIOR OF EQUIPMENT**<br>Use hard-wired antifreeze controls to safeguard your equipment.<br>**Failure to follow these instructions can result in equipment damage.** |

## Air Mixing

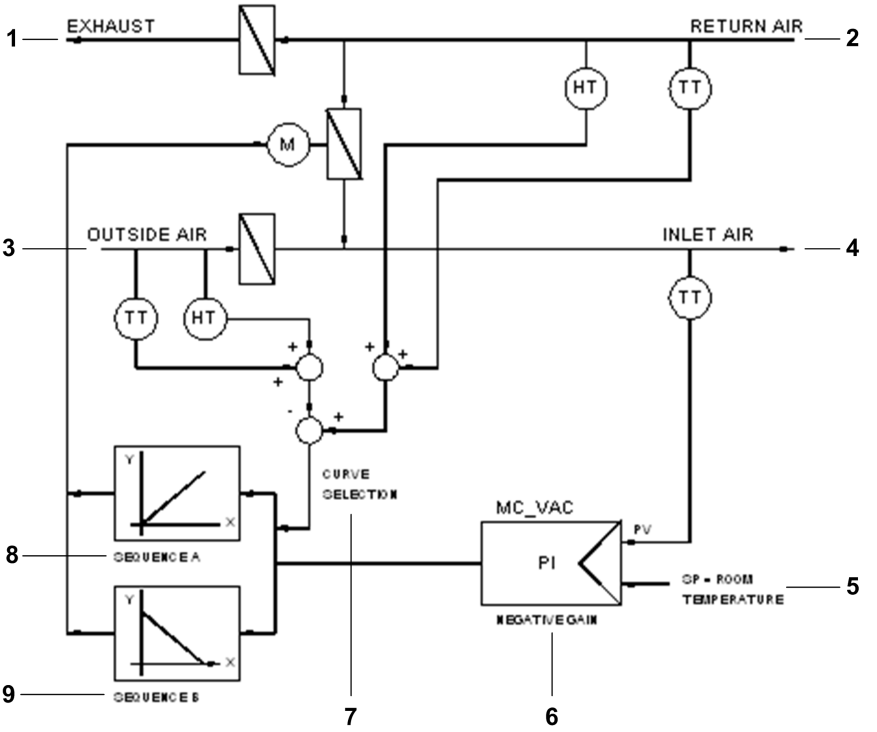
An example where switching of control direction is commonly used is air mixing and is shown in figure below.

In this application, return air from a room is mixed with the outside air to supply the inlet air to the room. In the winter, the outside air is at a lower temperature and humidity than the return air. The return air is therefore mixed with the outside air to provide free heating of the outside air. In the summer, the return air may have a lower temperature and humidity than the outside air. In this case, the return air is mixed with the outside air to provide free cooling.

The example shows the use of temperature and humidity sensors on the return air and outside air being used to determine the control direction of the controller.

The example also shows control of the mixing air damper. It is then assumed that the outside air damper is connected mechanically or electro–mechanically to the mixing air damper.

Example for Air Mixing




**1** Exhaust

**2** Return air

**3** Outside air

**4** Inlet air

**5** SP = Room temperature

**6** Negative gain

**7** Curve selection

**8** Sequence A

**9** Sequence B

In the winter, if H_OUTSIDE < H_RETURN, sequence B is used. In this case it can be seen that if the inlet air temperature is too low, the PI controller output (whose gain is negative) will be increasingly negative, opening the mixing air damper in order to mix a greater quantity of warmer air with the colder outside air, thereby increasing the inlet air temperature. Conversely, if the inlet air temperature is too warm, the PI controller output will be increasingly more positive, resulting in a closing of the mixing air damper in order to mix a smaller quantity of the warmer return air with the cooler outside air, thereby decreasing the inlet air temperature.

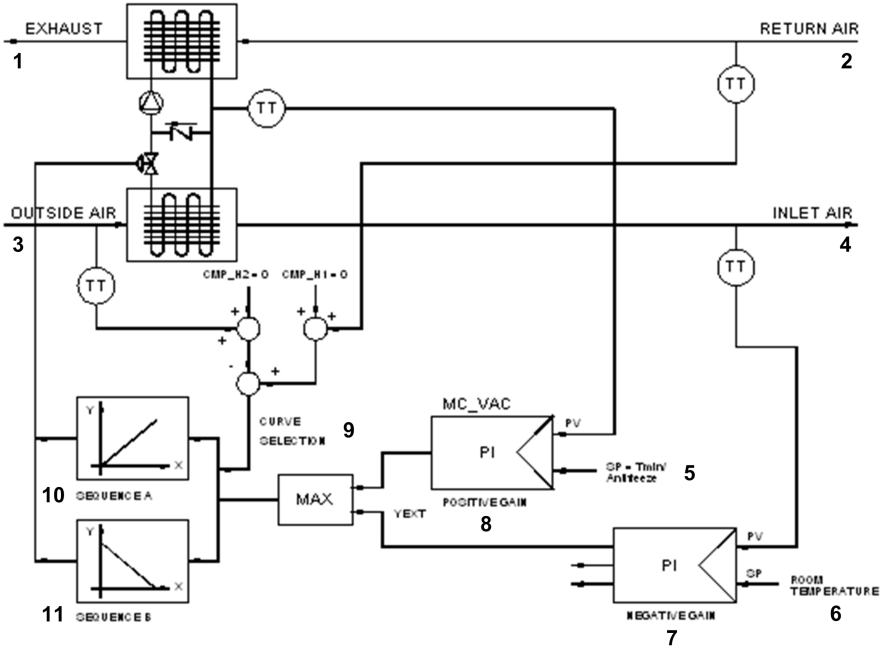
In the summer, if H_OUTSIDE > H_RETURN, sequence A is used. In this case it can be seen that if the inlet air temperature is too low, the PI controller output (whose gain is negative) will be increasingly negative, closing the mixing air damper in order to mix a smaller quantity of colder return air with the warmer outside air, thereby increasing the inlet air temperature. Conversely, if the inlet air temperature is too warm, the PI controller output will be increasingly more positive, resulting in the opening of the mixing air damper in order to mix a greater quantity of cooler return air with the warmer outside air, thereby decreasing the inlet air temperature.

## Heat Recovery Exchangers

Another example of heat recovery exchangers is shown in figure below.

In this example, a closed water loop between 2 heat exchangers is used to transfer heat between the return air and incoming outside air. A re–circulation pump is used in combination with a check valve and control valve. Closing the control valve will result in more of the water circulating through the check valve and less through the outside air heat exchanger. Conversely, opening the control valve, will result in more water circulating through the outside air heat exchanger.

Example Heat Recovery Exchangers




**1** Exhaust

**2** Return air

**3** Outside air

**4** Inlet air

**5** SP = $T_{min}$/antifreeze

**6** Room temperature

**7** Negative gain

**8** Positive gain

**9** Curve selection

**10** Sequence A

**11** Sequence B

The example shows 2 control loops – an inlet air controller and an anti–freeze controller.

The anti–freeze controller measures the water loop temperature and provides a command for minimum temperature so as to avoid freezing of return air condensed water on the return air exchanger surfaces.

The setpoint is set to a suitable value above the freezing point of water. Under normal conditions, the PV will be greater than the SP, the controller output will be zero, and the MAX selection block allows the inlet air controller to control the heat exchangers.

If however, the water temperature decreases below the minimum setpoint, the anti–freeze controller output will increase and take over from the inlet air controller to maintain the minimum water temperature through the return air heat exchanger, thereby helping to avoid freezing of any condensation that may have occurred.

The inlet air controller controls the inlet air temperature. An `MC_VAC` block is used for the anti–freeze controller and is run in CASCADE mode. The output of the inlet air controller is fed as the input Yext to the anti–freeze controller, thereby controlling the heat exchanger control valve.

If T_OUTSIDE < T_RETURN, sequence B is used. In this case it can be seen that if the inlet air temperature is too low, the PI controller output (whose gain is negative) will be increasingly negative, opening the control valve in order to divert a greater quantity of warmer water to the outside air heat exchanger, thereby increasing the inlet air temperature. Conversely, if the inlet air temperature is too warm, the PI controller output will be increasingly more positive, resulting in a closing of the control valve in order to divert a smaller quantity of the warmer water to the outside air heat exchanger, thereby decreasing the inlet air temperature.

If T_OUTSIDE > T_RETURN, sequence A is used. In this case it can be seen that if the inlet air temperature is too low, the PI controller output (whose gain is negative) will be increasingly negative, closing the control valve in order to divert a smaller quantity of cooler water to the outside air heat exchanger, thereby increasing the inlet air temperature. Conversely, if the inlet air temperature is too warm, the PI controller output will be increasingly more positive, resulting in the opening of the control valve in order to divert a greater quantity of the cooler water to the outside air heat exchanger, thereby decreasing the inlet air temperature.

# PI_VAC: PI Controller for Air Conditioning

## What's in This Chapter



# Introduction

This chapter describes the `PI_VAC` function block.

# Brief Description

## Function Description

The `PI_VAC` basic function block is a general PI–controller designed for air conditioning applications. It can be used for temperature control, humidity control, air mixing control or other general functions. As it is a basic function block it consists of a PI controller only with no setpoint compensation, output sequencing, or other functions. It is used extensively by the complex function blocks.

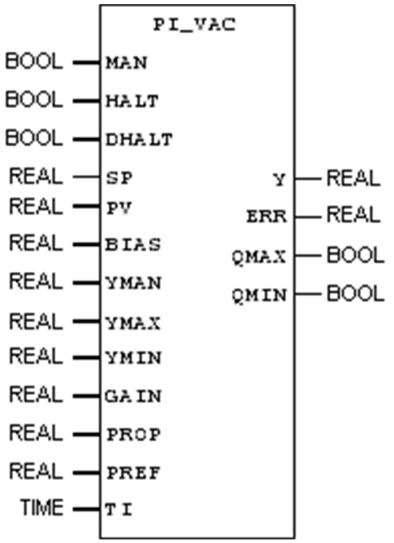
Where the standard complex function block library does not meet the requirements of a particular application, the `PI_VAC` basic function block can be combined with other basic function blocks to create the user's own complex function block using the Derived Function Block facilities.

