Air Handling Unit Baselines

Developer's Manual

invensys™
Controls

Table of contents

1. Introduction

The purpose of this manual is to guide the developer in the use of **Eliwell FREE Studio** and Eliwell's Air Handling Unit (AHU) Baselines to build a dedicated application for Air Handling Units.

1.1. FREE Studio

The AHU Baselines are compatible with **FREE Studio 2.0** or later versions

1.2. EULA

To install **FREE Studio** and the Air Handling Unit Baselines, you must accept the terms of the user license.

Read the End User License Agreement (EULA) carefully before continuing.

End User License Agreement is also available on the website
http://www.eliwell.it/content.aspx?id=4533

2. Glossary

DEVELOPER: designer / developer with knowledge of one or more IEC61131-3 standard programming languages. FREE Applicaton user

USER: end user typically using FREE Device. This person is not expected to be able to compile code. The USER must have access to adequate documentation.

RESPONSIBILITY: blocks functionality description

COLLABORATIONS: interactions between Baseline Architecture blocks

2.1. Abbreviations and definitions

A.H.U.: Air Handling Unit

Application, Device, Connection: abbreviations of **FREE (Studio) Application, FREE Device**, and **Free Connection**, respectively. Software suites

IEC Application, PLC Application, PLC PROGRAM: application developed in compliance with IEC61131-3 (industrial control programming standards) by means of the Application development environment (tool), to download to the target using Application or Device

Target device, Target: name given to the FREE Smart or FREE Evolution programmable controller or “instrument”

HMI: Human Machine Interface. Graphic interface developed with Free Studio for FREE Smart and SKP SKW terminals

Instance: object of a predefined 'class' of objects (function block, template, etc.)

IEC Language: programming language developed in compliance with IEC61131-3 (e.g. FBD, SFC)

BIOS menu, BIOS: factory-set BIOS parameters menu. The Bios cannot be edited.

Smart: abbreviation **of FREE Smart; Evolution**: abbreviation of **FREE Evolution**

Studio: abbreviation of **FREE Studio**. The software suite described in this document

Tab or form. The work environment is divided into sections or panels. Each panel may in turn be subdivided into forms or tabs (e.g. Resources tab)

Note: Many definitions and abbreviations are standard information technology and/or PLC terms and are not listed here.

For example a Function is a standard term. Other terms, such as 'Block,' will be described in the relevant headings.

3. Baseline architecture

This section describes the high level software structure (the architecture) of Eliwell baselines, introducing information that aids understanding of the starting application, with the result of facilitating the work of a developer who intends to work on the software to create a dedicated application.

3.1. Block diagram

The block diagram in Figure 1 represents the architecture of the Baselines. The single items in the diagram and their connections are described in the following headings.

The entire architecture is described in detail in **Appendix – Architecture** with descriptions of the RESPONSIBILITY of the individual blocks and the relationships (called Collaborations) between the GOAL block and the other blocks

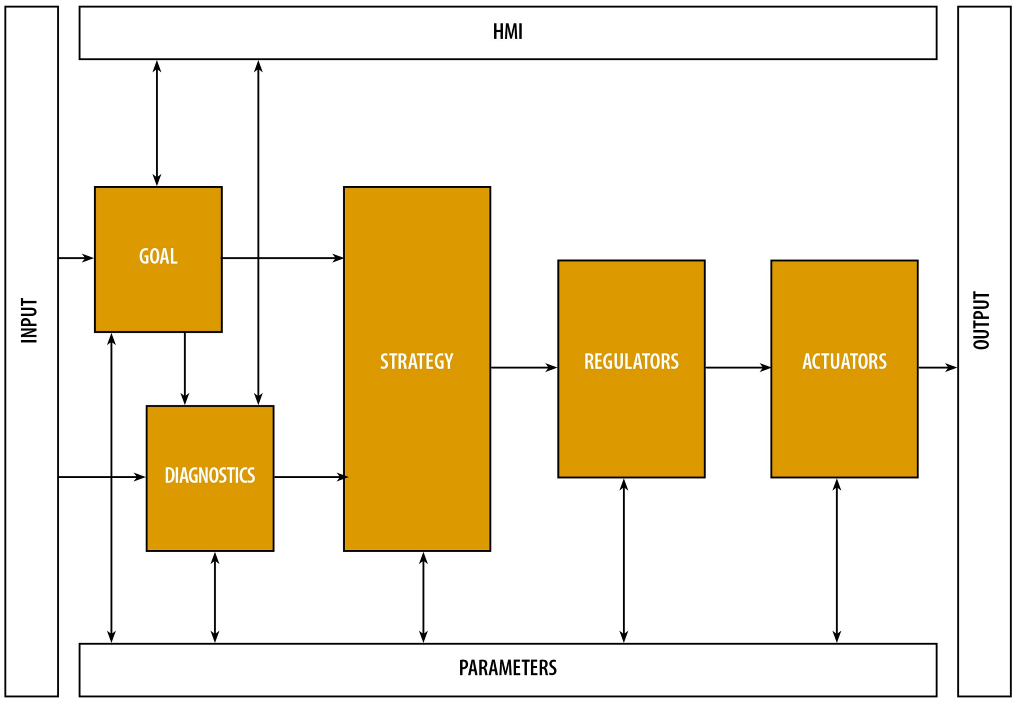


Figure 1: High level software structure

3.2. GOAL block

The GOAL block establishes the aims that are to be pursued in the unit control strategy.

Specifically, the data resulting from the GOAL block are:

- unit ON/OFF status;
- COOL/HEAT (summer/winter) unit operating mode;
- energy saving mode activation status (ECO mode);
- setpoints to follow;
- measurements (feedback) to be used for regulation, deriving from the choice of regulation probes.

The GOAL block is required to summarize these data starting from the different possible sources, which include parameters and digital and analog inputs.

EXAMPLE 3.1

As an example, consider how ON/OFF status is determined in the Baselines.

A digital input dedicated to forcing OFF status (remote OFF) is provided: when the input is high the unit status is forced to OFF.

When the remote OFF input is inactive, unit status depends on the enabling of time frame operation: if this is the case, ON/OFF status is determined by the active time frame settings, otherwise it is determined by a user parameter (e.g. set by graphic interface or by **Device**).

The GOAL block evaluates and assigns a priority level to the conditions listed above, determining the resulting ON/OFF status.

The GOAL block is not concerned with whether or how the goals can be achieved, delegating this task to the downstream blocks.

Refer to **Appendix – Architecture** for further information on the relationships between the GOAL block and the other blocks.

3.2.1. Software form in AHU Baselines

Working with an AHU Baseline, the GOAL block is composed of a series of programs written in FBD programming language, the names of which start with `Goal_`.

EXAMPLE 3.2

For example, the program responsible for determining the AHU ON/OFF status is `Goal_State`.

The results of the GOAL block are made available to the remaining parts of the application as global data in the *Global shared* > *Variables* folder.

3.3. DIAGNOSTICS block

The DIAGNOSTICS block evaluates the presence of fault conditions in the unit and controls activation and deactivation (reset) of the alarms.

The tests performed by the DIAGNOSTICS block include:

- evaluation of the digital diagnostic inputs for detection of unit fault conditions;

> **EXAMPLE 3.3**
>
> For example, the DIAGNOSTICS block evaluates the status of the air handling unit fans thermal protections.

- checking of values transmitted by probes in order to detect absent or faulty probes;
- evaluation of analog inputs associated with safeties/pre-alarms;

> **EXAMPLE 3.4**
>
> Again by way of example, the DIAGNOSTICS block of the AHU baselines checks that the temperature detected by the antifreeze probe is not below a value defined as critical.

The DIAGNOSTICS block is responsible for detecting error conditions, but not for remedying them.

Refer to **Appendix – Architecture** for further information on the relationships between the DIAGNOSTICS block and the other blocks.

3.3.1. Software form in the AHU baselines

Working with an AHU baseline, the DIAGNOSTICS block is constituted by the program of the same name, written in FBD language.

The results of the DIAGNOSTICS block are made available as alarm variables, grouped in the *Global shared* > *Alarms* folder.

Each network of the `Diagnostics` program is responsible for assigning a value to an individual alarm.

3.4. STRATEGY block

The STRATEGY block is responsible for choosing the strategy to use to achieve the aims determined by the GOAL block. In detail, this consists of:

- evaluating, on the basis of the readings from the field and the aims calculated by the GOAL block, whether there are requests to be fulfilled or pre-alarm or emergency situations that must be remedied;
- assigning priority to pending requests;

EXAMPLE 3.5

For example, in the AHU baselines it may occur that neither temperature nor humidity goals are fulfilled.

In this situation, the STRATEGY block decides which of the two goals is to be pursued first.

- selecting the strategy to use to fulfill the request from among the various possible strategies.

EXAMPLE 3.6

Still considering the AHU baselines, a cooling request can be fulfilled by the use of a cooler or, in certain conditions, by means of the free-cooling mechanism, which is more economical.

The STRATEGY block evaluates the necessary conditions for activation of free-cooling (notably, ambient air temperature must be within a clearly defined range of values): if these conditions are fulfilled, free-cooling is adopted as a strategy to fulfill the cooling request; otherwise, exclusively the cooler is utilized.

The strategy chosen by the STRATEGY block determines the enabling of the various components of the unit and the data to be used in their control - such as the setpoints to follow and the field measurements (feedback) to use.

EXAMPLE 3.7

For example, the cooling strategy enables the cooler regulator by sending it the value read by the thermoregulation probe and the value of the temperature to be reached (temperature setpoint in COOL mode).

Vice versa, heating coil regulation is disabled.

Refer to **Appendix – Architecture** for further information on the relationships between the STRATEGY block and the other blocks

3.4.1. Software form in the AHU baselines

Working with an AHU baseline, the STRATEGY block is constituted by the program of the same name, written in SFC language.

The STRATEGY block is a model of the unit's state machine, defining:

- the actions to be performed (i.e. the strategy to adopt) for each machine state;
- the possible transitions between states and the conditions in which they occur.

The chosen programming language (SFC) allows efficient translation of this state machine:

- the strategies correspond to ACTIONS of the SFC program, implemented in FBD language: each strategy contains an FBD network for each unit component;
- the transitions correspond to SFC program TRANSITIONS implemented in ST language.

3.5. REGULATORS and ACTUATORS blocks

The values assumed by the physical outputs of the controller are established by the set of REGULATORS and ACTUATORS blocks, which combine the regulation and outputs actuation logics, respectively.

EXAMPLE 3.8

In tangible terms, regulation logic is construed, for example, as a proportional regulator, a proportional-integral regulator, or a steps regulator.

For an example of actuation logic we can think of the actuator for a 3-point valve.

Collectively, the REGULATORS and ACTUATORS blocks perform the task of fulfilling the requests that the STRATEGY block expresses in terms of setpoints or percentage values.

EXAMPLE 3.9

For example, when the cooling strategy enables the cooler and assigns to it a value of 25°C, the REGULATOR block translates this request into a percent actuation value that can be assigned directly to an analog output or processed by the ACTUATOR block (if present), which translates it into values of digital and/or analog outputs.

Refer to **Appendix – Architecture** for further information on the relationships between REGULATORS and ACTUATORS blocks and the other blocks.

3.5.1. Software form in the AHU baselines

The REGULATORS and ACTUATORS blocks are grouped into functional blocks that represent the physical components of the air handling unit.

EXAMPLE 3.10

For example, the AHU baselines have a block for the outlet fan, one for the heating coil, one for the cooler, one for the humidifier, etc.

Each strategy determines the inputs of all the blocks and causes their execution.

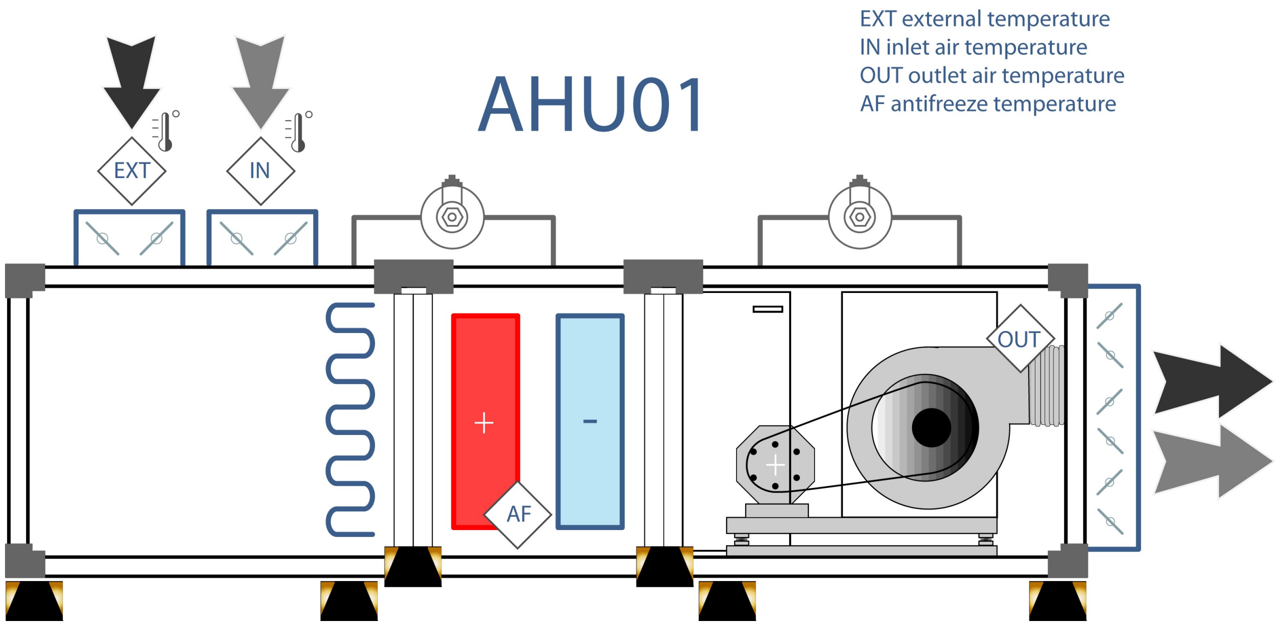
4. Derived applications

This section describes the operations to be performed to derive a dedicated application starting from the Eliwell AHU baselines.

4.1. Choosing the baseline

The first decision to make concerns the AHU baseline from which to start: the choice mainly depends on the type of AHU that the application must control.

Table 1 lists the main differences between the AHU baselines.

AHU Baselines
Layout 1 is an application dedicated controlling a simple AHU composed of: • ON/OFF dampers • a single fan (outlet), • electric heater, • cooler. The only goal pursued by layout 1 is to maintain the temperature of the controlled room within a given range of values.
 AHU01 EXT external temperature IN inlet air temperature OUT outlet air temperature AF antifreeze temperature EXT IN + - AF OUT

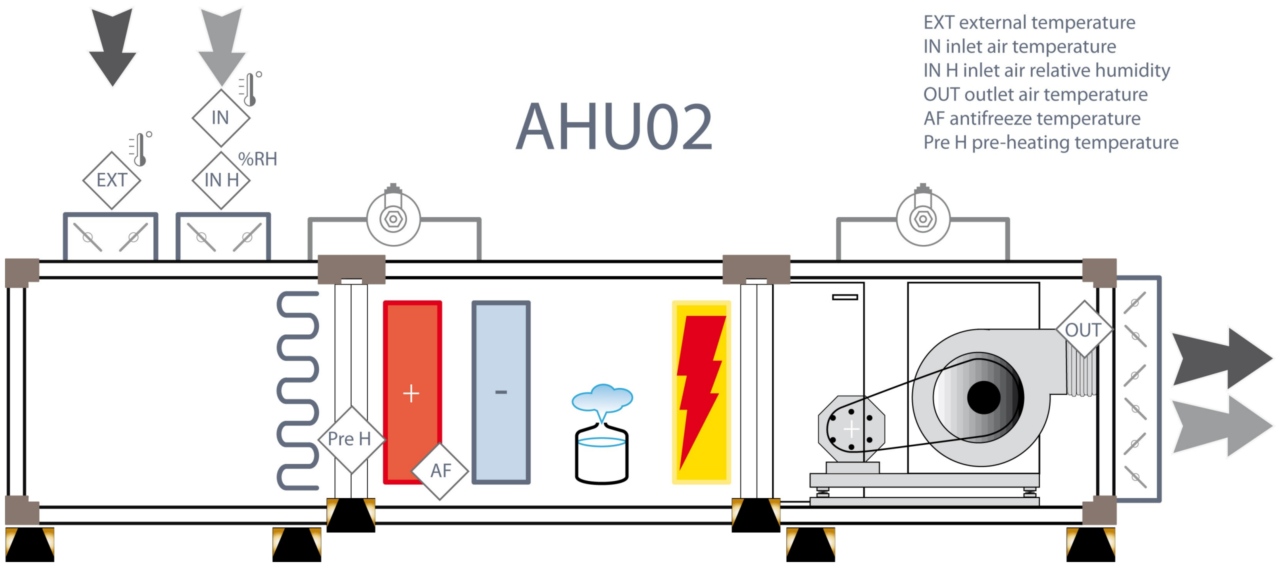
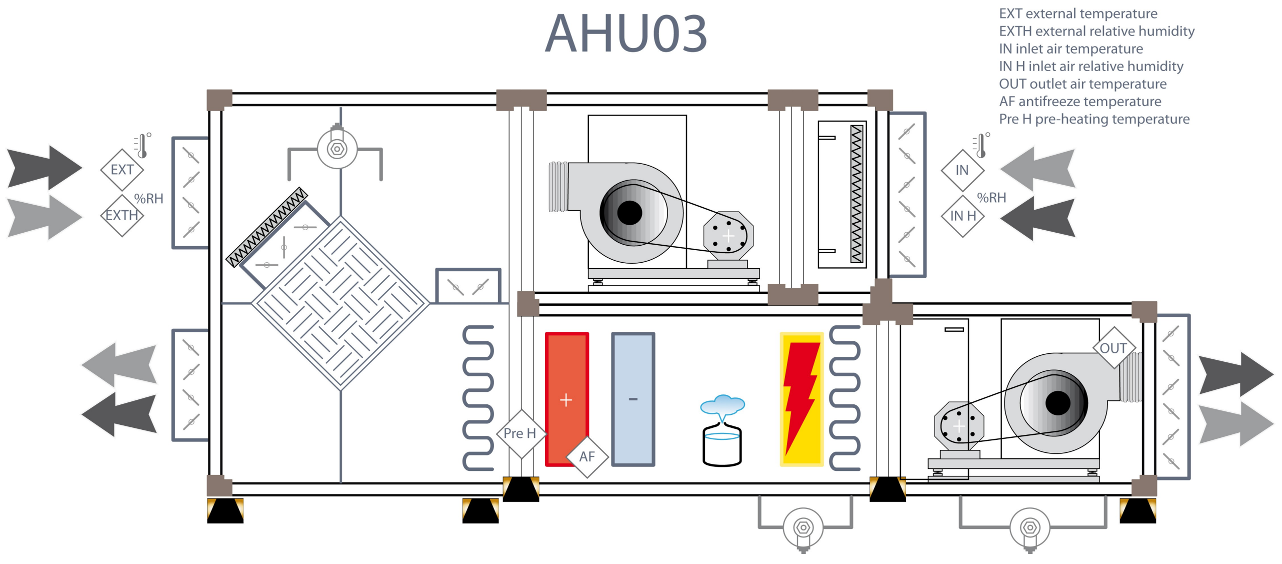
AHU Baselines
Layout 2 extends layout 1 to include: • humidifier, • post-heater (the component used for the dehumidification process). This layout has the further goal of maintaining the relative humidity of the controlled room within a given range of values.
 AHU02 EXT external temperature IN inlet air temperature IN H inlet air relative humidity OUT outlet air temperature AF antifreeze temperature Pre H pre-heating temperature EXT IN IN H %RH Pre H AF OUT + -
Layout 3 is an application dedicated to the control of a complete AHU, which includes all the elements present in the previous layouts, and also: • a second fan (inlet), • modulating control of dampers (for free-cooling/free-heating), • heat recovery unit
 AHU03 EXT external temperature EXTH external relative humidity IN inlet air temperature IN H inlet air relative humidity OUT outlet air temperature AF antifreeze temperature Pre H pre-heating temperature EXT %RH EXTH IN %RH IN H Pre H AF OUT + -

Table 1: List of AHU baselines

The choice of the starting baseline can be guided by the following considerations:

- if the AHU has the sole goal of temperature control and differs significantly from the characteristics of layout 3 (for example, it does not incorporate management of free-cooling/free-heating, modulating dampers and inlet fan), it may be more practical to start from layout 1;
- if the AHU has the dual goals of temperature and relative humidity control and differs significantly from the characteristics of layout 3 (for example, it does not incorporate management of free-cooling/free-heating, modulating dampers and inlet fan), it may be more practical to start from layout 2;
- in all other cases, it may be more practical to start from layout 3.

4.2. Deriving a new Application project from the baseline

Once the starting baseline has been chosen, a copy must be made in order to edit it so that a dedicated application can be derived.

The steps to follow are:

1. open the Application project corresponding to the chosen baseline;
2. save the application with a new name (*File menu* > *Save project as...*).

Refer to the **Application** user manual for more details.

4.3. Making changes to the application

Once you have made a working copy of the baseline, you can edit the source code in such a way as to introduce all the differences required by the specific application circumstances.

For an introduction to the high level structure of the application, refer to Chapter 3.

Even though there are no limitations to the type and number of changes that can be made, Chapter 5 describes the most frequent cases, illustrating the methods of proceeding and showing some practical examples.

5. Editing the application

This section describes the most frequent actions that must be taken on the baselines and provides indications on how to edit the application, providing numerous practical examples.

5.1.1. I/O static configuration

It is frequently necessary to map the physical I/O (inputs and outputs of the hardware that implements the application) on the logical I/O, i.e. on the symbols utilized in the application to refer to the input and output values.

EXAMPLE 5.1

Working on an application for **Smart** derived from layout 2, we decide to move the digital input corresponding to a humidifier alarm from the extended I/O to the local one.

Application allows editing of the I/O configuration, in table form, in the section *Resources > I/O Mapping* of the development environment: simply arrange the symbolic names in the grids in the way that best adapts to the specific application case in order to fulfill this requirement.

EXAMPLE 5.2

Considering Example 5.1, to achieve the required result we can follow this procedure:

1. open the section *Resources* > *I/O Mapping* > *Local*;
2. choose the digital input to be used (let us assume DIL3), and edit the contents of the corresponding cell by entering the symbolic name of the humidifier alarm, `I_HumidifierAlarm`;
3. open the section *Resources* > *I/O Mapping* > *Extended* and remove the symbolic name of the humidifier alarm from its prior position;
4. the logical input replaced in point 2, `I_ElectricHeaterThermal`, can be assigned to an unused digital input of the expansion.

5.1.2. Dynamic configuration of the I/O

It may be required to make the mapping of the physical I/O on the logical I/O depend on the value of one or more user parameters, in such a way as to be editable also after the development stage (e.g. at the time of installation).

EXAMPLE 5.3

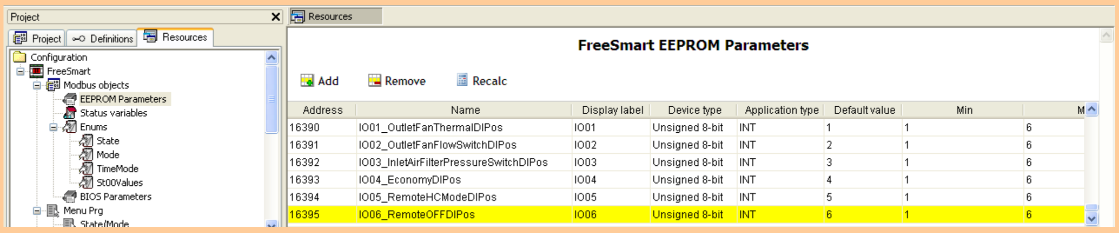
Working on an application for **Smart** derived from layout 1 that makes use of all and exclusively the local digital inputs (DIL1...6), we want the allocation of these inputs to the corresponding logic symbols to be parametric.

To achieve this result we must introduce another software level as a new PROGRAM assigned to the Timed task, which, on the basis of the configuration parameter values, assigns to each digital input the values read by the corresponding digital input/by the corresponding probe and - in the opposite sense - assigns the value of each logical output to the corresponding digital/analog output.

EXAMPLE 5.4

To implement the requirement expressed in Example 5.3, the following procedure can be adopted:

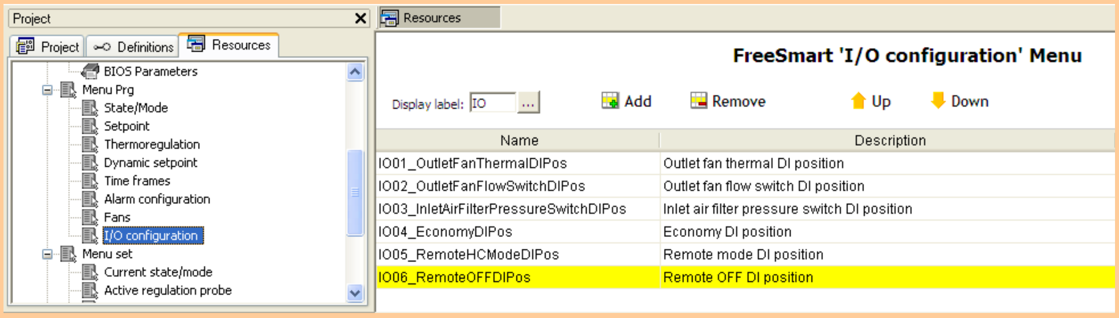
1. in the section *Resources* > *EEPROM Parameters*, six new parameters are defined (e.g. IO01, IO02, ..., IO06), one for each logical input (e.g., IO01 refers to `I_OutletFanThermal`, IO02 to `I_OutletFanFlowSwitch`, etc.), the value of which, between 1 and 6, identifies the corresponding local digital input;



FreeSmart EEPROM Parameters

Address	Name	Display label	Device type	Application type	Default value	Min	M
16390	IO01_OutletFanThermalDIPos	IO01	Unsigned 8-bit	INT	1	1	6
16391	IO02_OutletFanFlowSwitchDIPos	IO02	Unsigned 8-bit	INT	2	1	6
16392	IO03_InletAirFilterPressureSwitchDIPos	IO03	Unsigned 8-bit	INT	3	1	6
16393	IO04_EconomyDIPos	IO04	Unsigned 8-bit	INT	4	1	6
16394	IO05_RemoteHCModeDIPos	IO05	Unsigned 8-bit	INT	5	1	6
16395	IO06_RemoteOFFDIPos	IO06	Unsigned 8-bit	INT	6	1	6

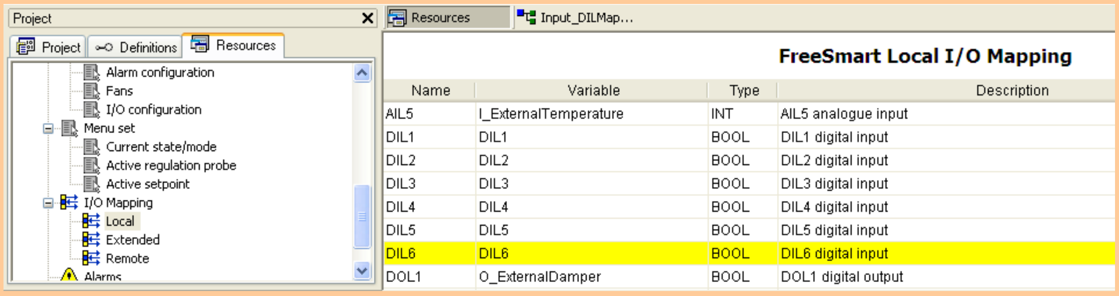
2. in the section *Resources* > *Menu Prg*, the new parameters are published in a menu;



FreeSmart 'I/O configuration' Menu

Name	Description
IO01_OutletFanThermalDIPos	Outlet fan thermal DI position
IO02_OutletFanFlowSwitchDIPos	Outlet fan flow switch DI position
IO03_InletAirFilterPressureSwitchDIPos	Inlet air filter pressure switch DI position
IO04_EconomyDIPos	Economy DI position
IO05_RemoteHCModeDIPos	Remote mode DI position
IO06_RemoteOFFDIPos	Remote OFF DI position

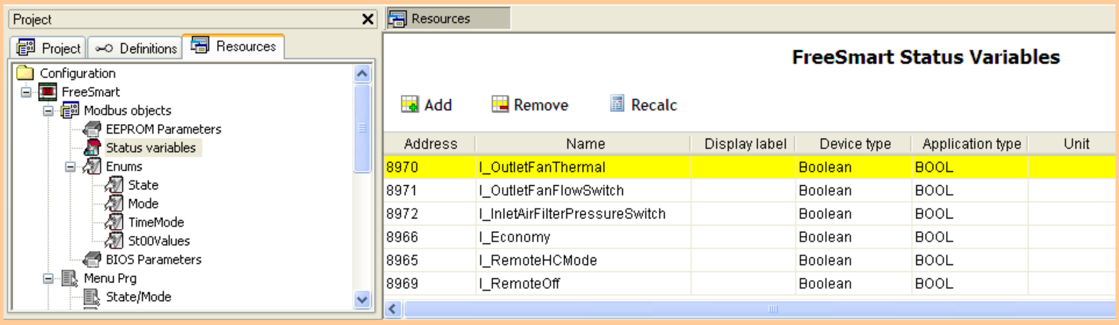
3. in the section *I/O Mapping* > *Local*, generic names are assigned to the variables corresponding to the local digital inputs;



FreeSmart Local I/O Mapping

Name	Variable	Type	Description
AIL5	I_ExternalTemperature	INT	AIL5 analogue input
DIL1	DIL1	BOOL	DIL1 digital input
DIL2	DIL2	BOOL	DIL2 digital input
DIL3	DIL3	BOOL	DIL3 digital input
DIL4	DIL4	BOOL	DIL4 digital input
DIL5	DIL5	BOOL	DIL5 digital input
DIL6	DIL6	BOOL	DIL6 digital input
DOL1	O_ExternalDamper	BOOL	DOL1 digital output

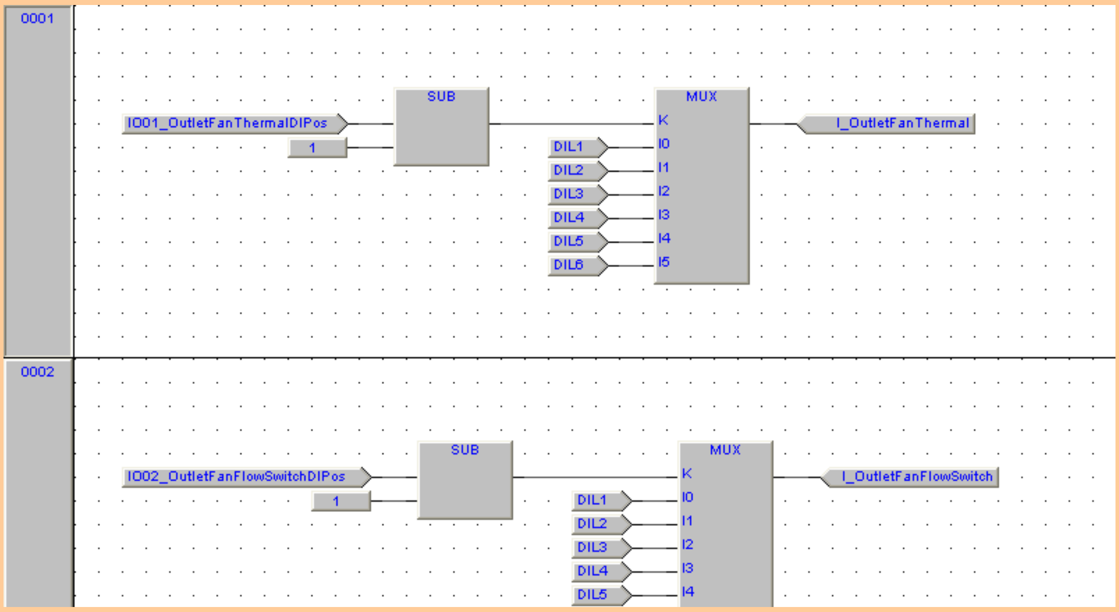
4. the symbols that were previously assigned to the local digital inputs are defined differently, for example as status variables (section *Resources > Status variables*);



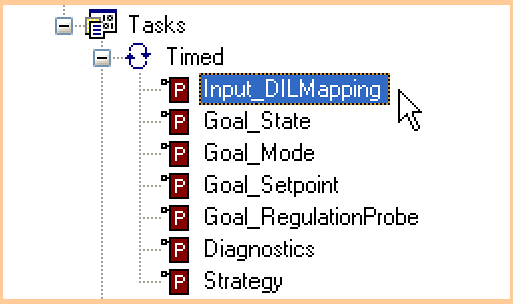
FreeSmart Status Variables

Address	Name	Display label	Device type	Application type	Unit
8970	I_OutletFanThermal		Boolean	BOOL	
8971	I_OutletFanFlowSwitch		Boolean	BOOL	
8972	I_InletAirFilterPressureSwitch		Boolean	BOOL	
8966	I_Economy		Boolean	BOOL	
8965	I_RemoteHCMode		Boolean	BOOL	
8969	I_RemoteOff		Boolean	BOOL	

5. a new PROGRAM is created in FBD language, with the name `Input_DILMapping`, comprising an FBD network for each logical input; each FBD network selects the local digital input to be used on the basis of the value read by the corresponding parameter;



6. the new PROGRAM is assigned to the Timed task as the first PROGRAM to execute: the rest of the application remains unchanged.