The EFB provides the following:

- SP (setpoint), PV (process variable) and BIAS inputs
- ability to operate in P, I, or PI modes
- bumpless initialization of the I contribution as well as bumpless switching between I and PI operation
- presetting the controller's gain in the form of GAIN or PROP with the possibility of using negative values for switching the control direction
- bumpless switching between GAIN and PROP operation in the case of PI control
- operation with Anti–Windup–Reset (AWR)
- manual operation mode with tracking of the I contribution in order to provide bumpless switching back to automatic mode
- HALT operating mode for freezing the controller output at its current value with tracking of the I contribution

- DYNAMIC HALT operating mode that helps to avoid step–changes in the controller output when the setpoint is changed
- upper and lower limits on controller output with indication when limits are reached using the QMAX and QMIN outputs
- display of the PI controller deviation (SP–PV)

## Representation




## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| MAN | BOOL | manual mode |
| HALT | BOOL | halt mode |
| DHALT | BOOL | dynamic halt for next scanning step |
| SP | REAL | setpoint |
| PV | REAL | actual value |
| BIAS | REAL | deviation (deviation compensation) |
| YMAN | REAL | manual manipulated variable |
| YMAX | REAL | upper limit manipulated variable |

| Parameter | Data Type | Meaning |
|---|---|---|
| YMIN | REAL | lower limit manipulated variable |
| GAIN | REAL | controller gain |
| PROP | REAL | proportional value |
| PREF | REAL | proportional value reference |
| TI | TIME | integral time |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y | REAL | manipulated variable |
| ERR | REAL | control deviation |
| QMAX | BOOL | upper limit of signalling device reached |
| QMIN | BOOL | lower limit of signalling device reached |

# Detailed Description

## Parameter Specifications

### SP, PV, BIAS

The setpoint SP and process variable PV are connected directly to the function block as real values. The controller deviation is calculated as:

ERR = SP – PV.

The deviation value is available for display.

A BIAS value may also be specified as a real value. This BIAS value is added to the result of the PI calculation. In this way, the user can verify that the controller output is ”biased” with a specific value during the first cycle. This is shown in figure below.

### GAIN, PROP, PREF, TI

The controller gain can be specified by either setting a value for the GAIN, or alternatively, by specifying the proportional rate using the PROP and PREF parameters as follows:

- **If GAIN is not equal to zero,**

  then the GAIN mode is activated. In this case, the controller output for P–only control is then calculated as:

Y = GAIN * (SP – PV)

A negative value of GAIN may be specified to reverse the action of the controller.

- **If GAIN = 0 and PROP is not equal to zero,**

  then the PROP mode is activated. In this mode, a change in the PV by an amount PROP will result in a change of the controller output by an amount PREF. In other words, for a P–only controller, the output is calculated as:

  Y = PREF * (SP – PV) / PROP

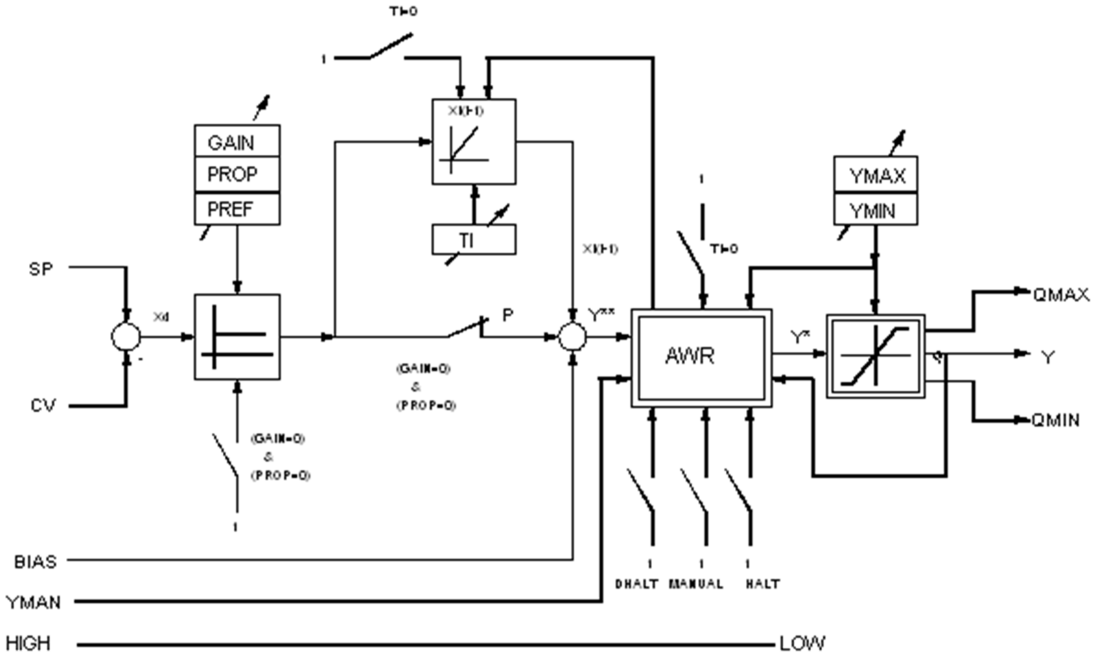
  In the case where an actuator is connected directly to the output of the controller, a PREF of 100 can be set to indicate 100% output.

  To reverse the action of the controller set the value of PROP as negative and the value of PREF as positive.

- **If GAIN = 0 and PROP = 0,**

  the proportional value of the controller is effectively switched off. By specifying an integral time using the TI input, the controller will act as an I–only controller. In the event that the integral time TI is set to zero, there will be no PI action, and only the BIAS value will be forwarded to the controller output. In this case, the operating modes MAN, HALT and DHALT will still be active.

Controller structure




Priority controller operating mode:

Operating mode switch over (P, I, PI or GAIN, PROP)...DHALT...MAN...Limitation

## Manual Operation

There are 3 possible modes of manual operation which are specified by setting the mutually exclusive MAN, HALT and DHALT parameters.

- **MAN mode (MAN = 1):**

  The value specified at YMAN is written to the controller loop output Y. This is effective in both P and PI modes. The I contribution is continually tracked to allow bumpless switching of the controller back to automatic mode. The output limits and anti–reset–windup are therefore still active in MAN mode.

- **HALT mode (HALT = 1):**

  In HALT mode the controller output is frozen at its current value. The I contribution is tracked to allow bumpless switching back to automatic mode.

- **DHALT mode (DHALT=1):**

  This mode is used to avoid step changes to the controller output when the setpoint is changed. When the setpoint is changed, the controller I contribution is adjusted so that the controller output does not change on the next controller cycle. The controller will subsequently integrate the output in a ramp–like fashion until the PV reaches the new SP.

## Output Parameters

The controller output is set at Y. The range of the output is defined by the parameters YMIN and YMAX. When these limits are reached, the parameters QMIN and QMAX are set. In the event that an output limit is reached, anti–reset–windup is activated. This helps to avoid the I–contribution from continually integrating, so that when the controller inputs create a change of direction of the output, the output Y will be immediately released from the upper or lower limit.

# SEQ_VAC: Scaling/Sequence Block for Air Conditioning

## What's in This Chapter



# Introduction

This chapter describes the SEQ_VAC function block.

# Brief Description

## Function Description

SEQ_VAC is a basic function block that can be used to scale real input variables to real output variables using a linear relationship. The block can be used to scale both process variable inputs as well as controller outputs.

The scaling is performed in the form of cartesian coordinates. The input variable range is specified as X1...X2. The output variable range is specified as Y1...Y2.

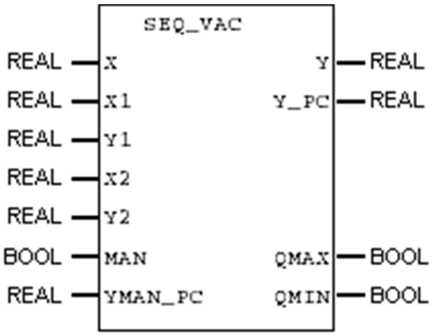
The EFB provides the following:

- 4 quadrant operation, allowing the creation of both positive, negative, forward and reverse acting sequences
- limitation of the outputs Y1...Y2 when the input range X1...X2 is exceeded

  When this happens, the variables QMIN and QMAX are set.
- manual operation mode to allow the presetting of the output in percentage form
- display of the output in both REAL and percentage form.

Additional parameters EN and ENO can be configured.

## Representation




## Parameter Description

### Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| X | REAL | input variable |
| X1 | REAL | 1. abscissa value, 1. value pair |
| Y1 | REAL | 1. ordinate value, 1. value pair |
| X2 | REAL | 1. abscissa value, 2. value pair |
| Y2 | REAL | 1. ordinate value, 2. value pair |
| MAN | BOOL | manual mode |
| YMAN_PC | REAL | manual control value 0 – 100% |

### Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y | REAL | output variable (control value) |
| Y_PC | REAL | output variable as a % |
| QMAX | BOOL | upper limit of signalling device reached |
| QMIN | BOOL | lower limit of signalling device reached |

# Detailed Description

## Basic Operation

The SEQ_VAC block is processed in each active cycle. Examples of the use of SEQ_VAC can be found under General Instructions, page 25.

## Parameter Specifications

The input variable to be scaled is denoted as X. The range over which it is to be scaled is specified by X1 and X2. X1 has to be less than X2. The corresponding output range is specified by the parameters Y1 and Y2.

If the input range X1 <= X <= X2 is exceeded, the output variable is clamped to the limits represented by the values Y1 and Y2 such that:

$Y_{(X>X2)}$ = Y2 and

$Y_{(X<X1)}$ = Y1.

If the input range is exceeded and clamping is activated, this is indicated by the setting of the variables QMAX and QMIN.

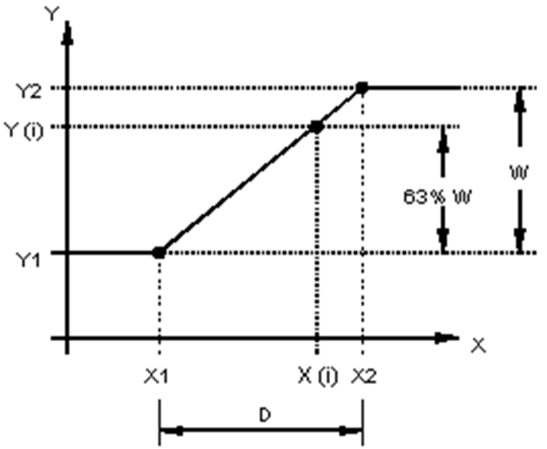
An increasing sequence is specified by setting Y1 < Y2.

A decreasing (reverse acting) sequence is specified by specifying Y1>Y2.

The percentage value of the scaling result is output at Y_PC.

**NOTE:** The smaller value of Y1 and Y2 is equated to 0% and the greater value to 100%. In other words, the percentage value is related to the size of the output variable Y independent of the direction (increasing/decreasing) of the sequence.

Ascending sequence, 1. quadrant




**W** Output value range

**D** Input definition range

**X(i), Y(i)** Current values

**X** 2173.0

**X1** 1000.0

**Y1** 400.0

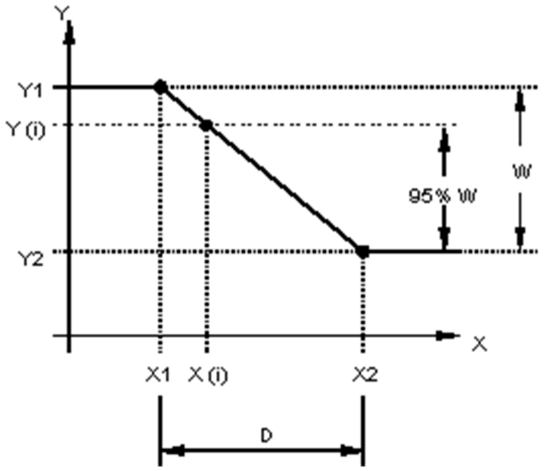
**X2** 3000.0

**Y2** 8000.0

**Y** 4857.4

**Y_PC** 58.65

Descending sequence, 1. quadrant




**W** Output value range

**D** Input definition range

**X(i), Y(i)** Current values

**X** 117.0

**X1** 100.0

**Y1** 7500.0

**X2** 450.0

**Y2** 1250.0

**Y** 7196.428

**Y_PC** 95.143

## Manual Operation

The SEQ_VAC block may be put in manual by setting MAN = 1.