Tasks
Timed
Input_DILMapping
Goal_State
Goal_Mode
Goal_Setpoint
Goal_RegulationProbe
Diagnostics
Strategy

5.2. ON/OFF status

The `Goal_State` PROGRAM determines the ON/OFF status (more precisely, the status can be OFF, STD_BY or ON) of the AHU and the activation status of specific operating modes, such as ECO (Economy) mode.

The result of the `Goal_State` execution is encoded in the `state` and `economy` status variables: this means that it is possible to edit or even completely replace the PROGRAM without affecting the rest of the application, as long as these status variables are assigned the correct values.

In the AHU baselines, the unit status is determined by analyzing the inputs and user parameters dedicated to this purpose, in accordance with a priority encoded in the `Goal_State` PROGRAM code (refer to Figure 2).

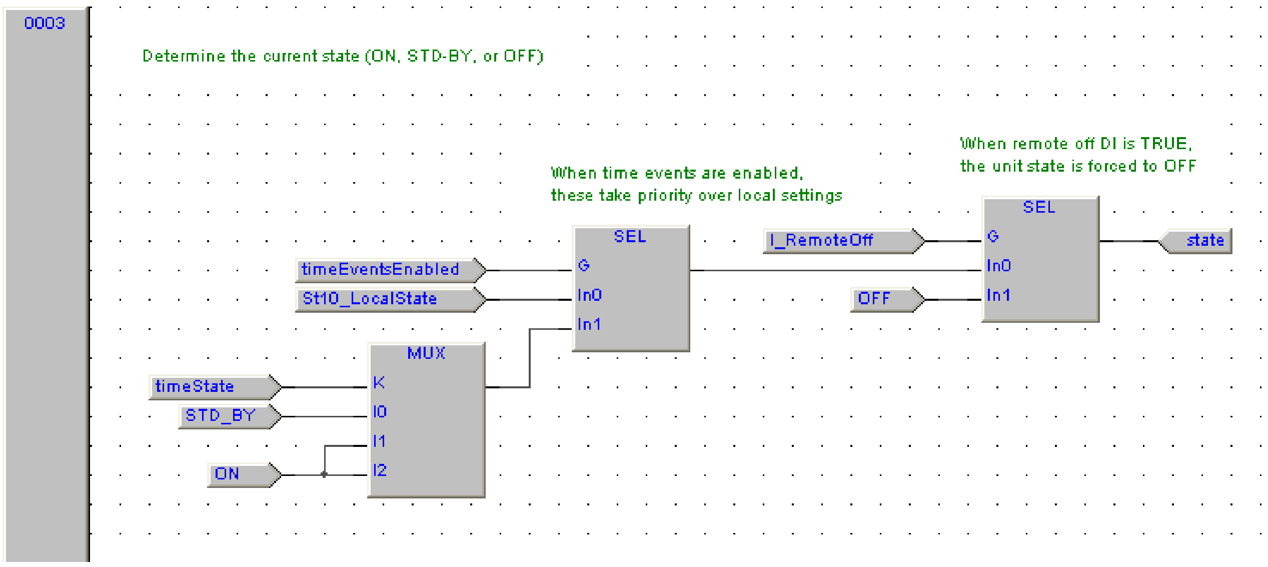

0003
Determine the current state (ON, STD-BY, or OFF)
When time events are enabled,
these take priority over local settings
When remote off DI is TRUE,
the unit state is forced to OFF
SEL
timeEventsEnabled
St10_LocalState
G
In0
In1
I_RemoteOff
OFF
SEL
G
In0
In1
state
MUX
timeState
STD_BY
ON
K
I0
I1
I2

Figure 2: Determination of the AHU status starting from the remote digital OFF input, from local state user parameter St10 and from the information derived from time frame operation, if enabled

Moreover, `Goal_State` is responsible for determining whether Economy mode is active or not (refer to Figure 3).

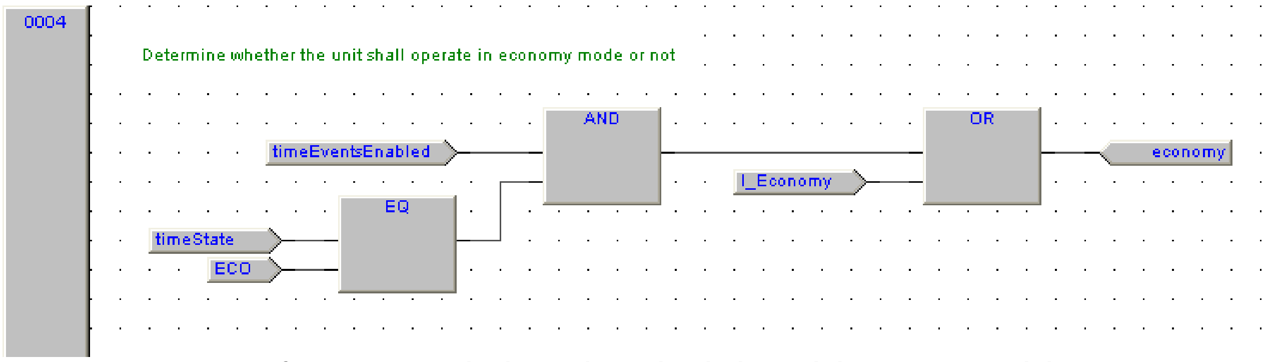

0004
Determine whether the unit shall operate in economy mode or not
AND
OR
timeEventsEnabled
I_Economy
economy
EQ
timeState
ECO

Figure 3: Activation of Economy mode depends on the dedicated digital input and the current value of the machine mode derived from time frame operation, if enabled

The same PROGRAM initially establishes the current time frame so that it can utilize the associated information (see Figure 4).

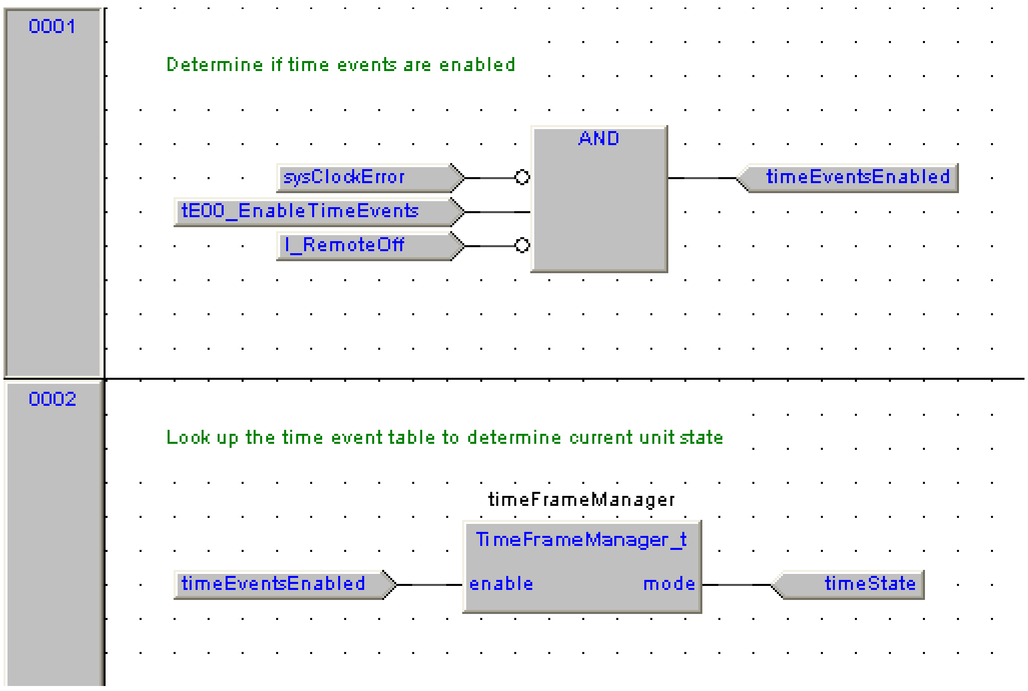


Figure 4: Enabling time frame operation depends on the availability of the system clock and the value of parameter tE00; the current time frame is established by using the library block `TimeFrameManager_t`

5.2.1. Editing the algorithm that determines ON/OFF status

The data to consider to establish the air handling unit's ON/OFF status and the priority to assign to said data can be significantly differentiated from case to case. The need frequently occurs to modify the algorithm for calculation of ON/OFF status.

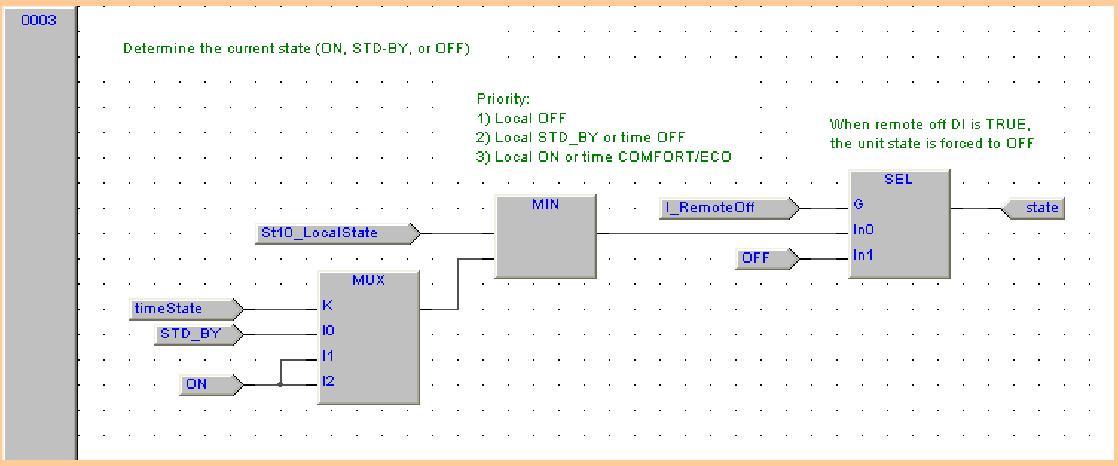
EXAMPLE 5.5

For example, consider a situation wherein it is not suitable to always award priority to time frames operation (if enabled) compared to status setting via user parameter, but in which the parameter-imposed OFF assumes priority with respect to a time-imposed ON status.

To do this, simply edit the FBD network that establishes the value of the status variable `state` within the `Goal_State` program.

EXAMPLE 5.6

Considering the need expressed in Example 5.5, network number 3 of the `Goal_State` PROGRAM can be edited in such a way as to force OFF status if local parameter St10 is OFF.


0003
Determine the current state (ON, STD-BY, or OFF)
Priority:
1) Local OFF
2) Local STD_BY or time OFF
3) Local ON or time COMFORT/ECO
When remote off DI is TRUE,
the unit state is forced to OFF
St10_LocalState
MIN
I_RemoteOff
OFF
SEL
G
In0
In1
state
MUX
K
I0
I1
I2
timeState
STD_BY
ON

5.2.2. Editing time frame operation

Time frame operation can be replaced with personalized operation depending on the needs of the specific application case.

EXAMPLE 5.7

Working on a derived application assume we want to insert a COMFORT 24/24h profile (COMFORT always), to be used in weekly programming.

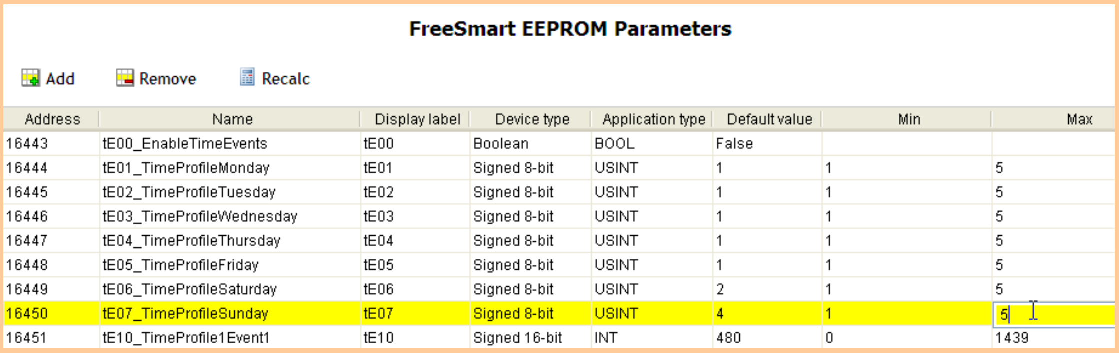
To do this, proceed as follows:

- replace the time frame operation block with a different one, which can be appropriately derived from the first one;
- edit (add/remote, redefine) the set of user parameters required to save the time frames.

EXAMPLE 5.8

Considering the requirements expressed in Example 5.7, proceed as follows:

1. change the definition of parameters tE01, tE02, ..., tE07, to which it must be possible to assign the value corresponding to the added profile;



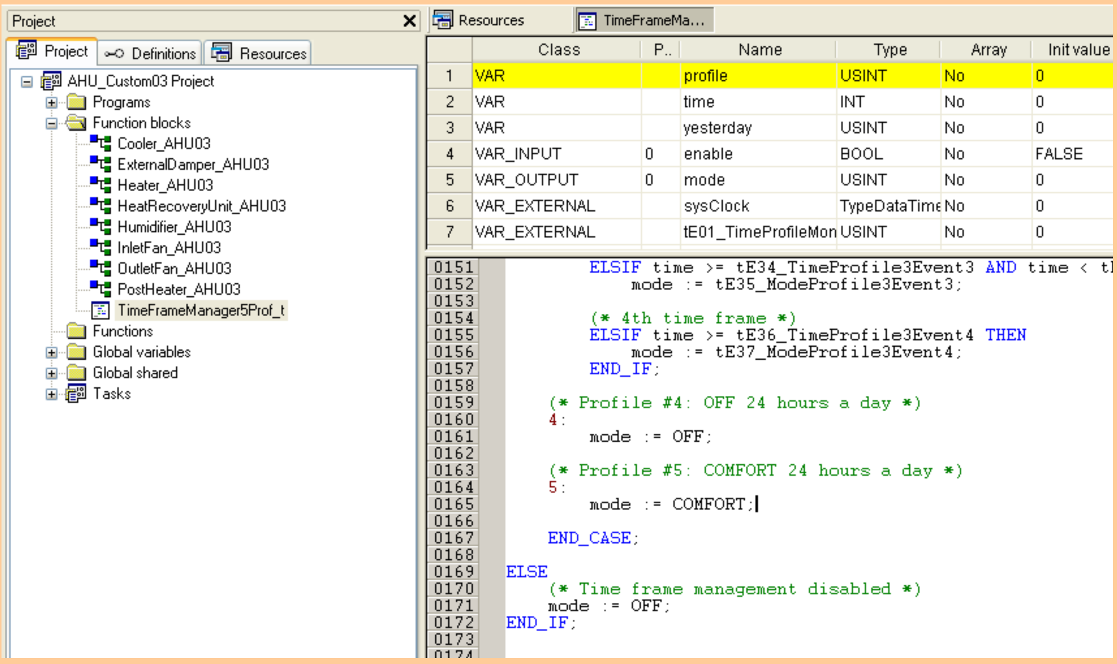
FreeSmart EEPROM Parameters

Add   Remove   Recalc

Address	Name	Display label	Device type	Application type	Default value	Min	Max
16443	tE00_EnableTimeEvents	tE00	Boolean	BOOL	False		
16444	tE01_TimeProfileMonday	tE01	Signed 8-bit	USINT	1	1	5
16445	tE02_TimeProfileTuesday	tE02	Signed 8-bit	USINT	1	1	5
16446	tE03_TimeProfileWednesday	tE03	Signed 8-bit	USINT	1	1	5
16447	tE04_TimeProfileThursday	tE04	Signed 8-bit	USINT	1	1	5
16448	tE05_TimeProfileFriday	tE05	Signed 8-bit	USINT	1	1	5
16449	tE06_TimeProfileSaturday	tE06	Signed 8-bit	USINT	2	1	5
16450	tE07_TimeProfileSunday	tE07	Signed 8-bit	USINT	4	1	5
16451	tE10_TimeProfile1Event1	tE10	Signed 16-bit	INT	480	0	1439

2. import a copy of the `TimeFrameManager_t` block from the Utils library;

3. edit the source code of the local copy of the `TimeFrameManager_t` block in such a way as to manage the added profile.



	Class	P..	Name	Type	Array	Init value
1	VAR		profile	USINT	No	0
2	VAR		time	INT	No	0
3	VAR		yesterday	USINT	No	0
4	VAR_INPUT	0	enable	BOOL	No	FALSE
5	VAR_OUTPUT	0	mode	USINT	No	0
6	VAR_EXTERNAL		sysClock	TypeDataTime	No	0
7	VAR_EXTERNAL		tE01_TimeProfileMon	USINT	No	0

```
        ELSIF time >= tE34_TimeProfile3Event3 AND time < t
            mode := tE35_ModeProfile3Event3;

        (* 4th time frame *)
        ELSIF time >= tE36_TimeProfile3Event4 THEN
            mode := tE37_ModeProfile3Event4;
        END_IF;

    (* Profile #4: OFF 24 hours a day *)
    4:
        mode := OFF;

    (* Profile #5: COMFORT 24 hours a day *)
    5:
        mode := COMFORT;

    END_CASE;

ELSE
    (* Time frame management disabled *)
    mode := OFF;
END_IF;
```

5.3. COOL/HEAT mode

The `Goal_Mode` PROGRAM determines the COOL/HEAT mode of air handling unit operation.

The result of `Goal_Mode` execution is encoded in the `hcMode` status variable: this means that it is possible to edit or even completely replace the PROGRAM without affecting the rest of the application, as long as these status variables are assigned the correct values.

In the AHU baselines, COOL/HEAT mode is established by cross-referencing the inputs value and dedicated user parameters with the value of an additional user parameter that defines the selectable modes (see Figure 5).

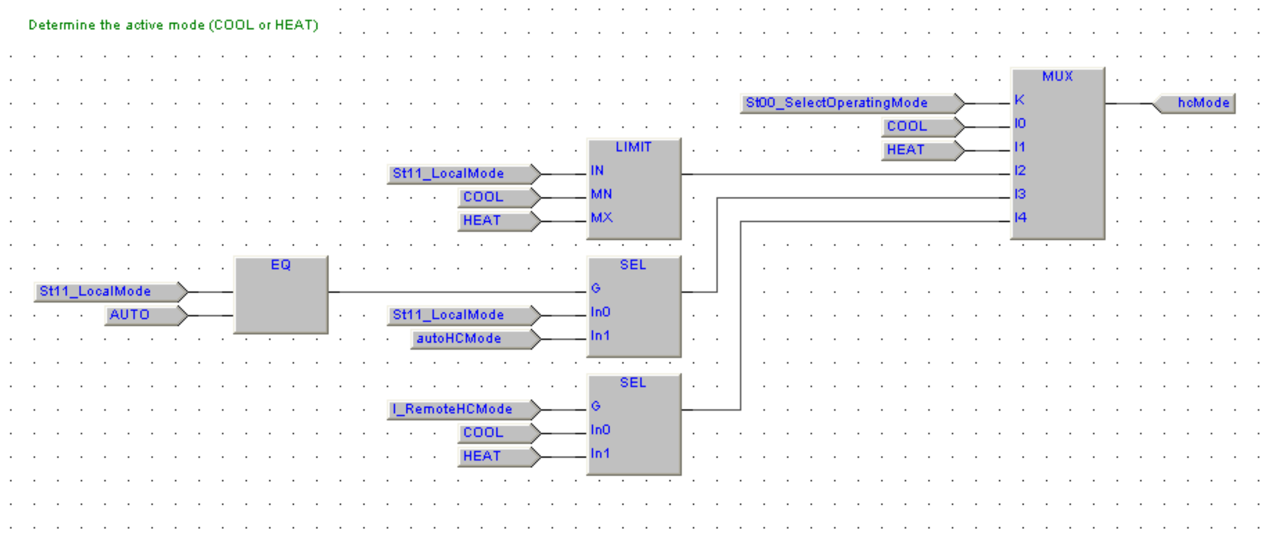


Figure 5: Determining the COOL/HEAT operating mode of the air handling unit; parameter St00 limits the selectable modes and defines the sources (local mode, AUTO mode, remote mode) to be used in the calculation

The same PROGRAM also resolves, on a preliminary basis, AUTO mode (Figure 6), selectable by the user: in AUTO mode, the mode changeover (from COOL to HEAT and vice versa) occurs without any action of the user on the basis of the value of a probe that can be selected by means of a parameter.

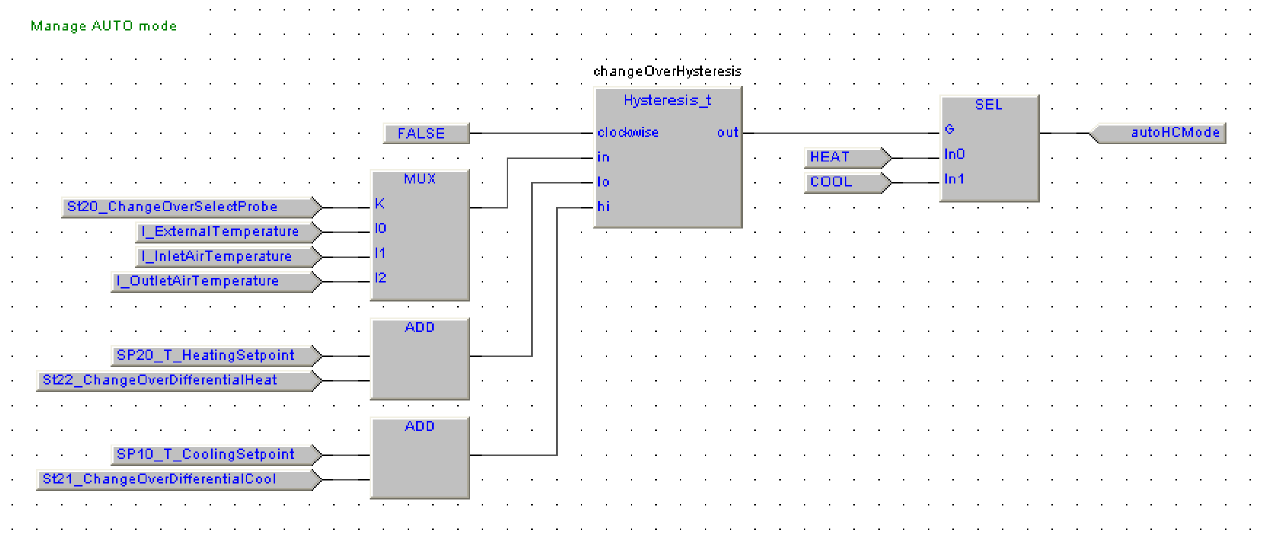


Figure 6: Resolution of AUTO mode: in this case the COOL/HEAT operating mode of the user is established by evaluating the value of the probe selected by means of user parameter St20 with reference to an interval of values (neutral zone) established by a combination of various user parameters (the lower limit is set by SP20 + St22, the upper limit by SP10 + St21)

5.4. Setpoints

The `Goal_Setpoint` PROGRAM calculates the setpoints relative to all the control goals managed by the application.

The result of `Goal_Setpoint` execution is encoded in the set of status variables dedicated to setpoints: this means that it is possible to edit or even completely replace the PROGRAM without affecting the rest of the application, as long as these status variables are assigned the correct values.

In AHU baselines the setpoint is first selected from a list of possible alternative sources and, thereafter, it is added to one or more differentials (see, for example, Figure 7).

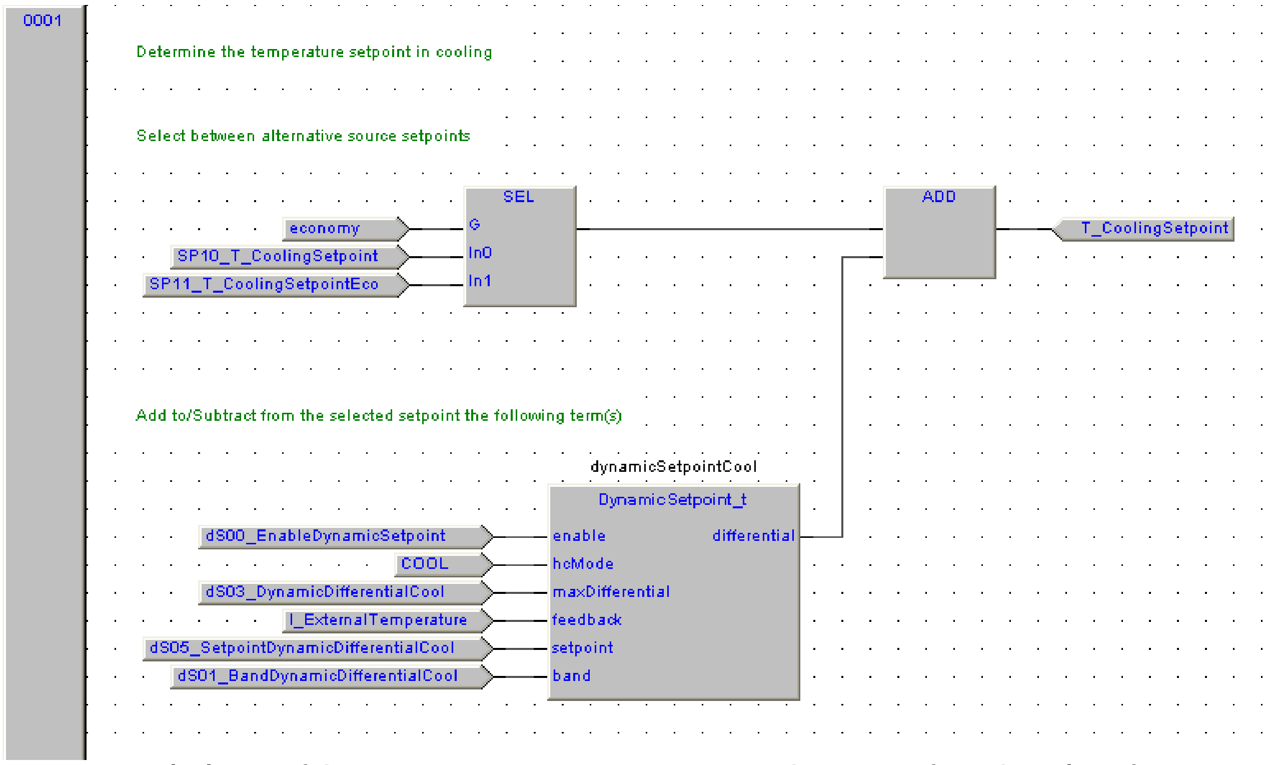

0001
Determine the temperature setpoint in cooling
Select between alternative source setpoints
SEL
economy
SP10_T_CoolingSetpoint
SP11_T_CoolingSetpointEco
G
In0
In1
ADD
T_CoolingSetpoint
Add to/Subtract from the selected setpoint the following term(s)
dynamicSetpointCool
DynamicSetpoint_t
dS00_EnableDynamicSetpoint
COOL
dS03_DynamicDifferentialCool
I_ExternalTemperature
dS05_SetpointDynamicDifferentialCool
dS01_BandDynamicDifferentialCool
enable
hcMode
maxDifferential
feedback
setpoint
band
differential

Figure 7: Calculation of the temperature setpoint in COOL mode, starting from the value of parameter SP10 or SP11, depending on whether or not Economy mode is active; the only differential applied is the dynamic differential on external temperature, if enabled by parameter (dS00)

5.5. Regulation probes

The `Goal_RegulationProbe` PROGRAM selects the probes to be used in the regulation stage (one probe for each control goal: temperature, relative humidity, etc.).

The result of the `Goal_RegulationProbe` execution is encoded in the set of status variables dedicated to the regulation probes: this means that it is possible to edit or even completely replace the PROGRAM without affecting the rest of the application, as long as these status variables are assigned the correct values.

In AHU baselines the value of one or more user parameters selects the probe to be used from a list (see, for example Figure 8).

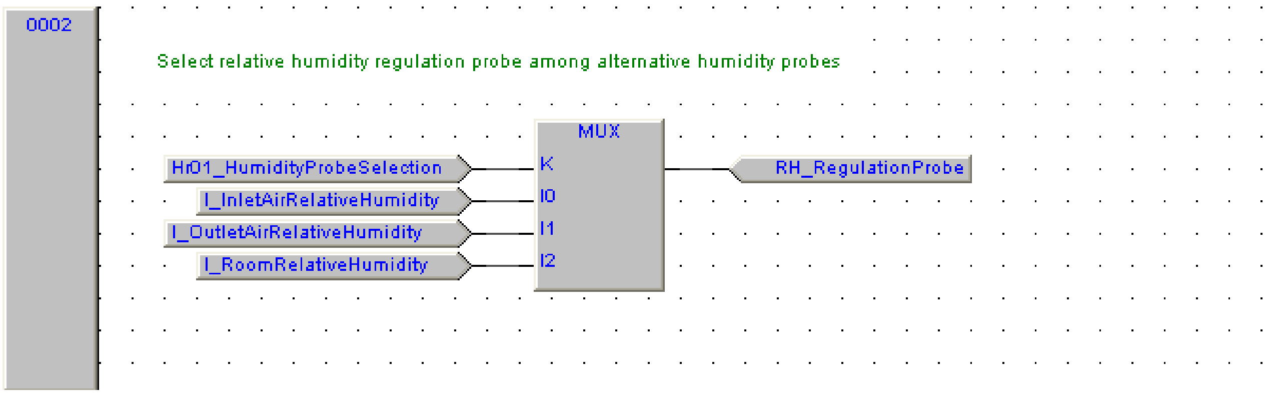

0002
Select relative humidity regulation probe among alternative humidity probes
MUX
Hr01_HumidityProbeSelection
I_InletAirRelativeHumidity
I_OutletAirRelativeHumidity
I_RoomRelativeHumidity
K
I0
I1
I2
RH_RegulationProbe

Figure 8: Selection of the relative humidity probe, chosen in accordance with the value of parameter Hr01 from the inlet probe, the outlet probe, and the room probe

5.5.1. Editing regulation probe selection

It may be necessary to add or remove a probe from the list of those selectable as the regulation probe.

EXAMPLE 5.9

For example, considering selection of the humidity regulation probe, assume we want to eliminate the possibility of using the outlet probe for regulation purposes.