In manual mode, the percentage value specified by YMAN_PC is written to the block's output, i.e., Y_PC = YMAN_PC. The real value Y is determined by the scaling of the sequence.

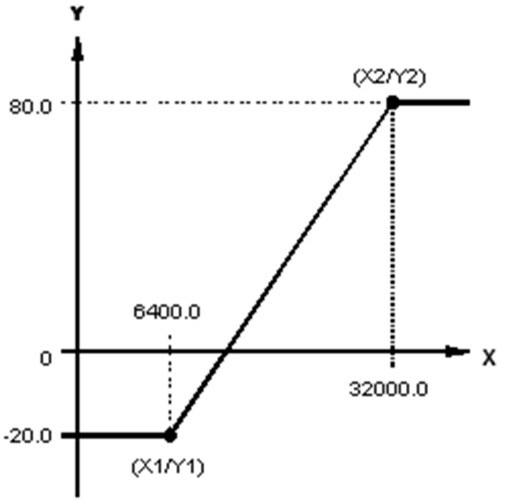
## Example

Convert an analog input 4-20 mA to a REAL value corresponding to a temperature range of –20 to +80 degrees Celsius.

The corresponding parameters are:

- X1 = 6400.0 (accordingly –20 degrees C)
- X2 = 32000.0 (accordingly +80 degrees C)
- Y1 = –20.0 (accordingly –20 degrees C)
- Y2 = 80.0 (accordingly +80 degrees C)

Sequence 4. –1. quadrant




**X1** 6400.0

**Y1** - 20.0

**X2** 32000.0

**Y2** 80.0

**X** (Analog value LZ. standard signal)

# SW_VAC: Summer/Winter Setpoint Compensation for Air Conditioning

## What's in This Chapter



# Introduction

This chapter describes the SW_VAC function block.

# Brief Description

## Function Description

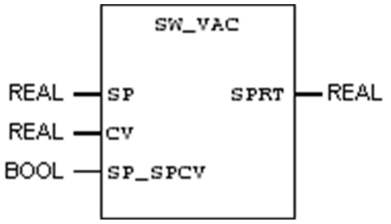
SW_VAC is a basic function block that provides summer/winter compensation of a temperature setpoint based on the outside air temperature command variable CV.

The EFB provides the following:

- summer/winter setpoint compensation with summer compensation following the DIN 1946 Part 2 standard
- switchable operating modes offering the choice of compensation or no compensation

Additional parameters EN and ENO may be configured.

## Representation



## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP | REAL | setpoint |
| CV | REAL | command variable<br>(as a rule the outside air temperature) |
| SP_SPCV | BOOL | setpoint / setpoint + command variable |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SPRT | REAL | setpoint room temperature |

# Detailed Description

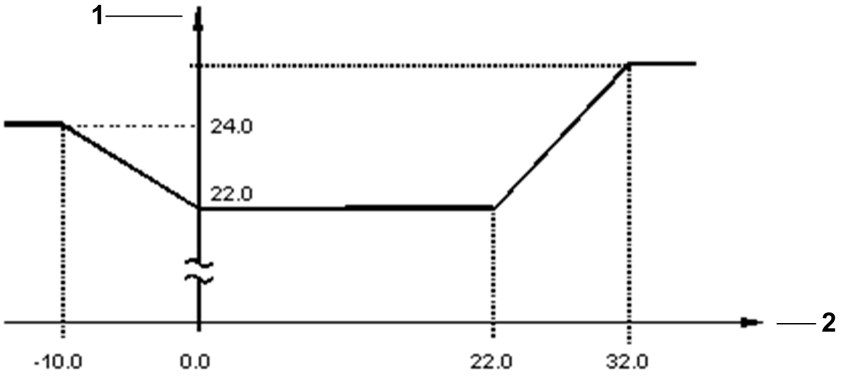
## Basic Operation

The `SW_VAC` function block is processed in each cycle.

The function block is used to adjust the room or inlet air temperature setpoint SPRT based on the outside air temperature command variable CV.

The compensation curves are shown in figure below.

Summer/winter compensation curve




**1** SPComp (Degrees C)

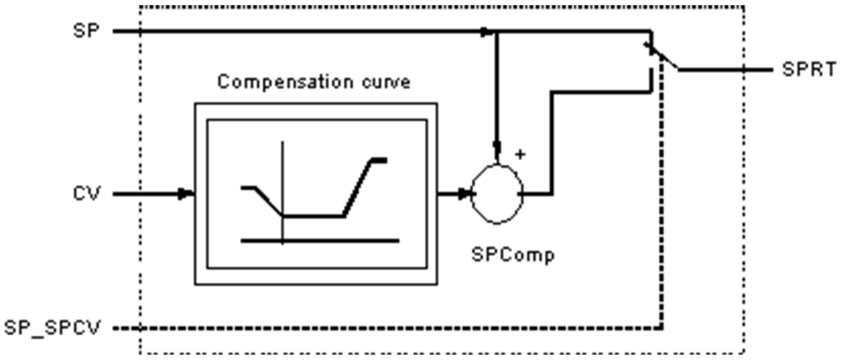
**2** CV (Degrees C)

The SP_SPCV input can be used to switch the block's mode of operation.

In "Setpoint" mode (SP_SPCV = 0), no compensation is performed and SPRT = SP.

In "Setpoint with command variable" mode (SP_SPCV = 1), compensation is performed and SPRT = SP + SPComp as shown in figure below).

Module structure




Under normal circumstances, when operating in compensation mode, SP is set to zero and SPRT = SPComp.

However, the user may choose to input a value for SP. This will have the effect of moving the summer/winter compensation curve shown in figure "Summer/winter compensation curve" vertically up or down.

## Parameter Specifications

The command variable is specified in degrees Celsius.

The setpoint SP is a real value and has to be scaled as appropriate.

The output of the block SPRT is scaled as a real value.

# THRS_VAC: Threshold Switch with Hysteresis for Air Conditioning

## What's in This Chapter



# Introduction

This chapter describes the THRS_VAC function block.

# Brief Description

## Function Description

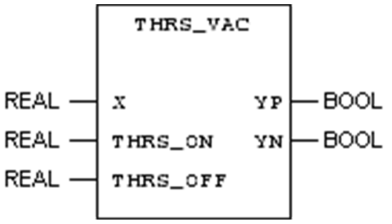
THRS_VAC is a basic function block that is used to detect a threshold on a REAL variable with a built in hysteresis.

The EFB provides the following:

- setting of an on and off threshold on a real input variable
- possibility to set the control direction by selecting any threshold values
- positive and negative output signals

Additional parameters EN and ENO may be configured.

## Representation



## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| X | REAL | input variable |
| THRS_ON | REAL | ‘energize’ limit value |
| THRS_OFF | REAL | ‘switch off’ limit value |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| YP | BOOL | positive reply signal |
| YN | BOOL | negative reply signal |

# Detailed Description

## Basic Operation

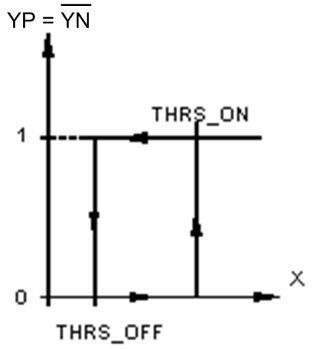
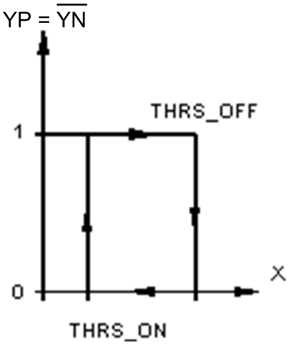
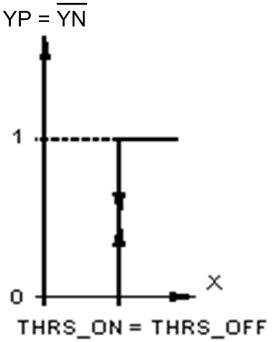
The block monitors an input variable X for 2 limits/thresholds.

When the on–threshold THRS_ON is reached, the output YP is set equal to 1, and when the off–threshold THRS_OFF is reached, the output YP is set to 0.

The output YN is set as the complement of YP.

The user is free to set any values for the thresholds. The behavior of the outputs based on the relative sizes of the 2 thresholds is shown in figures below.

**Hysteresis and switching function of THRS_VAC with different configurations**

| THRS_ON > THRS_OFF | THRS_ON < THRS_OFF | THRS_ON = THRS_OFF |
|---|---|---|
|  |  |  |







In the case where THRS_ON is not equal to THRS_OFF, when the variable X lies between the 2 thresholds, the output YP remains in its current state until the opposite threshold is reached. In this way, a hysteresis function is provided (see figure “THRS_ON > THRS_ OFF“). When THRS_ON = THRS_OFF, the block acts as a comparator switch.

The `THRS_VAC` function block is processed in each cycle. During the first cycle, YP = 0 and YN = 1.

The `THRS_VAC` block may be used as a switch in many applications such as summer/winter compensation, day/night temperature drops, anti–freeze device, etc.

# UC2_VAC: Universal PI Controller for Air Conditioning with 2 Outputs

## What's in This Chapter



# Introduction

This chapter describes the UC2_VAC function block.

# Brief Description

## Function Description

The UC2_VAC complex function block is a general PI–controller designed for air conditioning applications.

It can be used for temperature control, humidity control, air mixing control or other general functions.

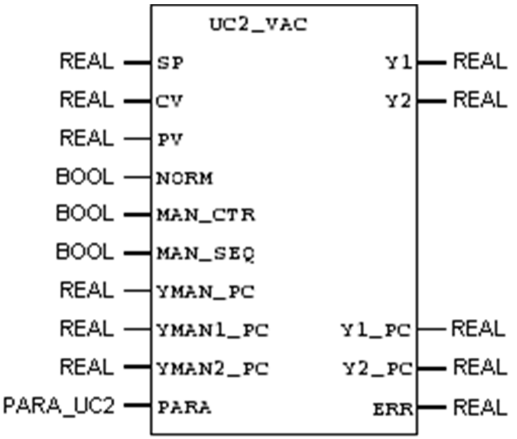
It consists of a PI controller, summer/winter setpoint compensation and 2 output sequences.