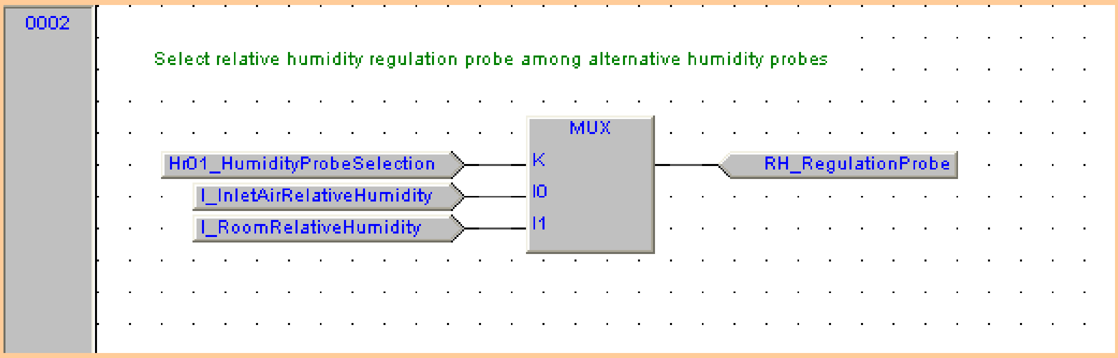
To achieve this result, proceed as follows:

- edit the corresponding FBD network in the `Goal_RegulationProbe` PROGRAM, so that the probe is included/excluded in the selection;
- edit the definition of the selection parameter, specifically to increase/decrease the maximum value.

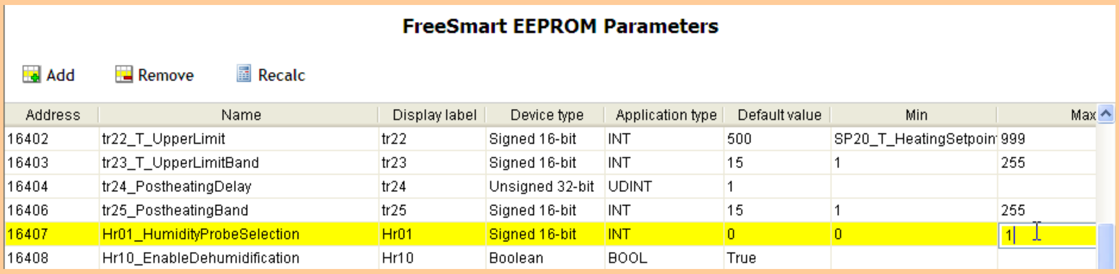
EXAMPLE 5.10

Returning to Example 5.9, we can proceed as follows:

1. Remove `I_OutletAirRelativeHumidity` from network number 2 of the `Goal_RegulationProbe` PROGRAM, decreasing the number of inputs of the `MUX` block and altering the connections appropriately;


0002
Select relative humidity regulation probe among alternative humidity probes
Hr01_HumidityProbeSelection
I_InletAirRelativeHumidity
I_RoomRelativeHumidity
MUX
K
I0
I1
RH_RegulationProbe

2. Edit the definition of the `Hr01_HumidityProbeSelection` parameter, altering the interval of values between 0 and 1.



Address	Name	Display label	Device type	Application type	Default value	Min	Max
16402	tr22_T_UpperLimit	tr22	Signed 16-bit	INT	500	SP20_T_HeatingSetpoin	999
16403	tr23_T_UpperLimitBand	tr23	Signed 16-bit	INT	15	1	255
16404	tr24_PostheatingDelay	tr24	Unsigned 32-bit	UDINT	1		
16406	tr25_PostheatingBand	tr25	Signed 16-bit	INT	15	1	255
16407	Hr01_HumidityProbeSelection	Hr01	Signed 16-bit	INT	0	0	1
16408	Hr10_EnableDehumidification	Hr10	Boolean	BOOL	True		

5.5.2. Setting the regulation probe

As the limit case of the requirement expressed in heading 5.5.1., it may be required to establish the regulation probe, releasing it from the selection parameter, which can therefore be removed.

EXAMPLE 5.11

Taking Example 5.9 to the extreme case, assume we decide that the relative humidity regulation probe must be the inlet probe.

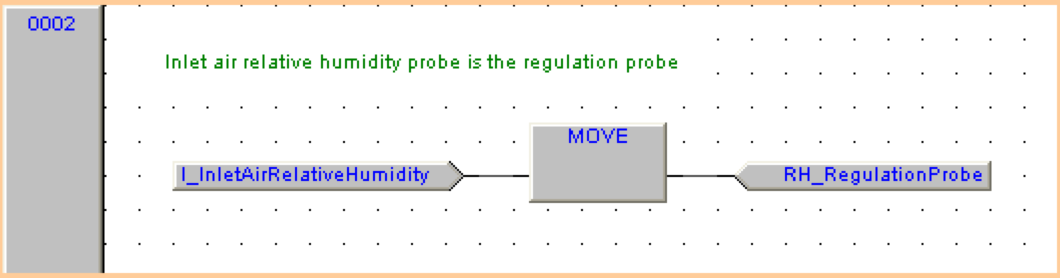
To achieve this result, proceed as follows:

- simplify the corresponding FBD network in the Goal_RegulationProbe PROGRAM, which becomes a simple assignment;
- remove the selection parameter.

EXAMPLE 5.12

To achieve the goal established in Example 5.11, we can proceed as follows:

1. remove the MUX block, Hr01_HumidityProbeSelection, I_OutletAirRelativeHumidity and I_RoomRelativeHumidity from network number 2 of the Goal_RegulationProbe PROGRAM;
2. insert a MOVE block, which connects I_InletAirRelativeHumidity and RH_RegulationProbe;


0002
Inlet air relative humidity probe is the regulation probe
MOVE
I_InletAirRelativeHumidity
RH_RegulationProbe

3. in the *Resources > EEPROM Parameters* section, delete parameter Hr01_HumidityProbeSelection.

5.6. Diagnostics

The Diagnostics PROGRAM establishes the alarms activation status, i.e. the status of all the variables defined in the *Resources > Alarms* section.

The Diagnostics PROGRAM first assesses whether or not to enable diagnostics: otherwise, all the alarms are deactivated (see Figure 9).

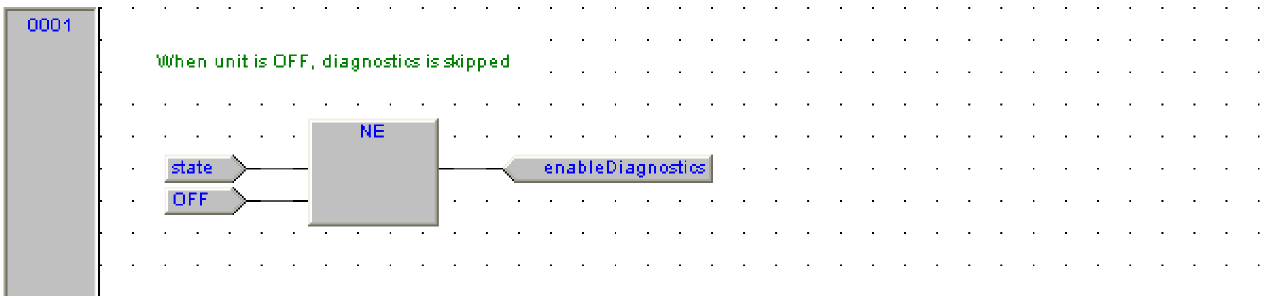


Figure 9: If the AHU status is OFF, diagnostics is disabled

Subsequently (Figure 10), the eventual request for resetting of the alarms received via the user interface, is managed.

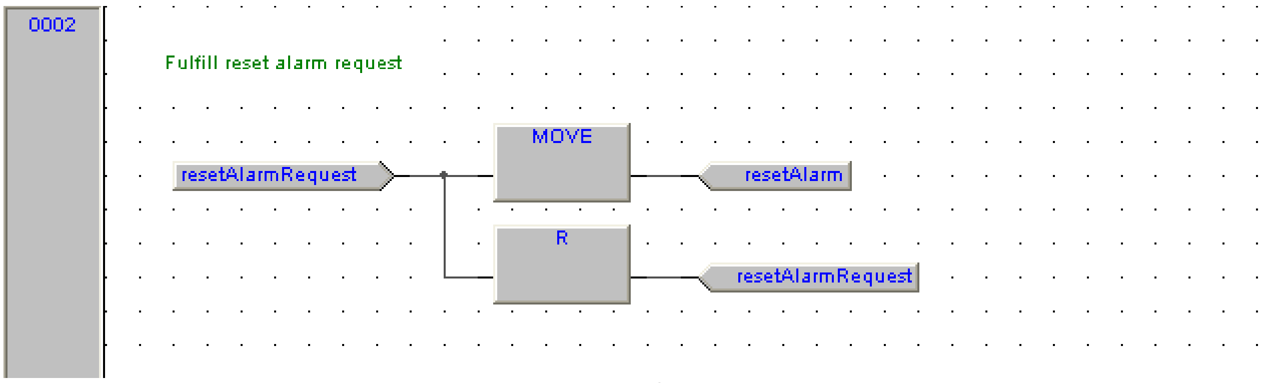


Figure 10: Management of the alarms reset request

Finally (Figure 11), all the alarm conditions monitored by the application are analyzed one at a time.

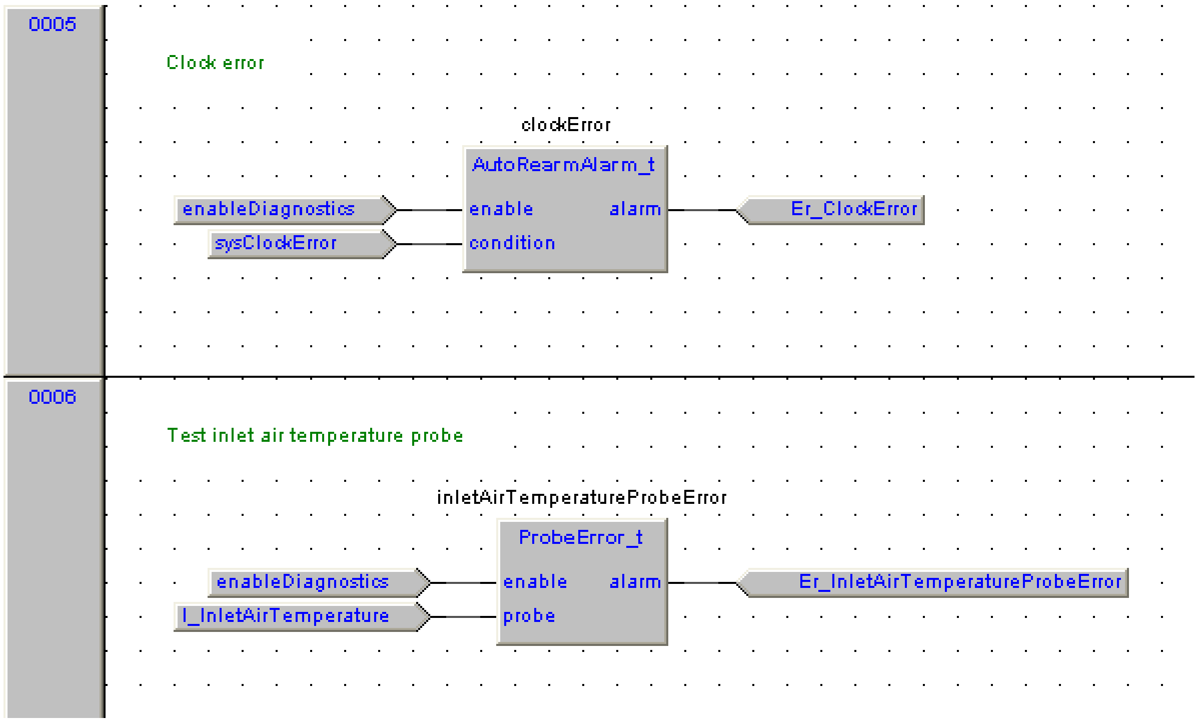


Figure 11: `Er_ClockError` encodes clock error status: this is a manually reset alarm (i.e. it requires an explicit reset by the user) and it monitors the status of `sysClockError` system variables. `Er_InletAirTemperatureProbeError` encodes the inlet air temperature probe error status

If the logics that determine the value of one or more alarms already provided in the baselines are edited, it is not necessary to edit the parts of the application that use said alarms.

However, the most frequent modifications of the `Diagnostics` PROGRAM are associated with the addition or removal of an alarm, which can impact on the alarm management delegated to the downstream parts of the application (in particular, to the `Strategy` PROGRAM).

5.6.1. Adding an alarm

If a derived application envisages signalling/management of an alarm condition that is not provided for in the AHU baselines, an alarm must be added, its signalling must be managed and the any reactions of the controller must be implemented.

EXAMPLE 5.13

Assume, for example, we want to add the management of a fire alarm read on a digital input connected to a smoke detector.

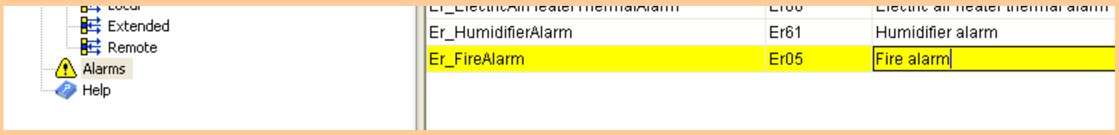
To manage a new alarm condition, the following procedure is necessary:

- create a new alarm variable in the section *Resources > Alarms*;
- manage, in a new FBD network of the `Diagnostics` program, activation and reset of the alarm that has just been added;
- if so required, manage the active alarm condition in the `Strategy` PROGRAM with an adequate reaction/safety measures procedure.

EXAMPLE 5.14

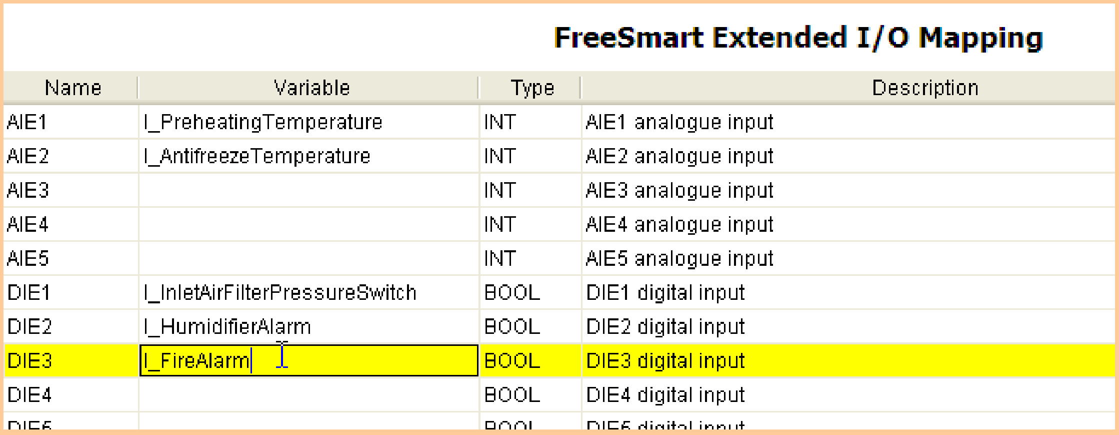
Returning to Example 5.14, the requirement can be fulfilled with the following modifications:

1. in the *Resources > Alarms* section, add a new alarm variable, `Er_FireAlarm`;



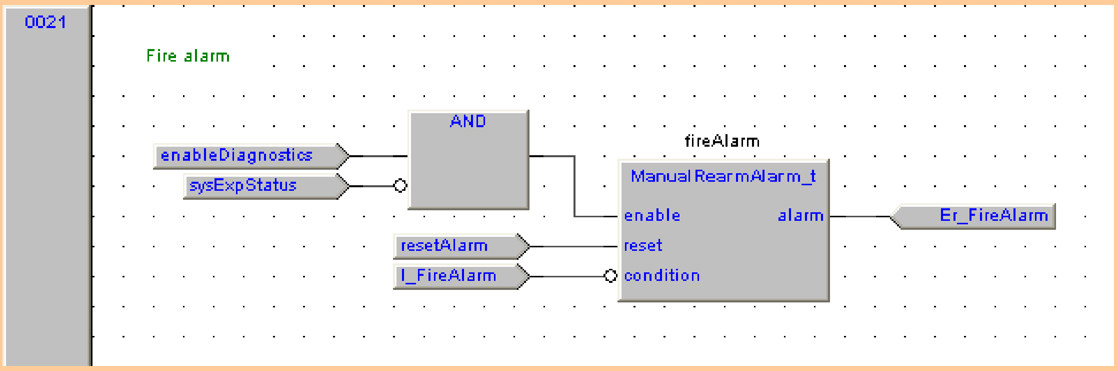
Er_HumidifierAlarm	Er61	Humidifier alarm
Er_FireAlarm	Er05	Fire alarm