The EFB provides the following:

- winter/summer setpoint compensation as per DIN 1946 part 2
- 4 quadrant operation of the output sequence scaling
- display of output variables as percentages
- upper and lower limits on outputs
- presetting the controllers' gains in the form of GAIN or PROP with the possibility of using negative values for switching the control direction
- operation with Anti–Windup–Reset (AWR)
- manual adjustment of either the PI–controller total output or the individual sequence outputs (Y1 and Y2) using percentages

  When the controller output is manually adjusted, the EFB tracks the I contribution in order to provide bumpless switching back to automatic mode.
- display of the P and PI controller deviations (SP–PV)

## Representation




## Parameter Description

### Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP | REAL | setpoint |
| CV | REAL | command variable |
| PV | REAL | actual value |
| NORM | BOOL | basic setting |
| MAN_CTR | BOOL | manual for controller |
| MAN_SEQ | BOOL | manual for sequences |
| YMAN_PC | REAL | total manual manipulated variable as a % for controller |
| YMAN1_PC | REAL | individual manual manipulated variable as a % for 1 sequence |
| YMAN2_PC | REAL | individual manual manipulated variable as a % for 2. sequence |
| PARA | PARA_UC2 | parameter structure |

### Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y1 | REAL | manipulated variable / output variable 1 |
| Y2 | REAL | manipulated variable / output variable 2 |

| Parameter | Data Type | Meaning |
|---|---|---|
| Y1_PC | REAL | manipulated variable / output variable 1 as a % |
| Y2_PC | REAL | manipulated variable / output variable 2 as a % |
| ERR | REAL | system deviation |

## Type Description

PARA_UC2

| Parameter | Data Type | Meaning |
|---|---|---|
| SP_SPCV | BOOL | setpoint / setpoint + command variable |
| **PI Controller** | | |
| YMAX | REAL | upper limit manipulated variable |
| YMIN | REAL | lower limit manipulated variable |
| GAIN | REAL | controller gain |
| PROP | REAL | proportional value |
| PREF | REAL | proportional value reference |
| TI | TIME | reset time |
| **1. Sequence** | | |
| X1_1 | REAL | 1. abscissa value |
| Y1_1 | REAL | 1. ordinate value |
| X2_1 | REAL | 2. abscissa value |
| Y2_1 | REAL | 2. ordinate value |
| **2. Sequence** | | |
| X1_2 | REAL | 1. abscissa value |
| Y1_2 | REAL | 1. ordinate value |
| X2_2 | REAL | 2. abscissa value |
| Y2_2 | REAL | 2. ordinate value |

# Detailed Description

## EFB Structure

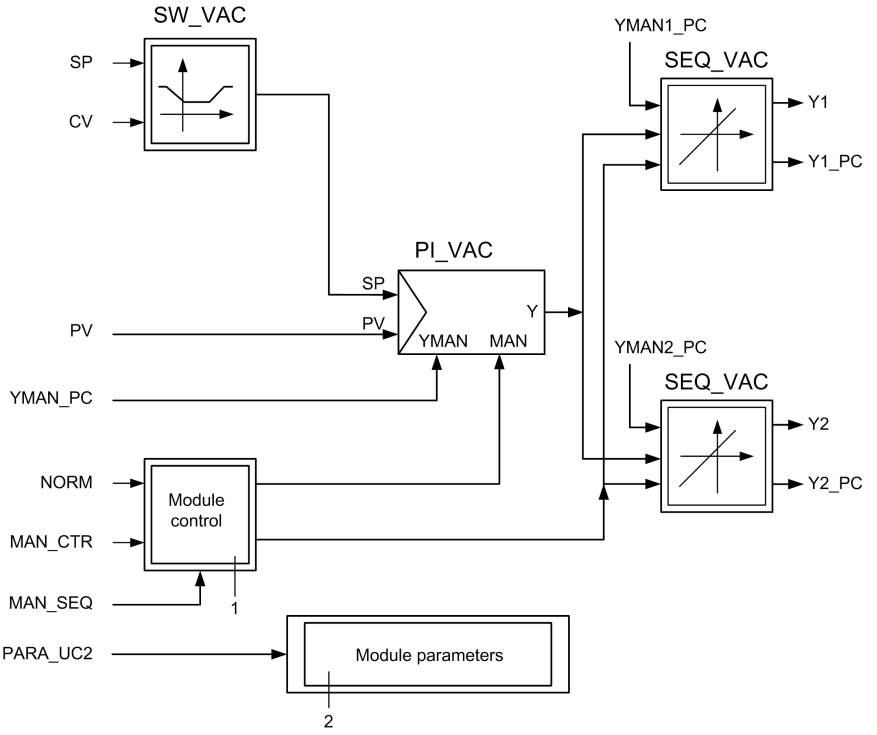
A block diagram representation of the UC2_VAC complex function block is shown in figure below.

It is made up of the following basic function blocks:

- SW_VAC, page 73
- PI_VAC, page 62
- SEQ_VAC, page 67

Please refer to the respective individual basic function block description for detailed information.

Controller structure




**1** Module control

**2** Module parameters

## Basic Operation

The setpoint for the UC2_VAC function block is fed to the PI–controller via the summer/winter setpoint compensation block SW_VAC, page 73.

The output of the SW_VAC block is fed to the setpoint input of the PI controller.

The PI–controller output is fed to 2 sequence output blocks SEQ_VAC. The PI–controller output can be maintained within minimum and maximum limits by using the PI–controller output limits YMAX1 and YMIN1.

## Manual Operation

There are 3 possible modes of manual operation which are specified by setting the mutually exclusive NORM, MAN_CTR and MAN_SEQ parameters.

- **NORM mode (NORM = 1):**

  The output of the PI–controller output is set to zero. The zero value is fed to the output sequences that are active in NORM mode. As a result the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters. In the case where the controller output limits YMIN2 and YMAX2 have been set to values that do not enclose zero, the output will be set to the value of YMIN2 and YMAX2 that is closest to zero. In other words:

  Y = YMAX2 if YMAX2 < 0 AND YMIN2 < 0,

  Y = YMIN2 if YMAX2 > 0 AND YMIN2 > 0.

  Again, the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters.

- **MAN_CTR mode (MAN_CTR=1):**

  The output of the PI controller is set equal to the user–specified parameter YMAN_PC. Again, the output sequences are active in this mode and therefore the actual values of Y1 and Y2 that are output to the control actuators will depend on the output sequence parameters.

- **MAN_SEQ mode (MAN_SEQ=1):**

  The outputs Y1 and Y2 are set to the values specified by the parameters YMAN1_PC and YMAN2_PC:

  i.e., Y1_PC = YMAN1_PC and Y2_PC = YMAN2_PC.

In MAN_CTR mode, the I contribution of the PI–controller is tracked so that a bumpless transfer back to automatic mode may be carried out. In NORM and MAN-SEQ modes, the I contribution is set to zero.

## Output Parameters

The UC2_VAC outputs Y1 and Y2 are available in real or percentage form (Y1, Y2, Y1_PC and Y2_PC).

The percentage values specified for the manual control of outputs (YMAN_PC, YMAN1_PC and YMAN2_PC) refer to the specified range of the appropriate output. For example, the variable YMAN_PC sets the total output of the PI_VAC basic function block. The range of this output is specified by the parameters YMIN (0%) and YMAX (100%).

The variables YMAN1_PC and YMAN2_PC set the outputs of the 2 SEQ_VAC blocks, the ranges of which are specified by their corresponding ordinate vales Y1..Y2 (where Y1 and Y2 refer to the SEQ_VAC ordinates, not the actual outputs Y1 and Y2 of the 2 SEQ_VAC blocks).

**NOTE:** The smaller value of Y1 and Y2 is equated to 0% and the greater value to 100%. In other words, the percentage value is related to the size of the output variable Y irrespective of its direction:

- Y1 < Y2 –> 0%..100% = Y1..Y2
- Y1 > Y2 –> 0%..100% = Y2..Y1

For more information on the operation of the SEQ_VAC block, please refer to the respective description, page 67.

# UC3_VAC: Universal PI Controller for Air Conditioning with 3 Outputs

## What's in This Chapter



# Introduction

This chapter describes the `UC3_VAC` function block.

# Brief Description

## Function Description

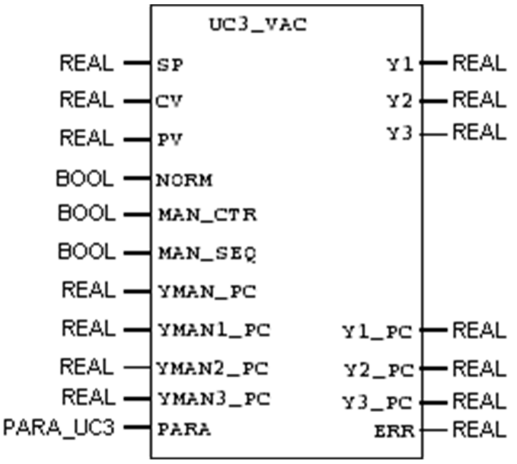
The `UC3_VAC` complex function block is a general PI–controller designed for air conditioning applications. It can be used for temperature control, humidity control, air mixing control or other general functions. It consists of a PI controller, summer/winter setpoint compensation and 3 output sequences.

The EFB provides the following:

- winter/summer setpoint compensation as per DIN 1946 part 2
- 4 quadrant operation of the output sequence scaling
- display of output variables as percentages
- upper and lower limits on outputs
- presetting the controllers' gains in the form of GAIN or PROP with the possibility of using negative values for switching the control direction
- operation with Anti–Windup–Reset (AWR)
- manual adjustment of either the PI–controller total output or the individual sequence outputs (Y1 and Y2) using percentages

  When the controller output is manually adjusted, the EFB tracks the I contribution in order to provide bumpless switching back to automatic mode.
- display of the P and PI controller deviations (SP–PV)

## Representation




## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP | REAL | setpoint |
| CV | REAL | command variable |
| PV | REAL | actual value |
| NORM | BOOL | basic setting |
| MAN_CTR | BOOL | manual for controller |
| MAN_SEQ | BOOL | manual for sequences |
| YMAN_PC | REAL | total manual manipulated variable as a % for controller |
| YMAN1_PC | REAL | individual manual manipulated variable as a % for 1 sequence |
| YMAN2_PC | REAL | individual manual manipulated variable as a % for 2 sequence |
| YMAN3_PC | REAL | individual manual manipulated variable as a % for 3 sequence |
| PARA | PARA_UC3 | parameter structure |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y1 | REAL | manipulated variable / output variable 1 |
| Y2 | REAL | manipulated variable / output variable 2 |
| Y3 | REAL | manipulated variable / output variable 3 |
| Y1_PC | REAL | manipulated variable / output variable 1 as a % |
| Y2_PC | REAL | manipulated variable / output variable 2 as a % |
| Y3_PC | REAL | manipulated variable / output variable 3 as a % |
| ERR | REAL | system deviation |

## Type Description

PARA_UC3

| Parameter | Data Type | Meaning |
|---|---|---|
| SP_SPCV | BOOL | setpoint / setpoint + command variable |
| **PI Controller** | | |
| YMAX | REAL | upper limit manipulated variable |
| YMIN | REAL | lower limit manipulated variable |
| GAIN | REAL | controller gain |
| PROP | REAL | proportional value |
| PREF | REAL | proportional value reference |
| TI | TIME | reset time |
| **1. Sequence** | | |
| X1_1 | REAL | 1. abscissa value |
| Y1_1 | REAL | 1. ordinate value |
| X2_1 | REAL | 2. abscissa value |
| Y2_1 | REAL | 2. ordinate value |
| **2. Sequence** | | |
| X1_2 | REAL | 1. abscissa value |
| Y1_2 | REAL | 1. ordinate value |
| X2_2 | REAL | 2. abscissa value |
| Y2_2 | REAL | 2. ordinate value |

| Parameter | Data Type | Meaning |
|---|---|---|
| **3. Sequence** | | |
| X1_3 | REAL | 1. abscissa value |
| Y1_3 | REAL | 1. ordinate value |
| X2_3 | REAL | 2. abscissa value |
| Y2_3 | REAL | 2. ordinate value |

# Detailed Description

## EFB Structure

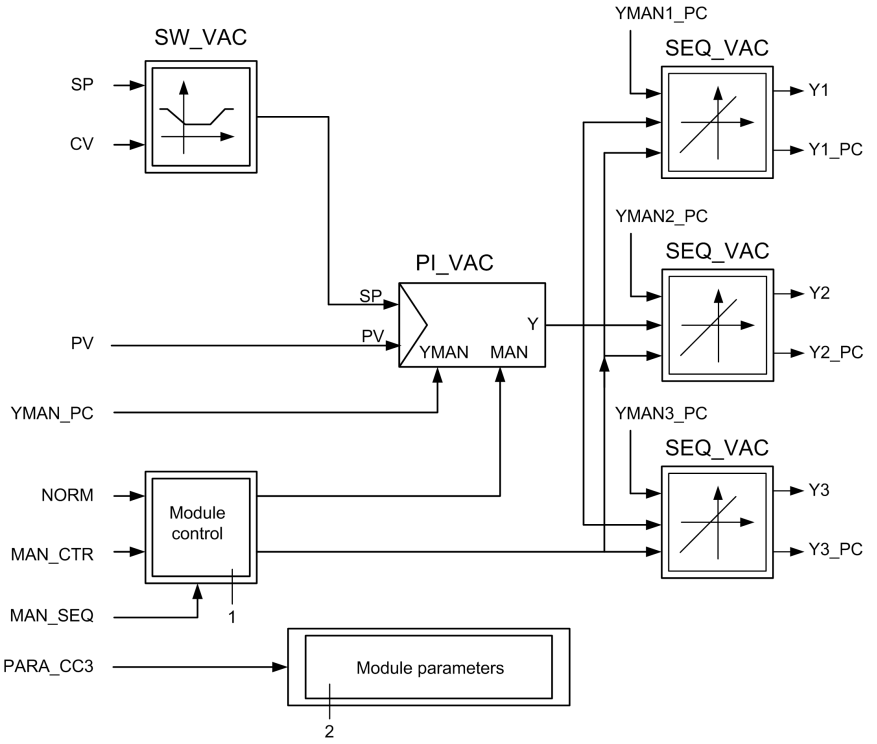
A block diagram representation of the UC3_VAC complex function block is shown in figure below.

It is made up of the following basic function blocks:

- SW_VAC, page 73
- PI_VAC, page 62
- SEQ_VAC, page 67

Please refer to the respective individual basic function block description for detailed information.

Controller structure




**1** Module control

**2** Module parameters

## Basic Operation

The setpoint for the UC3_VAC function block is fed to the PI–controller via the summer/winter setpoint compensation block SW_VAC, page 73. The output of the SW_VAC block is fed to the setpoint input of the PI controller.

The PI–controller output is fed to 3 sequence output blocks SEQ_VAC. The PI-controller output can be maintained within minimum and maximum limits by using the PI–controller output limits YMAX1 and YMIN1.

## Manual Operation

There are 3 possible modes of manual operation which are specified by setting the mutually exclusive NORM, MAN_CTR and MAN_SEQ parameters.

- **NORM mode (NORM = 1):**

  The output of the PI–controller output is set to zero. The zero value is fed to the output sequences that are active in NORM mode. As a result the actual values of Y1, Y2 and Y3 that are output to the control actuators will depend on the output sequence parameters. In the case where the controller output limits YMIN2 and YMAX2 have been set to values that do not enclose zero, the output will be set to the value of YMIN2 and YMAX2 that is closest to zero. In other words:

  Y = YMAX2 if YMAX2 < 0 AND YMIN2 < 0,

  Y = YMIN2 if YMAX2 > 0 AND YMIN2 > 0.

  Again, the actual values of Y1, Y2 and Y3 that are output to the control actuators will depend on the output sequence parameters.

- **MAN_CTR mode (MAN_CTR=1):**

  The output of the PI controller is set equal to the user–specified parameter YMAN_PC. Again, the output sequences are active in this mode and therefore the actual values of Y1, Y2 and Y3 that are output to the control actuators will depend on the output sequence parameters.

- MAN_SEQ mode (MAN_SEQ=1):

  The outputs Y1, Y2 and Y3 are set to the values specified by the parameters YMAN1_PC, YMAN2_PC and YMAN3_PC,

  i.e., Y1_PC = YMAN1_PC, Y2_PC = YMAN2_PC and Y3_PC = YMAN3_PC.

In MAN_CTR mode, the I contribution of the PI–controller is tracked so that a bumpless transfer back to automatic mode may be carried out. In NORM and MAN-SEQ modes, the I contribution is set to zero.

## Output Parameters

The UC3_VAC outputs Y1, Y2 and Y3 are available in real or percentage form (Y1, Y2, Y3, Y1_PC, Y2_PC and Y3_PC).

The percentage values specified for the manual control of outputs (YMAN_PC, YMAN1_PC, YMAN2_PC and YMAN3_PC) refer to the specified range of the appropriate output. For example, the variable YMAN_PC sets the total output of the PI_VAC basic function block. The range of this output is specified by the parameters YMIN (0%) and YMAX (100%).

The variables YMAN1_PC, YMAN2_PC and YMAN3_PC set the outputs of the 2 SEQ_VAC blocks, the ranges of which are specified by their corresponding ordinate vales Y1..Y2 (where Y1 and Y2 refer to the SEQ_VAC ordinates, not the actual outputs Y1, Y2 and Y3 of the 2 SEQ_VAC blocks).

**NOTE:** The smaller value of Y1 and Y2 is equated to 0% and the greater value to 100%. In other words, the percentage value is related to the size of the output variable Y irrespective of its direction:

- Y1 < Y2 –> 0%..100% = Y1..Y2
- Y1 > Y2 –> 0%..100% = Y2..Y1

For more information on the operation of the `SEQ_VAC` block, please refer to the respective description, page 67.

# VQ_VAC: Measured Value Deadband Block for Air Conditioning

## What's in This Chapter



# Introduction

This chapter describes the VQ_VAC function block.

# Brief Description

## Function Description

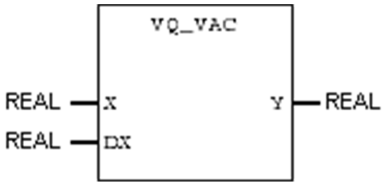
The VQ_VAC basic function block performs the function of adding a deadband DX to an input variable X. If X changes by more than the amount DX, the output Y is updated with the value of X.

The function block may be used to:

- stabilize a process variable that has a certain amount of signal noise, e.g., room pressure measurement
- set a preset minimum modification that has to occur for a value before it will be changed
- provide a dynamic matching of the range of a minimum modification around a current input variable

Additional parameters EN and ENO may be configured.

## Representation



## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| X | REAL | input variable |
| DX | REAL | minimum modification input |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| Y | REAL | output variable |

# Detailed Description

## Basic Operation

The `VQ_VAC` basic function block updates an output Y, with an input value X when X changes by an amount greater than the value set at the input DX.

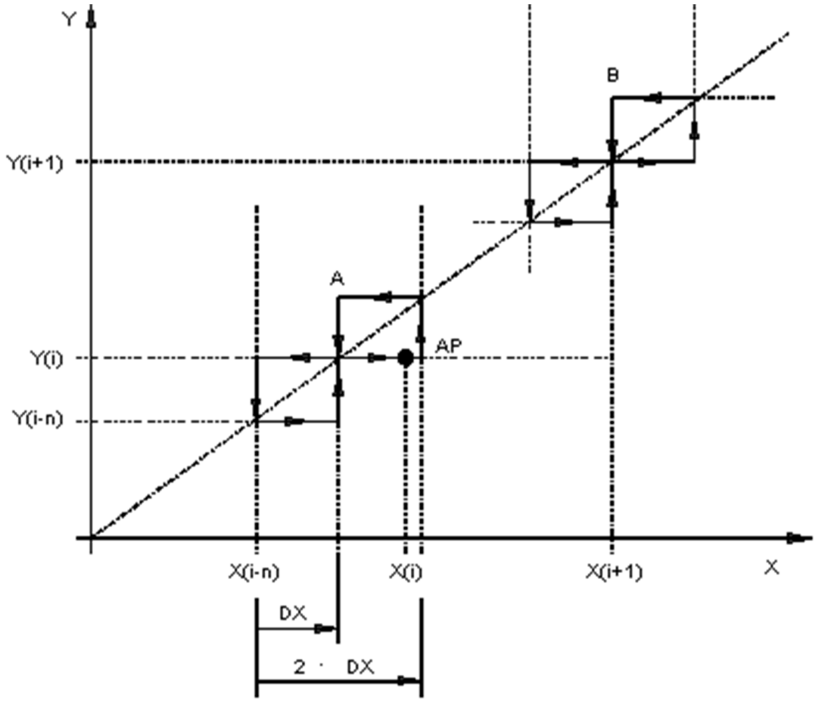
During the first cycle, the output Y is set to equal the value of input X. Y will then retain this value until:

$X_{new} <= X – DX$ or $X_{new} >= X + DX$

At which point, $Y = X_{new}$

This is shown in figure below.

Quantization diagram VQ_VAC




**A** Quantization diagram in the case of 'slow' movement of the working point AP from an input value X(i–n) lying behind n steps of the current input value X(i)

**B** New quantization diagram in the case of stepped modification from X(i) to X(i+1), if the jump is >= DX.

The function block can be used to either stabilize a varying input or to set a dynamic deadband range around a value. The range is dynamic in the sense that it is not absolute, but is relative to the input value X. One example of where it could be used is to set a trigger point relative to a controller setpoint which may itself be changed by an operator.

**NOTE:** Where the block is used to manipulate a process variable to a controller, care has to be taken by performing on site tests that the stability of the control loop is not adversely affected by the introduced deadband.

## ⚠CAUTION

**UNEXPECTED BEHAVIOR OF EQUIPMENT**

Test your application and the stability of your control loop carefully.

**Failure to follow these instructions can result in injury or equipment damage.**

## Parameters

The input X and output Y are real variables. The variable DX has to be set as a positive real value. The scaling basic function block `SEQ_VAC` may be used before or after `VQ_VAC` in order to scale the measured value X/Y.

# WASH_VAC: Basic Washer Block for Air Conditioning

## What's in This Chapter



# Introduction

This chapter describes the WASH_VAC function block.