2. in the *Resources > I/O Mapping* section, add the diagnostic digital input `I_FireAlarm`, corresponding to the smoke detector;

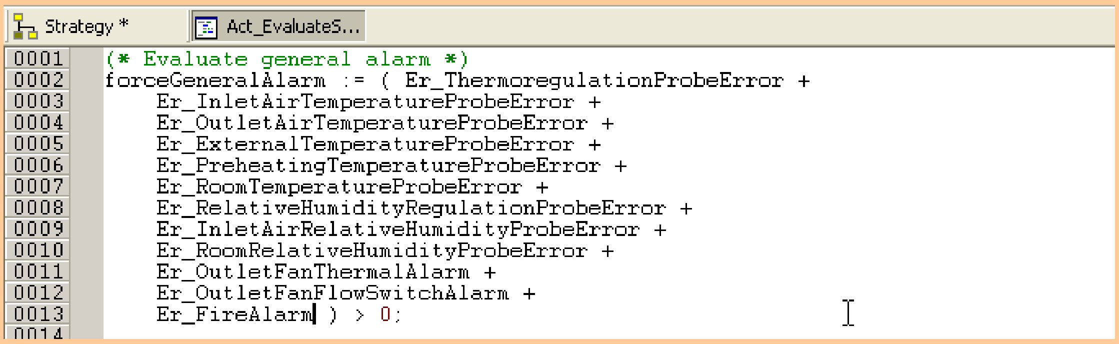

FreeSmart Extended I/O Mapping

Name	Variable	Type	Description
AIE1	I_PreheatingTemperature	INT	AIE1 analogue input
AIE2	I_AntifreezeTemperature	INT	AIE2 analogue input
AIE3		INT	AIE3 analogue input
AIE4		INT	AIE4 analogue input
AIE5		INT	AIE5 analogue input
DIE1	I_InletAirFilterPressureSwitch	BOOL	DIE1 digital input
DIE2	I_HumidifierAlarm	BOOL	DIE2 digital input
DIE3	I_FireAlarm	BOOL	DIE3 digital input
DIE4		BOOL	DIE4 digital input
DIE5		BOOL	DIE5 digital input

3. encode activation and resetting of the alarm with a simple FBD network and the assistance of the blocks available in the Alarms library;



4. in `Strategy > Act_EvaluateSecurities`, add `Er_FireAlarm` among the alarms that determine implementation of air handling unit safety measures.



```
(* Evaluate general alarm *)
forceGeneralAlarm := ( Er_ThermoregulationProbeError +
    Er_InletAirTemperatureProbeError +
    Er_OutletAirTemperatureProbeError +
    Er_ExternalTemperatureProbeError +
    Er_PreheatingTemperatureProbeError +
    Er_RoomTemperatureProbeError +
    Er_RelativeHumidityRegulationProbeError +
    Er_InletAirRelativeHumidityProbeError +
    Er_RoomRelativeHumidityProbeError +
    Er_OutletFanThermalAlarm +
    Er_OutletFanFlowSwitchAlarm +
    Er_FireAlarm ) > 0;
```

5.7. Strategy

The `Strategy` PROGRAM defines the state machine of the application.

The principal states (Figure 12) are:

- ON (normal operation);
- OFF (unit off);
- pre-alarm or emergency states:
  - OFF with alarm (emergency condition);
  - antifreeze.

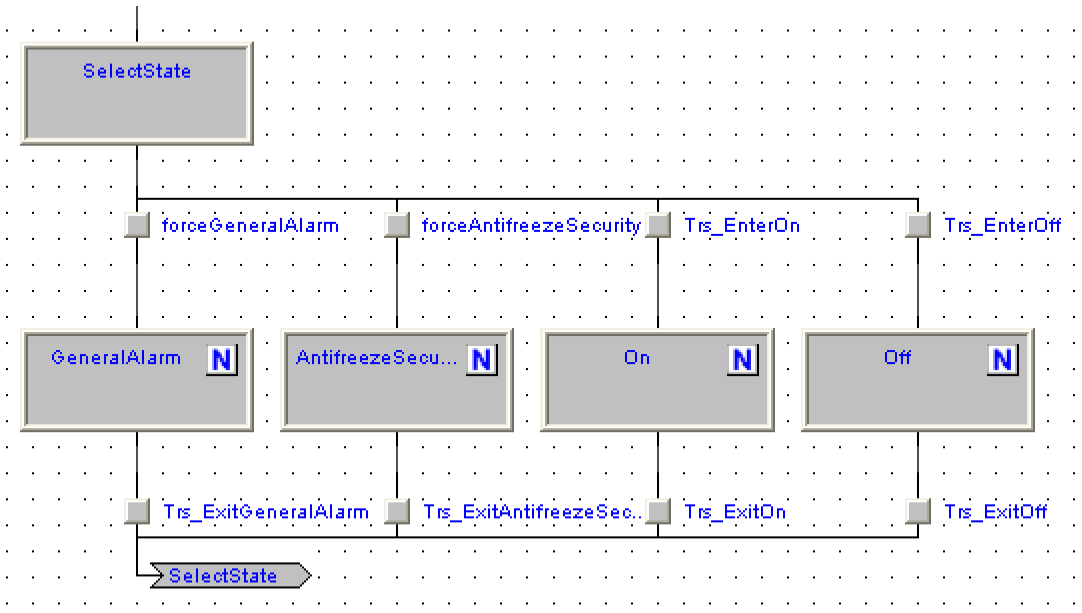


Figure 12: Principal states of the AHU baselines: in addition to normal operating status (ON) and OFF status, there is also an alarm status and an extraordinary procedure for the antifreeze regime

The ON status is, in turn, a states machine (Figure 13), wherein each state represents a strategy to fulfill a goal (temperature, humidity, etc.).

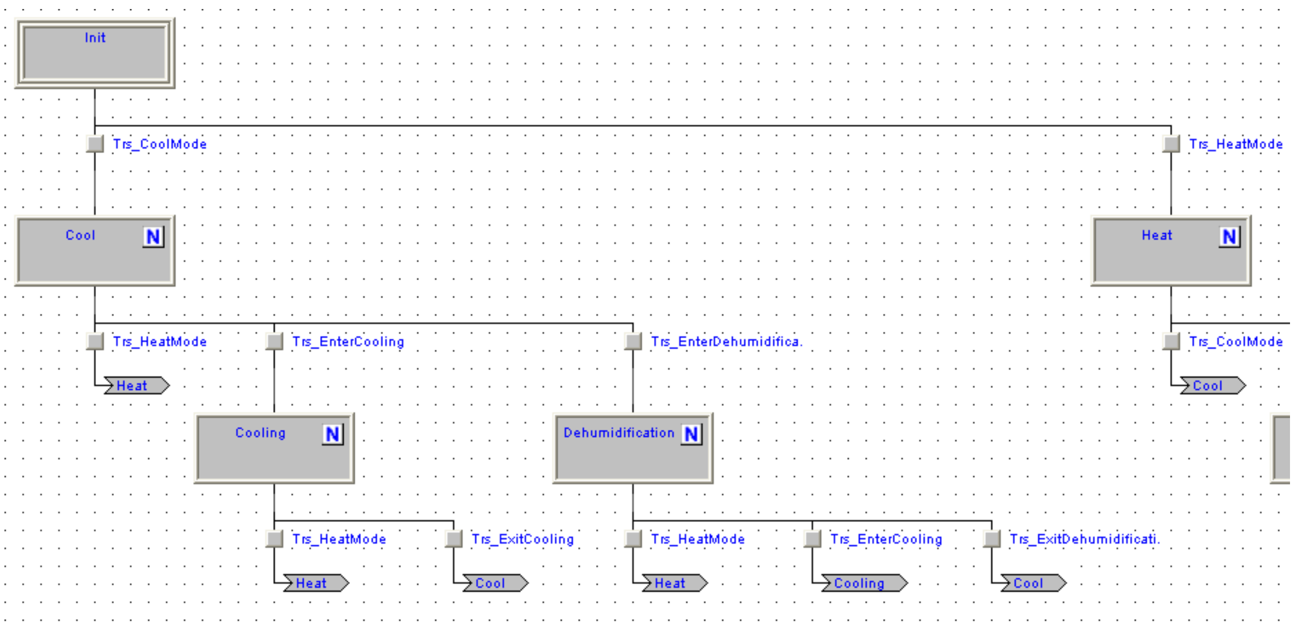


Figure 13: states machine of layout 2, during normal operation of the unit: considering, for example, summer mode (COOL), the unit can activate the procedures for cooling (STEP Cooling), priority, and dehumidification (STEP Dehumidification)

5.7.1. Removing a strategy

It may occur that some of the pre-alarm/emergency situations envisaged by the Air Handling Unit baselines, or the strategies to fulfill the control aims, are not relevant for our particular application case: it may therefore be advisable to eliminate them.

EXAMPLE 5.15

For example, consider an application derived from layout 2, which differs from the latter due to the absence of a humidifier and hence the impossibility of performing the dehumidification strategy.

To do this, proceed as follows:

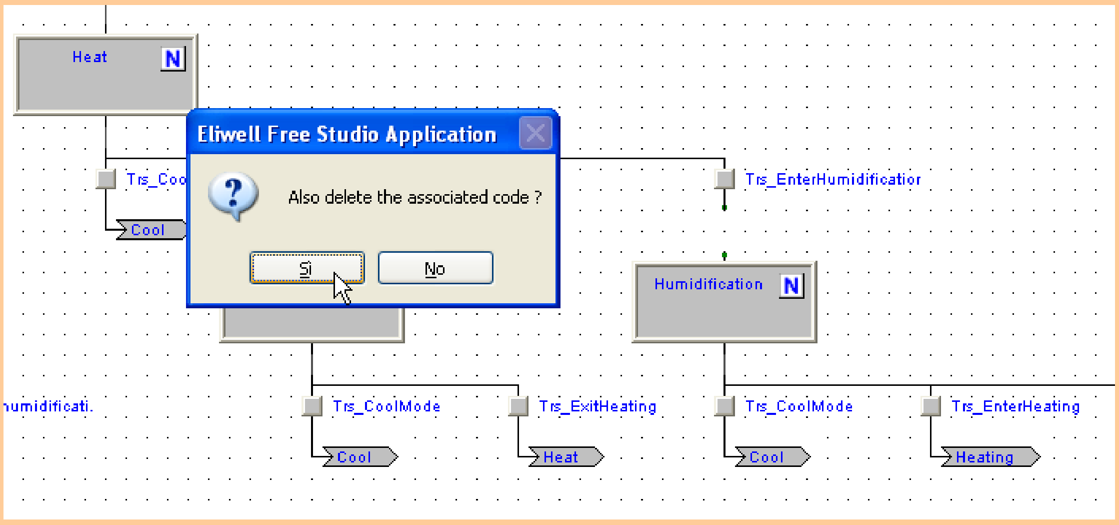
- remove the SFC STEP corresponding to the status to be eliminated;
- if it is not used in other STEPS, remove the ACTION corresponding to the strategy to be eliminated;
- adjust the connections of the SFC diagram, removing the transitions to and from the eliminated state.

It may be necessary to adjust other parts of the `Strategy` PROGRAM or the application, for example, to eliminate an alarm or an AHU component that is no longer required.

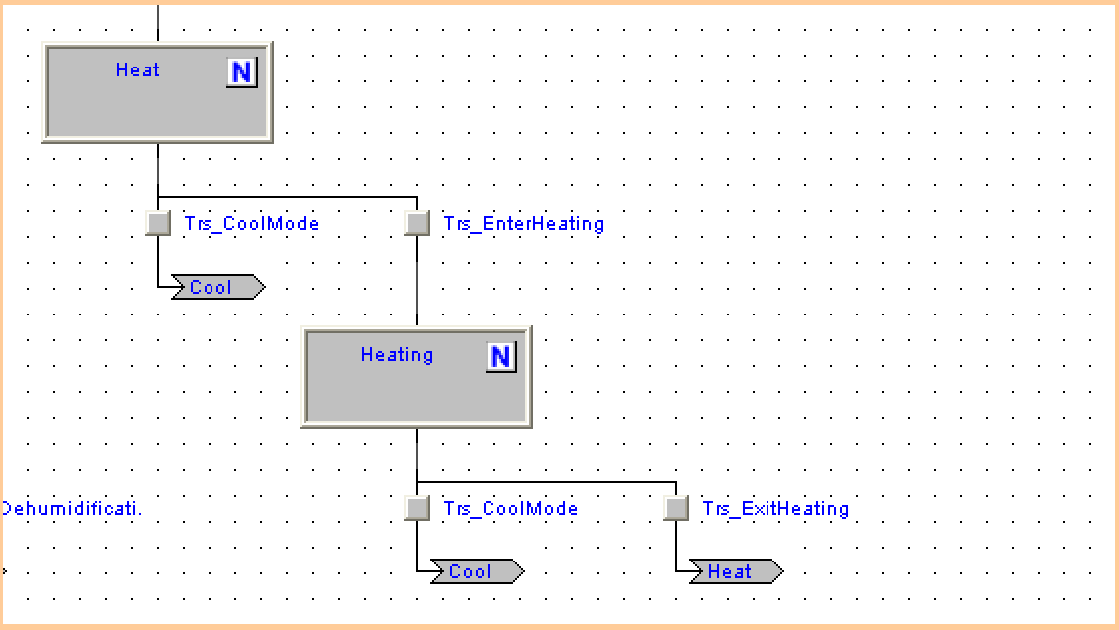
EXAMPLE 5.16

Returning to Example 5.15, it is sufficient:

1. to eliminate the `Humidification` STEP and the associated code (`Act_Humidification`) in the `Act_On` ACTION;



2. to eliminate the exit transitions from the eliminated STEP and the third pin of the exit transition from the `Heat` STEP (entering in the eliminated STEP);



3. to eliminate the `Humidifier_AHU02` FUNCTION_BLOCK, its global instance `humidifier` and the FBD networks in the `Strategy` PROGRAM (one per strategy) dedicated to its management.

5.7.2. Adding a strategy

Derived applications can manage pre-alarm or emergency situations that are not envisaged by the Air Handling Unit baselines and also provide additional strategies to fulfill the control objectives.

EXAMPLE 5.17

Returning to Example 5.13, assume we wish to react to a fire alarm with a dedicated procedure.

To do this, proceed as follows:

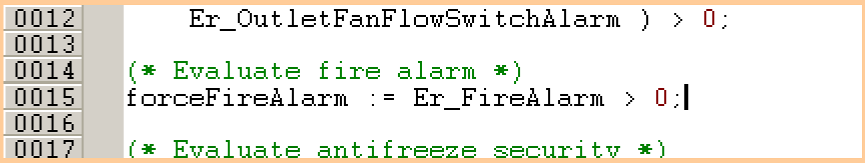
- create the SFC STEP corresponding to the state to be added;
- create the ACTION corresponding to the strategy to be added, assigning it to the STEP we have just created;
- adjust the connections of the SFC diagram, adding the transitions to and from the added state.

It may be necessary to adjust other parts of the `Strategy` PROGRAM or the application, for example, to manage a new alarm variable or a new AHU component.