# Brief Description

## Function Description

The WASH_VAC basic function block provides the calculation of a dew point value SP_TW based upon a dry bulb temperature SP_T and relative humidity value SP_RH.

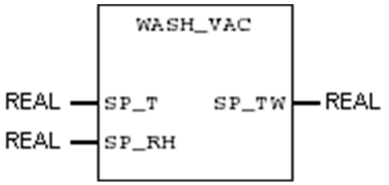
One possible application for the WASH_VAC block is the possibility to control a room's humidity by regulating the outlet temperature of a washer.

| ***NOTICE*** |
| --- |
| **UNEXPECTED BEHAVIOR OF EQUIPMENT**<br>Use this block only with air washer providing 100% humidity saturated air.<br>**Failure to follow these instructions can result in equipment damage.** |

In other words, the block assumes that the outlet washer dry bulb temperature is equivalent to the dew point temperature. Such an approach has the advantage of controlling a humidity using a cost–effective temperature sensor rather than a more expensive humidity sensor.

Additional parameters EN and ENO may be configured.

## Representation




## Parameter Description

Inputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP_T | REAL | setpoint temperature at target location (room temperature) |
| SP_RH | REAL | nominal value relative humidity at target location (room humidity) |

Outputs

| Parameter | Data Type | Meaning |
|---|---|---|
| SP_TW | REAL | setpoint washer outlet temperature |

# Detailed Description

## Basic Operation

The principle of operation of the WASH_VAC basic function block relies on the fact that the dewpoint temperature of air is equal to the dry bulb temperature when the air is totally saturated, i.e., its relative humidity is equal to 100%.

The purpose of the WASH_VAC basic function block is to calculate the dewpoint temperature setpoint SP_TW, based on a specified room temperature setpoint SP_T and relative humidity setpoint SP_RH.

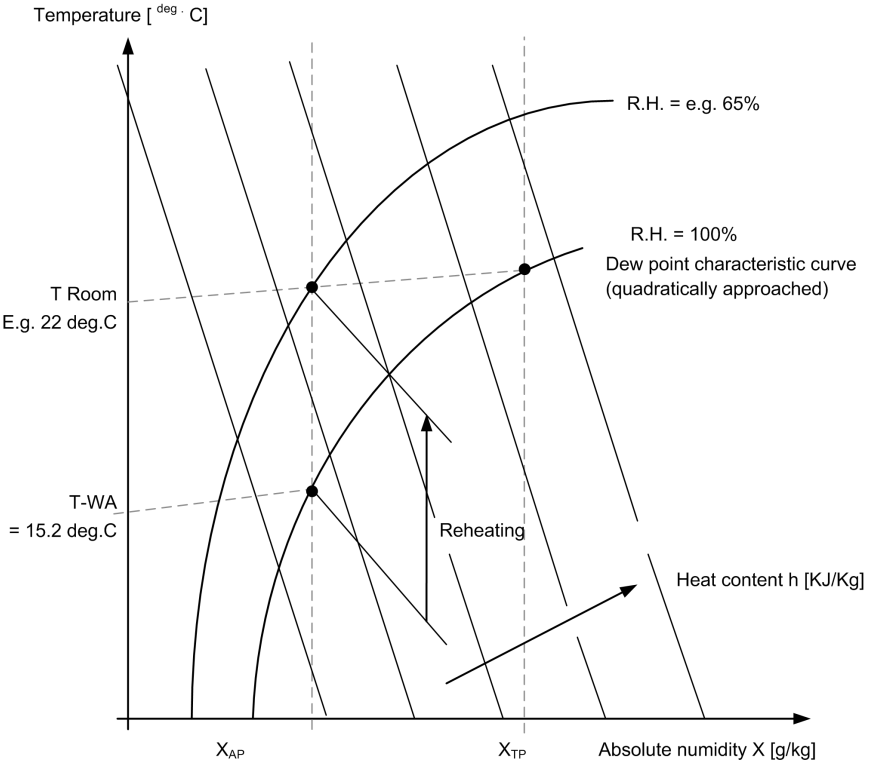
For a washer application, by controlling the washer outlet temperature to SP_TW, if the air is reheated to the room temperature setpoint SP_T, then the room relative humidity value will equal SP_RH.

This can be done because reheating the air does not change the air's absolute humidity, but only its dry bulb temperature and therefore its relative humidity.

It therefore follows that if the washer outlet air has a relative humidity of 100%, the dewpoint temperature can be measured and controlled using a simple dry bulb temperature sensor rather than a more expensive humidity sensor.

An example is shown on the figure below. For a room temperature of 22 degrees Celsius and a relative humidity of 65%, the `WASH_VAC` function block will calculate the dewpoint setpoint as 15.2 degrees Celsius. By controlling the washer outlet temperature to 15.2 degrees Celsius and subsequently reheating the air to 22 degrees Celsius, the room relative humidity will equal 65%.

**h/x diagram with clarification of the method of operation of the WASH_VAC**




## Parameters

The air temperature dry bulb temperature setpoint is specified by the variable SP_T in degrees Celsius. The relative humidity is specified by the variable SP_RH as a % value.

The washer outlet temperature setpoint is output in degrees Celsius at the location specified at SP_TW.

## Example

An example application consisting of a preheat coil, cooling coil, washer and reheat coil is shown in the figure below.

The setpoint calculated by WASH_VAC is fed to a lower level PI controller whose PV is connected to the washer temperature transmitter and whose output is used to control the preheat and cooling coils.

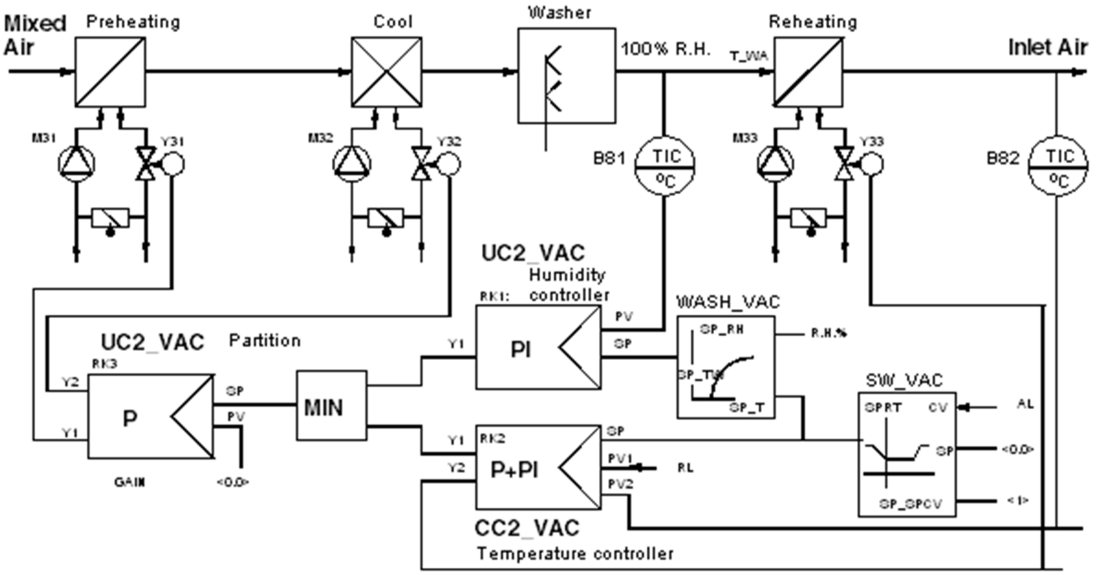
In other words, this PI controller maintains the appropriate dewpoint temperature off the washer.

In addition, a temperature controller CC2_VAC is used to control the room and inlet air temperature by driving the preheat/reheat coils and cooling coil.

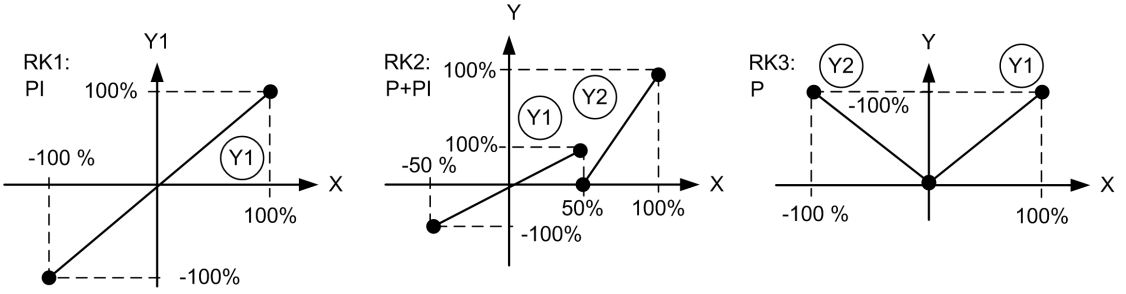
A minimum select is used to switch between the humidity and temperature controller. The controller RK3 is configured as a P controller only with a PV set to zero and the gain set to 1. In this way, it simply acts as an output sequencer that divides the setpoint input between the preheat and cooling coils.

The washer outlet temperature is set to a range of 5 to 30 degrees Celsius to avoid icing up or overheating of the washer.

**Integration of WASH_VAC in a room / inlet air cascade control system**



## Sequences


RK1:
PI
Y1
100%
-100 %
Y1
X
100%
-100%
RK2:
P+PI
Y
100%
Y1
Y2
100%
-50 %
X
50%
100%
-100%
RK3:
P
Y2
Y
Y1
-100%
X
-100 %
100%

# Glossary

## A

**ADDM_TYPE:**

This predefined type is used as an output for the ADDM function. This is an ARRAY[0..8] OF Int. You can find it in the library, in the same family as the EFs that use it.

**ADDR_TYPE:**

This predefined type is used as an output for the ADDR function. This is an ARRAY[0..5] OF Int. You can find it in the library, in the same family as the EFs that use it.

**ANL_IN:**

`ANL_IN` is the abbreviation of the analog input data type. It is used when processing analog values. `%IW` addresses in the configured analog input module, which are specified in the list of I/O components, are automatically assigned to data types, and therefore must be occupied by unassigned variables only.

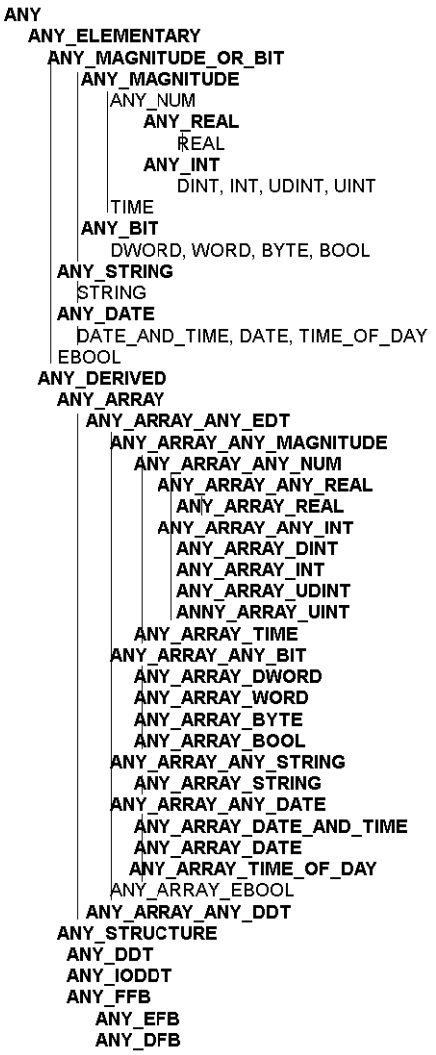
**ANL_OUT:**

`ANL_OUT` is the abbreviation of the analog output data type. It is used when processing analog values. `%MW` addresses in the configured analog input module, which are specified in the list of I/O components, are automatically assigned to data types, and therefore must be occupied by unassigned variables only.

**ANY:**

There is a hierarchy among the various data types. In the DFBs, it is sometimes possible to declare variables that can contain several types of values. In that case we use `ANY_xxx` types.

The figure below describes this hierarchical structure:




**ARRAY:**

An `ARRAY` is a table containing elements of a single type.

The syntax is as follows: `ARRAY [<limits>] OF <Type>`

Example:

`ARRAY [1..2] OF BOOL` is a one-dimensional table with two elements of type `BOOL`.

`ARRAY [1..10, 1..20] OF INT` is a two-dimensional table with 10x20 elements of type `INT`.

**Assigned variables:**

A variable whose position in the PLC memory can be known. For example, the `Water_pressure` variable is associated with `%MW102`. `Water_pressure` is said to be assigned.

## B

**BCD:**

BCD is the abbreviation of the Binary Coded Decimal format.

BCD can be used to represent decimal numbers between 0 and 9 using a set of four bits (nybble).

In this format, the four bits used to encode decimal numbers have an unused range of combinations.

Example of BCD encoding:

- The number `2,450`
- is encoded: `0010 0100 0101 0000`

**BOOL:**

`BOOL` is the abbreviation for the Boolean type. This is the basic data type in computing. A `BOOL` variable can have either of the following two values: 0 (`FALSE`) or 1 (`TRUE`).

A bit extracted from a word is of type `BOOL`, for example: `%MW10.4`.

**BYTE:**

When 8 bits are grouped together, they are called a `BYTE`. You can enter a `BYTE` either in binary mode or in base 8.

The `BYTE` type is encoded in an 8 bit format which, in hexadecimal format, ranges from `16#00` to `16#FF`.

## D

**DATE_AND_TIME:**

See `DT`.

**DATE:**

The `DATE` type, encoded in BCD in a 32 bit format, contains the following information:

- the year encoded in a 16 bit field;

- the month encoded in an 8 bit field;
- the day encoded in an 8 bit field.

The `DATE` type must be entered as follows: **`D#`** `<Year>` **-** `<Month>` **-** `<Day>`

This table shows the upper/lower limits of each field:

| Field | Limits | Comment |
|---|---|---|
| Year | [1990,2099] | Year |
| Month | [01,12] | The leading 0 is displayed; it can be omitted during data entry. |
| Day | [01,31] | For months 01/03/05/07/08/10/12 |
| | [01,30] | For months 04/06/09/11 |
| | [01,29] | For month 02 (leap years) |
| | [01,28] | For month 02 (non-leap years) |

**DBCD:**

Representation of a double integer in double BCD format.

BCD format can be used to represent decimal numbers between 0 and 9 using a set of four bits.

In this format, the four bits used to encode decimal numbers have an unused range of combinations.

Example of DBCD encoding:

- The number `78,993,016`
- is encoded: `0111 1000 1001 1001 0011 0000 0001 0110`

**DDT:**

DDT is the abbreviation of Derived Data Type.

A derived data type is a set of elements with the same type (`ARRAY`) or with different types (structure).

**DFB:**

DFB is the abbreviation of Derived Function Block.

DFB types are function blocks that can be defined by the user in ST, IL, LD or FBD language.

Using these DFB types in an application makes it possible to:

- simplify the design and entry of the program;
- make the program easier to read;

- make it easier to debug;
- reduce the amount of code generated.

**DINT:**

DINT is the abbreviation of Double INTeger (encoded in 32 bits).

The upper/lower limits are as follows: -(2 to the power of 31) to (2 to the power of 31) - 1.

Example:

-2147483648, 2147483647, 16#FFFFFFFF.

**DT:**

DT is the abbreviation of Date and Time.

The DT type, encoded in BCD in a 64 bit format, contains the following information:

- the year encoded in a 16 bit field;
- the month encoded in an 8 bit field;
- the day encoded in an 8 bit field;
- the time encoded in an 8 bit field;
- the minutes encoded in an 8 bit field;
- the seconds encoded in an 8 bit field.

**NOTE:** The 8 least significant bits are not used.

The DT type must be entered as follows:

**DT#** <Year> **-** <Month> **-** <Day> **-** <Hour> **:** <Minutes> **:** <Seconds>

This table shows the upper/lower limits of each field:

| Field | Limits | Comment |
|---|---|---|
| Year | [1990,2099] | Year |
| Month | [01,12] | The leading 0 is displayed; it can be omitted during data entry. |
| Day | [01,31] | For months 01/03/05/07/08/10/12 |
| | [01,30] | For months 04/06/09/11 |
| | [01,29] | For month 02 (leap years) |
| | [01,28] | For month 02 (non-leap years) |
| Hour | [00,23] | The leading 0 is displayed; it can be omitted during data entry. |
| Minute | [00,59] | The leading 0 is displayed; it can be omitted during data entry. |
| Second | [00,59] | The leading 0 is displayed; it can be omitted during data entry. |

**DWORD:**

DWORD is the abbreviation of Double Word.

The DWORD type is encoded in a 32 bit format.

This table shows the upper/lower limits of each of the bases that can be used:

| Base | Lower limit | Upper limit |
|---|---|---|
| Hexadecimal | 16#0 | 16#FFFFFFFF |
| Octal | 8#0 | 8#37777777777 |
| Binary | 2#0 | 2#11111111111111111111111111111111 |

Examples of representation:

| Data | Representation in one of the bases |
|---|---|
| 00000000000010101101110011011110 | 16#ADCDE |
| 00000000000000010000000000000000 | 8#200000 |
| 00000000000010101011110011011110 | 2#10101011110011011110 |

## E

**EBOOL:**

EBOOL is the abbreviation of Extended BOOLean. An EBOOL type has a value (0 (FALSE) or 1 (TRUE), but also rising or falling edges and forcing functions.

An EBOOL variable occupies one byte in memory.

The byte contains the following information:

- one bit for the value;
- one bit for the history (whenever the object changes state, the value is copied to the history bit);
- one bit for forcing (equal to 0 if the object is not forced, or 1 if the bit is forced).

The default value of each bit is 0 (FALSE).

**EFB:**

EFB is the abbreviation of Elementary Function Block.

This is a block used in a program which performs a predefined logical function.

EFBs have states and internal parameters. Even if the inputs are identical, the output values may differ. For example, a counter has an output indicating that the preselection

value has been reached. This output is set to 1 when the current value is equal to the preselection value.

**EF:**

EF is the abbreviation of Elementary Function.

This is a block used in a program which performs a predefined logical function.

A function does not have any information on the internal state. Several calls to the same function using the same input parameters will return the same output values. You will find information on the graphic form of the function call in the "[functional block (instance)]". Unlike a call to a function block, function calls include only an output which is not named and whose name is identical to that of the function. In FBD, each call is indicated by a unique [number ] via the graphic block. This number is managed automatically and cannot be modified.

You position and configure these functions in your program in order to execute your application.

You can also develop other functions using the SDKC development kit.

**Elementary function:**

See EF.

**EN:**

EN stands for **EN**able; it is an optional block input. When the EN input is enabled, an ENO output is set automatically.

If EN = 0, the block is not enabled; its internal program is not executed, and ENO is set to 0.

If EN = 1, the block's internal program is run and ENO is set to 1. If an error occurs, ENO is set to 0.

If the EN input is not connected, it is set automatically to 1.

**ENO:**

ENO stands for **E**rror **NO**tification; this is the output associated with the optional input EN.

If ENO is set to 0 (because EN = 0 or in case of an execution error):

- the status of the function block outputs remains the same as it was during the previous scanning cycle that executed correctly;
- the output(s) of the function, as well as the procedures, are set to "0".

## F

**FBD:**

FBD is the abbreviation of Function Block Diagram.

FBD is a graphical programming language that works like a flowchart. By adding simple logical blocks (`AND`, `OR`, etc.), each function or function block in the program is represented in this graphical format. For each block, the inputs are on the left and the outputs on the right. Block outputs can be linked to inputs of other blocks in order to create complex expressions.

**FFB:**

Collective term for EF (elementary function), EFB (elementary function block) and DFB (derived function block).

**FTP:**

File Transfer Protocol.

**Function Block Diagram:**

See FBD.

**Function:**

See EF.

## G

**Global Data:**

Global Data provides the automatic exchange of data variables for the coordination of PLC applications.

**GRAY:**

The Gray code, or "reflected binary", is used to encode a numerical value developed in a string of binary configurations that may be differentiated by changing the status of a single bit.

For example, this code can be used to avoid the following random event: in pure binary, changing the value 0111 to 1000 may produce a range numbers between 0 and 1,000, given that the bits do not all change value at the same time.

Equivalence between decimal, BCD and Gray:

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 100 |
| Gray | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 110 |

## H

**HTTP:**

Hypertext Transfer Protocol

## I

**I/O scanning:**

An I/O scan continuously polls I/O modules to collect data bits and status, error, and diagnostics information. This process monitors inputs and control outputs.

**%I:**

According to the CEI standard, `%I` indicates a language object of type discrete IN.

**IEC 61131-3:**

International standard: programmable logic controllers

Part 3: programming languages

**IL:**

IL is the abbreviation of Instruction List.

This language is a series of basic instructions.

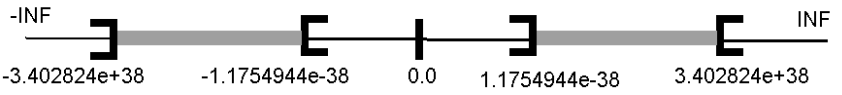
It is very close to assembly language used to program processors.

Each instruction is made up of an instruction code and an operand.

**INF:**

Used to indicate that a number exceeds the authorized limits.

For an integer, the value ranges (shown in gray) are as follows:




When a result is:

- less than -3.402824e+38, the symbol `-INF` (for -infinity) is displayed;
- greater than +3.402824e+38, the symbol `INF` (for +infinity) is displayed;

**INT:**

`INT` is the abbreviation of single INTeger (encoded in 16 bits).

The upper/lower limits are as follows: -(2 to the power of 15) to (2 to the power of 15) - 1.

Example:

`-32768, 32767, 2#1111110001001001, 16#9FA4.`

**IODDT:**

IODDT is the abbreviation of Input/Output Derived Data Type.

The term IODDT indicates a structured data type representing a module or a channel of a PLC module. Each expert module has its own IODDTs.

**%IW:**

According to the CEI standard, `%IW` indicates a language object of type analog IN.