EXAMPLE 5.18

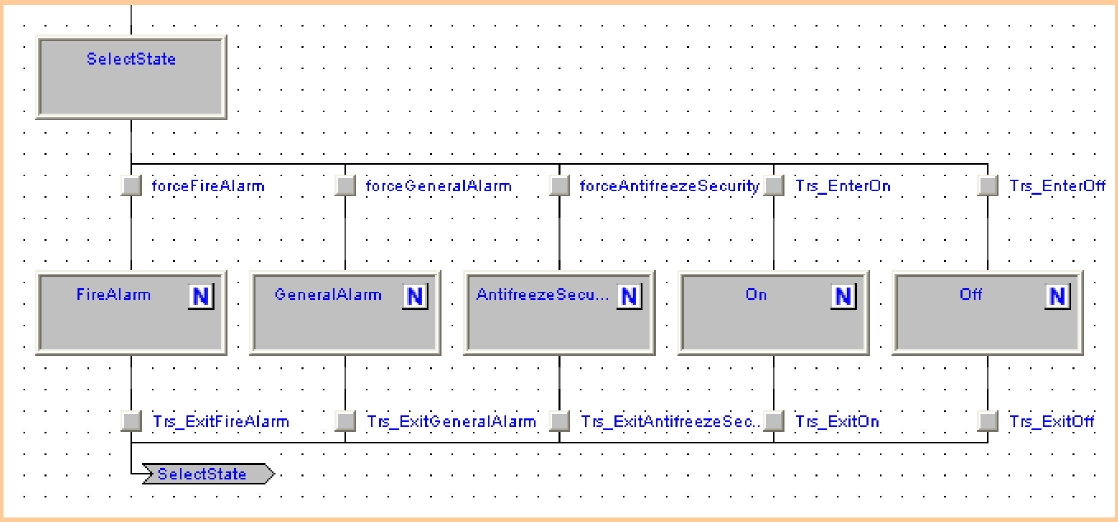
To meet the requirement expressed in Example 5.17, proceed as follows:

1. edit `Act_EvaluateSecurities` so that it assigns a value to a flag dedicated to the fire alarm, `forceFireAlarm`;

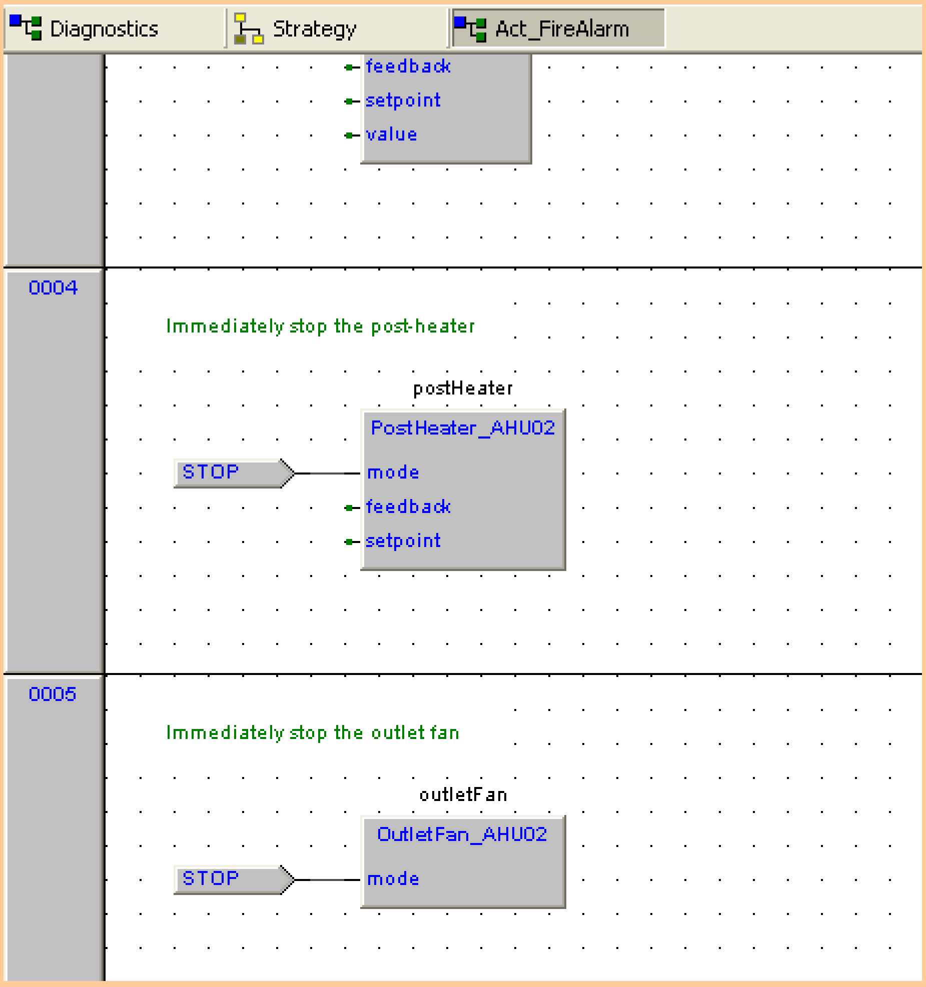


```
    Er_OutletFanFlowSwitchAlarm ) > 0;

(* Evaluate fire alarm *)
forceFireAlarm := Er_FireAlarm > 0;

(* Evaluate antifreeze security *)
```

2. insert a new STEP, `FireAlarm`, connected in parallel to the other main states, but in the first position, because it has higher priority;
3. as a condition of the entry transition to the `FireAlarm` STEP, select `forceFireAlarm`; as an exit condition, select a new TRANSITION, corresponding to the denied value of `forceFireAlarm`;
4. create a new ACTION `Act_FireAlarm` and assign it to the `FireAlarm` STEP;



5. encode the new strategy in the `Act_FireAlarm` ACTION, for example in FBD language, wherein each network is dedicated to controlling a component of the air handling unit.



5.8. Regulation and actuation of outputs

The components of the AHU are modelled within the AHU baselines with the FUNCTION_BLOCKS utilized by the `Strategy` PROGRAM in the execution of the various strategies.

5.8.1. Editing the regulation of an AHU component.

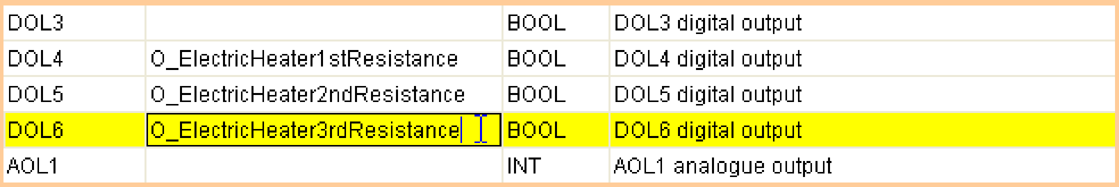
In developing an application derived from the AHU baselines it is frequently necessary to modify the regulation and/or actuation of the controller outputs in order to mirror the differences in the physical actuators of which the unit is composed. Provided the FUNCTION_BLOCK interface remains unchanged, it will not be necessary to alter the rest of the application.

EXAMPLE 5.19

For example, consider an application derived from layout 1, from which it differs due to the presence of an electric heater with resistances (in the number of 3) in place of the heating coil with modulating valve.

We can proceed as follows:

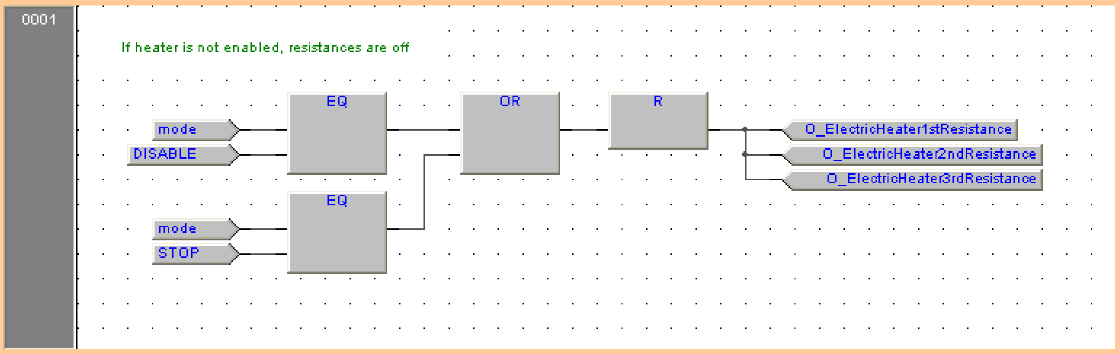
1. edit the application outputs: the baseline uses an analog output to regulate the electric heater, while the derivative application uses three digital outputs (one for each resistance);



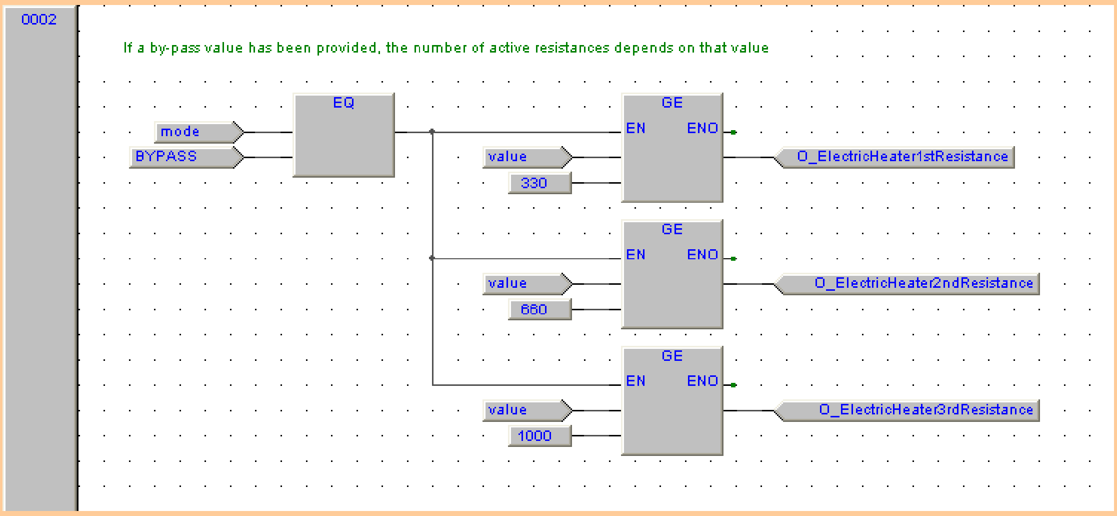
DOL3		BOOL	DOL3 digital output
DOL4	O_ElectricHeater1stResistance	BOOL	DOL4 digital output
DOL5	O_ElectricHeater2ndResistance	BOOL	DOL5 digital output
DOL6	O_ElectricHeater3rdResistance	BOOL	DOL6 digital output
AOL1		INT	AOL1 analogue output

2. change implementation of the `Heater_AHU01` block, so that it translates the same commands received from STRATEGY into actions on three digital outputs; specifically:

   a) when the electric heater is not enabled, the three digital outputs are `FALSE`;


0001
If heater is not enabled, resistances are off
mode
DISABLE
EQ
mode
STOP
EQ
OR
R
O_ElectricHeater1stResistance
O_ElectricHeater2ndResistance
O_ElectricHeater3rdResistance

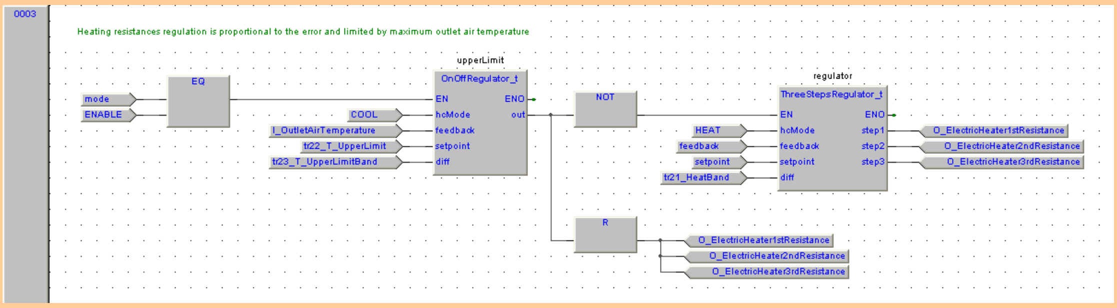
   b) when the power level of the electric heater is set, if the level is 33%, only the digital output corresponding to the first resistance is `TRUE`; if the level is 66%, two digital outputs are `TRUE`; if the level is 100%, all three digital outputs are `TRUE`;



If a by-pass value has been provided, the number of active resistances depends on that value

0002
mode
BYPASS
EQ
GE
EN
ENO
value
330
O_ElectricHeater1stResistance
GE
EN
ENO
value
660
O_ElectricHeater2ndResistance
GE
EN
ENO
value
1000
O_ElectricHeater3rdResistance

c) in normal operation of the heating coil, regulation is performed in three steps, the differentials of which correspond to one third of the thermoregulation band in HEAT mode (you can use the `ThreeStepsRegulator_t` block of the Regulators library); actuation of the outputs is disabled if the outlet temperature exceeds the upper limit set by parameter.



Heating resistances regulation is proportional to the error and limited by maximum outlet air temperature

5.8.2. Adding a component to the AHU

In developing a dedicated application derived from a baseline, it may be necessary to manage a physical component of the AHU that is not envisaged by the baseline.

EXAMPLE 5.20

Consider the case of an application derived from layout 1, from which it differs due to the presence of a heat recovery unit.

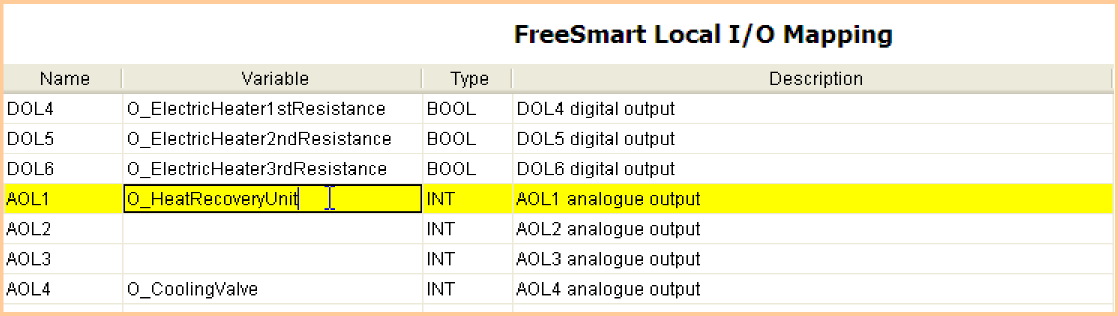
Management of the new component calls for:

- definition of an additional FUNCTION_BLOCK that is responsible for translating the commands received from STRATEGY into operations on the outputs;
- editing of the `Strategy` PROGRAM so that the new component is managed in all operating modes (in each "strategy").