## K

**Keyword:**

A keyword is a unique combination of characters used as a syntax element in a programming language (see the definition provided in appendix B of the IEC 61131-3 standard. All the keywords used in Control Expert and included in the IEC 61131-3 standard appear in appendix C of that standard. Keywords cannot be used as identifiers [names of variables, sections, DFB types, etc.] in your program).

**%KW:**

According to the CEI standard, `%KW` indicates a language object of type constant word.

# L

**LD:**

LD is the abbreviation of Ladder Diagram.

LD is a programming language that represents instructions to be executed as graphical diagrams very similar to electrical diagrams (contacts, coils, etc.).

**Literal value in base 10:**

A literal value in base 10 is used to represent a decimal integer value. This value may be preceded by the "+" and "-" signs. If the "_" character is used in the literal value, it is not significant.

Example:

`-12, 0, 123_456, +986`

**Literal value in base 16:**

A literal value in base 16 is used to represent a hexadecimal integer. The base is determined by the number "16" and the "#" sign. The "+" and "-" signs are prohibited. To make it easier to read, you can use the "_" sign between the bits.

Example:

`16#F_F` or `16#FF` (decimal 255)

`16#E_0` or `16#E0` (decimal 224)

**Literal value in base 2:**

A literal value in base 2 is used to represent a binary integer. The base is determined by the number "2" and the "#" sign. The "+" and "-" signs are prohibited. To make it easier to read, you can use the "_" sign between the bits.

Example:

`2#1111_1111` or `2#11111111` (decimal 255)

`2#1110_0000` or `2#11100000` (decimal 224)

**Literal value in base 8:**

A literal value in base 8 is used to represent an octal integer. The base is determined by the number "8" and the "#" sign. The "+" and "-" signs are prohibited. To make it easier to read, you can use the "_" sign between the bits.

Example:

`8#3_77` or `8#377` (decimal 255)

8#34_0 or 8#340 (decimal 224)

**Literal value of a real with an exponent:**

Number that may be expressed using standard scientific notation. In that case the representation is as follows: mantissa + exponent.

Example:

-1.34E-12 or -1.34e-12

1.0E+6 or 1.0e+6

1.234E6 or 1.234e6

**Literal value of a real:**

A literal real value is a number expressed with one or more decimals.

Example:

-12,0,0,0,+0,456,3,14159_26

**Literal value of an integer:**

A literal value of an integer is used to enter integer values in the decimal system. Values may be preceded by the "+" and "-" signs. Underscore signs (_) separating numbers are not significant.

Example:

-12,0,123_456,+986

**Literal value of time:**

The TIME type has the following units: days (d), hours (h), minutes (m), seconds (s) and milliseconds (ms). A literal value of type TIME is represented by a combination of the preceding types prefixed with T#, t#, TIME# or time#.

Examples: T#25h15m, t#14,7S, TIME#5d10h23m45s3ms

## M

**%M:**

According to the CEI standard, %M indicates a language object of type memory bit.

**Multitoken:**

Operating mode of an SFC. In multitoken mode, the SFC can have several steps that are active simultaneously.

**%MW:**

According to the CEI standard, %MW indicates a language object of type memory word.

## N

**Naming conventions (identifier):**

An identifier is a series of letters, digits, and underscores starting with a letter or an underscore (e.g. the name of a function block type, an instance, a variable, or a section). Accented letters (such as ö, ü, é and õ) may be used, except in names of projects and DFBs. Underscore signs are significant in identifiers. For example, A_BCD and AB_CD are interpreted as different identifiers. You cannot use several underscores in succession or at the start of an identifier.

Identifiers cannot contain spaces. They do not differentiate uppercase and lowercase characters. For example, ABCD and abcd are interpreted as the same identifier.

According to the IEC 61131-3 standard, leading digits are not authorized in identifiers. However, you can use them if, from the **Tools > Project options** dialog box, in the **Language extensions** tab, you check the **Leading digits authorized** box.

Identifiers cannot be keywords.

**NAN:**

Used to indicate that the result of an operation is not a number (NAN = Not A Number).

Example: calculating the square root of a negative number.

> **NOTE:** The CEI 559 standard defines two classes of NAN: the silent NAN (QNAN) and the signaling NAN (SNAN). A QNAN is a NAN with a most significant fraction bit while an SNAN is a NAN without a most significant fraction bit (bit number 22). QNANs can be propagated via most arithmetic operations without throwing an exception. As for SNANs, they generally indicate an invalid operation when they are used as operands in arithmetic operations (see %SW17 and %S18).

**Network:**

There are two meanings of the work "network".

- In LD:

  a network is a set of interconnected graphic elements. The scope of a network is local, concerning the organizational unit (section) of the program containing the network.

- With expert communication modules:

  a network is a set of stations that intercommunicate. The term "network" is also used to define a group interconnected graphic elements. This group then makes up part of a program that may comprise a group of networks.

## P

**Procedure:**

Procedures are technically functional views. The only difference with elementary functions is the fact that procedures can include more than one output and that they handle the VAR_IN_OUT data type. In appearance, procedures are no different from elementary functions.

Procedures are an extension to the IEC 61131-3 standard.

## Q

**%Q:**

According to the CEI standard, %Q indicates a language object of type discrete OUT.

**%QW:**

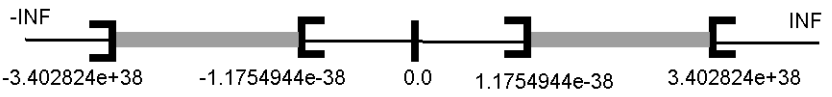
According to the CEI standard, %QW indicates a language object of type analog OUT.

## R

**REAL:**

The REAL type is encoded in a 32 bit format.

The possible value ranges are shown in the figure below:




When a result is:

- between -1,175494e-38 and 1,175494e-38, it is considered to be a DEN;
- less than -3.402824e+38, the symbol -INF (for - infinity) is displayed;
- greater than +3.402824e+38, the symbol INF (for + infinity) is displayed;
- undefined (square root of a negative number), the symbolNAN is displayed.

  **NOTE:** The IEC 559 standard defines two classes of NAN: the silent NAN (QNAN) and the signaling NAN (SNAN). A QNAN is a NAN with a most significant fraction bit while an SNAN is a NAN without a most significant fraction bit (bit number 22). QNANs can be propagated via most arithmetic operations without throwing an exception. As for SNANs, they generally indicate n invalid operation when they are used as operands in arithmetic operations (see %SW17 and %S18).

**NOTE:** When a DEN (non-standardized number) is used as an operand, the result is not significant.

## S

**SFC:**

SFC is the abbreviation of Sequential Function Chart.

An SFC can be used to graphically represent in a structured manner the operation of a sequential PLC. This graphical description of the PLC's sequential behavior and of the various resulting situations is created using simple graphic symbols.

**SIL:**

Safety Integrity Level

Safety functions are executed to achieve and maintain the safe state of a system. The IEC 61508 specifies 4 levels of safety performance for a safety function. These are called safety integrity levels (SIL), ranging from 1 (the lowest) to 4 (the highest). The Quantum Safety PLC is certified for use in SIL2 applications in which the de-energized state is the safe state, for example in an Emergency Shutdown (ESD) system.

You can use the Schneider safety products for creating a Hot Standby (HSBY) solution if you require high availability for a safety system.

**Single token:**

Operating mode for an SFC diagram in which only one step can be active at a given time.

**SNMP:**

Simple Network Management Protocol.

**STRING:**

A STRING variable is a series of ASCII characters. The maximum length of a string is 65,534 characters.

**ST:**

ST is the abbreviation of Structured Text.

The structured literal language is a developed language similar to computer programming languages. It can be used to organize a series of instructions.

## T

**TIME_OF_DAY:**

See TOD.

**TIME:**

The TIME type expresses a time in milliseconds. Encoded in 32 bits, this type can be used to obtain times from 0 to $2^{32}$-1 milliseconds.

The TIME type has the following units: days (d), hours (h), minutes (m), seconds (s) and milliseconds (ms). A literal value of type TIME is represented by a combination of the preceding types prefixed with T#, t#, TIME# or time#.

Examples: T#25h15m, t#14,7S, TIME#5d10h23m45s3ms

**TOD:**

TOD is the abbreviation of Time Of Day.

The TOD type, encoded in BCD in a 32 bit format, contains the following information:

- the hour encoded in an 8 bit field;
- the minutes encoded in an 8 bit field;
- the seconds encoded in an 8 bit field.

**NOTE:** The 8 least significant bits are not used.

The TOD type must be entered as follows: **TOD#** <Hour>**:**<Minutes>**:**<Seconds>

This table shows the upper/lower limits of each field:

| Field | Limits | Comment |
|---|---|---|
| Hour | [00,23] | The leading 0 is displayed; it can be omitted during data entry. |
| Minute | [00,59] | The leading 0 is displayed; it can be omitted during data entry. |
| Second | [00,59] | The leading 0 is displayed; it can be omitted during data entry. |

Example: TOD#23:59:45.

**Token:**

Active step in an SFC.

**TOPO_ADDR_TYPE:**

This predefined type is used as an output for the READ_TOPO_ADDR function. This is an ARRAY[0..4] OF Int. You can find it in the library, in the same family as the EFs that use it.

## U

**UDINT:**

UDINT is the abbreviation of Unsigned Double INTeger (encoded in 32 bits). The upper/lower limits are as follows: 0 to (2 to the power of 32) - 1.

Example:

0, 4294967295, 2#11111111111111111111111111111111, 8#37777777777, 16#FFFFFFFF.

**UDP:**

*user datagram protocol*. UDP is a connectionless Internet communications protocol defined by IETF RFC 768. This protocol facilitates the direct transmission of datagrams on IP networks. UDP/IP messages do not expect a response, and are therefore ideal for applications in which dropped packets do not require retransmission (such as streaming video and networks that demand real-time performance).

**UINT:**

UINT is the abbreviation of the Unsigned INTeger format (encoded in 16 bits). The upper/lower limits are as follows: 0 to (2 to the power of 16) - 1.

Example:

0, 65535, 2#1111111111111111, 8#177777, 16#FFFF.

**Unassigned variables:**

A variable whose position in the PLC memory cannot be known. A variable that is not linked to an address is called unassigned variable.

## V

**Variable:**

Memory entity of type BOOL, WORD, DWORD, etc., whose contents can be modified by the program currently running.

## W

**WORD:**

The type WORD is encoded in a 16 bit format and is used to perform processing on series of bits.

This table shows the upper/lower limits of each of the bases that can be used:

| Base | Lower limit | Upper limit |
|---|---|---|
| Hexadecimal | 16#0 | 16#FFFF |
| Octal | 8#0 | 8#177777 |
| Binary | 2#0 | 2#1111111111111111 |

Examples of representation

| Data | Representation in one of the bases |
|---|---|
| 0000000011010011 | 16#D3 |
| 1010101010101010 | 8#125252 |
| 0000000011010011 | 2#11010011 |

# Index

## P



## S



## T



## U



## V



## W

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison
France

+ 33 (0) 1 41 29 70 00

www.se.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.



EIO0000000359.01