EXAMPLE 5.21

Considering Example 5.20, we can proceed as follows:

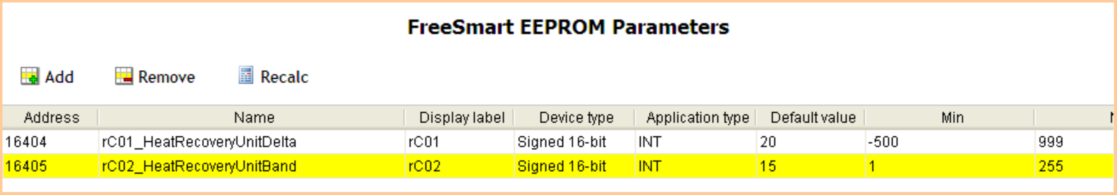
1. define the logical output to control the heat recovery unit, which it is assumed to be of the rotary type and that requires the controller to specify the speed (analog output);



FreeSmart Local I/O Mapping

Name	Variable	Type	Description
DOL4	O_ElectricHeater1stResistance	BOOL	DOL4 digital output
DOL5	O_ElectricHeater2ndResistance	BOOL	DOL5 digital output
DOL6	O_ElectricHeater3rdResistance	BOOL	DOL6 digital output
AOL1	O_HeatRecoveryUnit	INT	AOL1 analogue output
AOL2		INT	AOL2 analogue output
AOL3		INT	AOL3 analogue output
AOL4	O_CoolingValve	INT	AOL4 analogue output

2. define the parameters required for regulation of the rotary heat recovery unit, which, in this example, we assume to be limited to:
   a) a setpoint on the difference between the expulsion temperature and ambient air temperature;
   b) the proportional band beyond which the regulator saturates (speed equal to 100%);



FreeSmart EEPROM Parameters

Add Remove Recalc

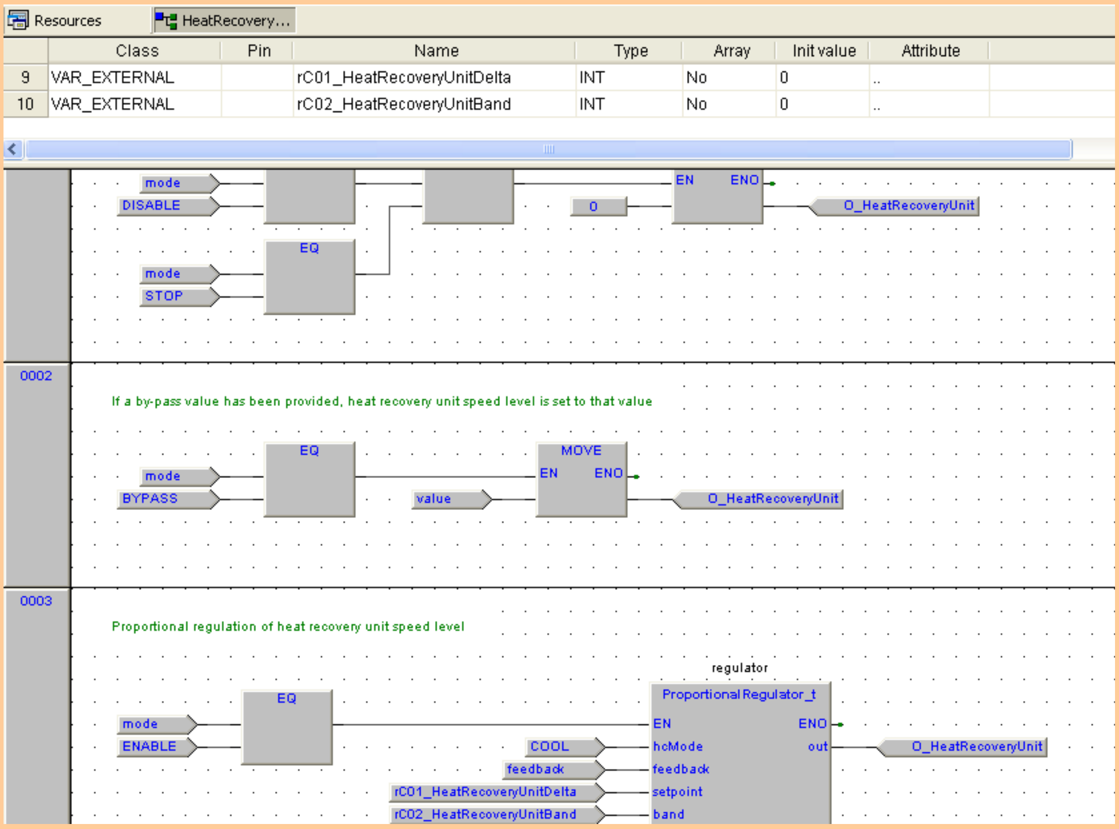
Address	Name	Display label	Device type	Application type	Default value	Min	
16404	rC01_HeatRecoveryUnitDelta	rC01	Signed 16-bit	INT	20	-500	999
16405	rC02_HeatRecoveryUnitBand	rC02	Signed 16-bit	INT	15	1	255

3. create the FUNCTION_BLOCK dedicated to control of the heat recovery unit in order to manage the following operating modes:

   a) disabled: in this case the rotary heat recovery unit speed is null, and the `I_HeatRecoveryUnit` output is set to 0;

   b) enabled: the speed of the rotary heat recovery unit is subject to proportional regulation controlled by parameters `rC01` and `rC02`;

   c) fixed speed: the speed of the rotary heat recovery unit is fixed at a level specified by STRATEGY;



Resources | HeatRecovery...

	Class	Pin	Name	Type	Array	Init value	Attribute
9	VAR_EXTERNAL		rC01_HeatRecoveryUnitDelta	INT	No	0	..
10	VAR_EXTERNAL		rC02_HeatRecoveryUnitBand	INT	No	0	..

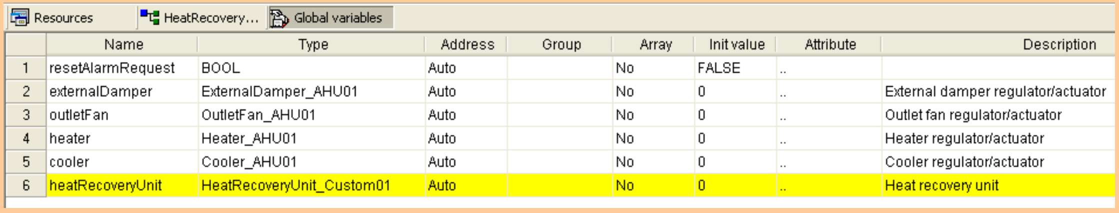
0002

If a by-pass value has been provided, heat recovery unit speed level is set to that value

0003

Proportional regulation of heat recovery unit speed level

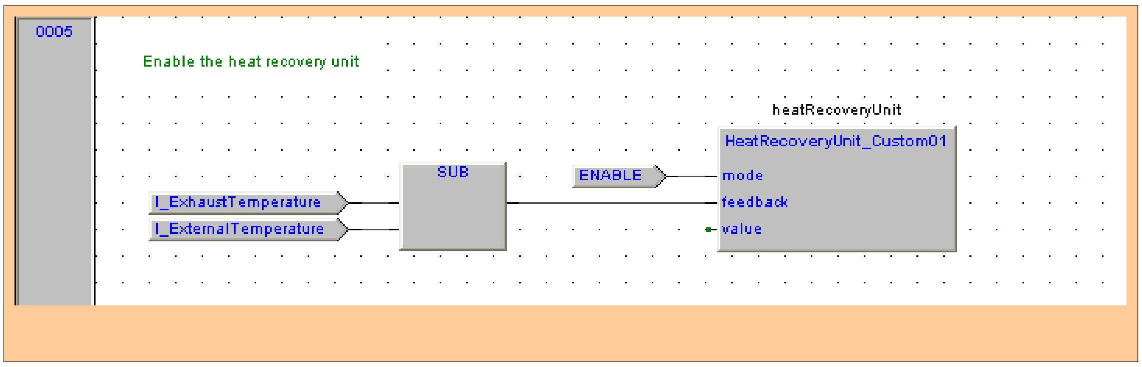
4. create a global instance of the FUNCTION_BLOCK;



Resources | HeatRecovery... | Global variables

	Name	Type	Address	Group	Array	Init value	Attribute	Description
1	resetAlarmRequest	BOOL	Auto		No	FALSE	..	
2	externalDamper	ExternalDamper_AHU01	Auto		No	0	..	External damper regulator/actuator
3	outletFan	OutletFan_AHU01	Auto		No	0	..	Outlet fan regulator/actuator
4	heater	Heater_AHU01	Auto		No	0	..	Heater regulator/actuator
5	cooler	Cooler_AHU01	Auto		No	0	..	Cooler regulator/actuator
6	heatRecoveryUnit	HeatRecoveryUnit_Custom01	Auto		No	0	..	Heat recovery unit

5. edit the `Strategy` PROGRAM, inserting - in each strategy - a new FBD network dedicated to control of the heat recovery unit.



5.9. LCD terminal menu

The `SKWHMI` PROGRAM is responsible for defining the user menu for the LCD terminal.

The menu tree is represented by an analogous construct in SFC language (see Figure 14), wherein each STEP represents a submenu and each TRANSITION represents the possible transitions between them.

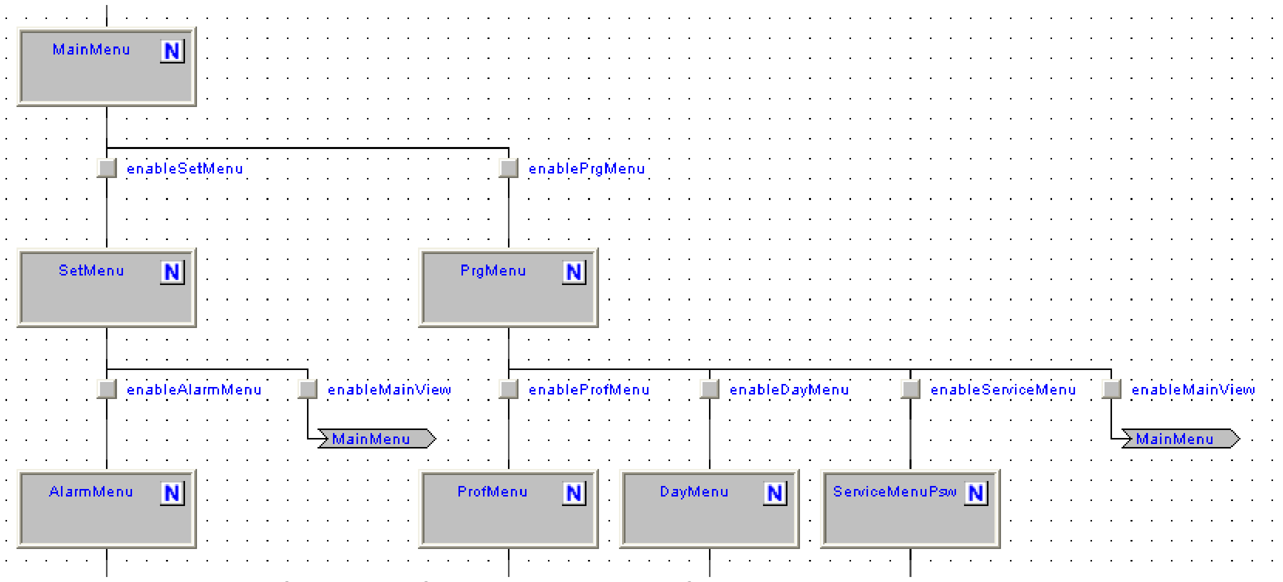


Figure 14: Structure of the menu for the LCD terminal: from the main menu you can access the set menu to edit the setpoints and display active alarms, or the Prg menu, which in turn allows access to various submenus for parameter programming procedures

Each transition is associated with a Boolean variable that represents the enabling of a single submenu (see Figure 15 for example): these variables are assigned values in the ACTIONS associated with the STEPS of the diagram and, at each instant, only one of them must be `TRUE`.

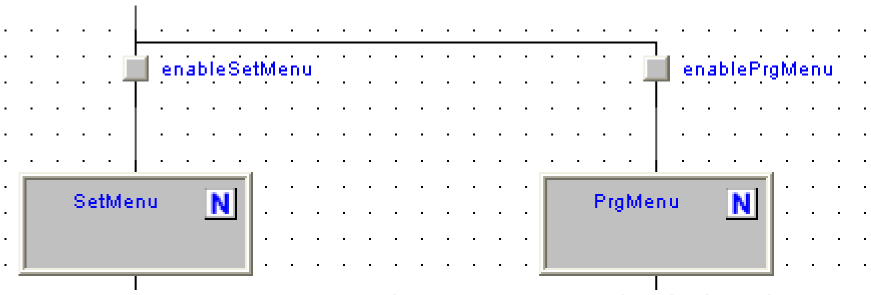
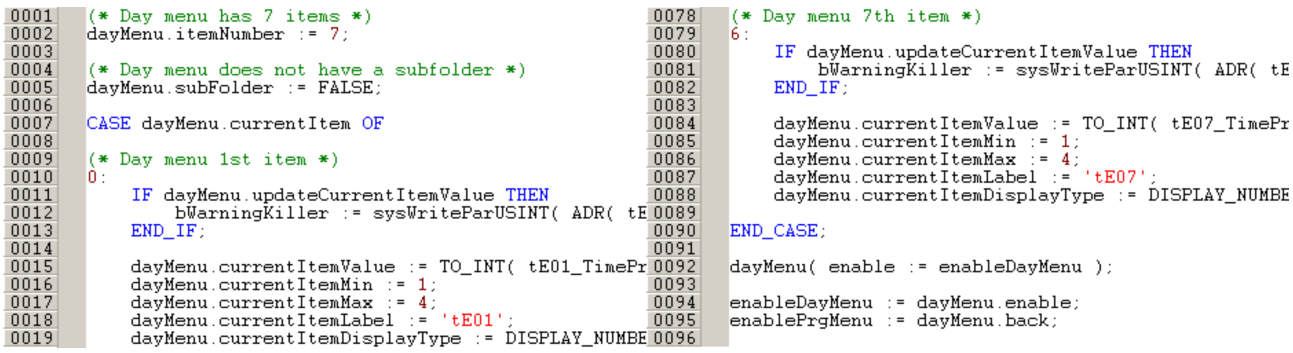


Figure 15: transition in the set menu is enabled when the **`enableSetMenu`** variable is **`TRUE`**; transition in the Prg menu is enabled when the **`enablePrgMenu`** variable is **`TRUE`**

The ACTIONS of the `SKWHMI` PROGRAM define the submenus and their items, utilizing the blocks of the SmartHMI library for management of user navigation (Figure 16).



```
(* Day menu has 7 items *)
dayMenu.itemNumber := 7;

(* Day menu does not have a subfolder *)
dayMenu.subFolder := FALSE;

CASE dayMenu.currentItem OF

(* Day menu 1st item *)
0:
    IF dayMenu.updateCurrentItemValue THEN
        bWarningKiller := sysWriteParUSINT( ADR( tE
    END_IF;

    dayMenu.currentItemValue := TO_INT( tE01_TimePr
    dayMenu.currentItemMin := 1;
    dayMenu.currentItemMax := 4;
    dayMenu.currentItemLabel := 'tE01';
    dayMenu.currentItemDisplayType := DISPLAY_NUMBE
```

```
(* Day menu 7th item *)
6:
    IF dayMenu.updateCurrentItemValue THEN
        bWarningKiller := sysWriteParUSINT( ADR( tE
    END_IF;

    dayMenu.currentItemValue := TO_INT( tE07_TimePr
    dayMenu.currentItemMin := 1;
    dayMenu.currentItemMax := 4;
    dayMenu.currentItemLabel := 'tE07';
    dayMenu.currentItemDisplayType := DISPLAY_NUMBE

END_CASE;

dayMenu( enable := enableDayMenu );

enableDayMenu := dayMenu.enable;
enablePrgMenu := dayMenu.back;
```

Figure 16: two fragments of **`Act_DayMenu`**, containing the definition of the Day menu and of its 7 items, invocation of the **`SKWRightDisplayMenu_t`** library block (**`dayMenu`** variable) and management of the enabling flags of the menu itself and the parent memory (Prg menu)

5.9.1. Adding an item to a menu

A frequent variation to the menu for the LCD terminal included in the AHU baselines, is the addition of an item to a menu.

EXAMPLE 5.22

Returning to Example 5.21, assume we want to insert parameter `rC01` in the password protected `ThermoregulationMenu`.

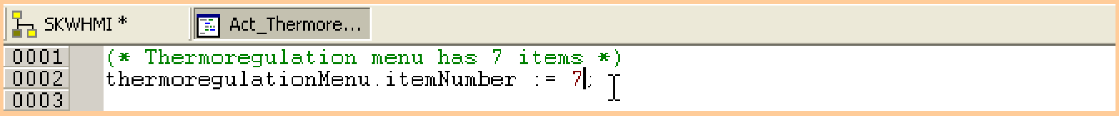
In this case the changes are limited to the ACTION that defines the menu to be edited. In particular, it is necessary to:

- edit the definition of the menu, increasing the number of items;
- add the definition of the new item.

EXAMPLE 5.23

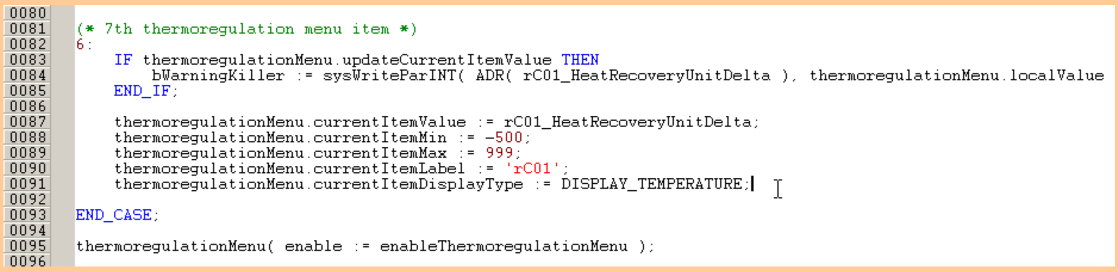
To achieve the goal established in Example 5.22, we can proceed as follows:

1. increase the value assigned to `thermoregulationMenu.itemNumber`;



```
(* Thermoregulation menu has 7 items *)
thermoregulationMenu.itemNumber := 7;
```

2. insert the definition of `rC01` in the desired position, specifying the minimum and maximum values, the text identification, and the type of value to be displayed (`DISPLAY_TEMPERATURE`).



```

(* 7th thermoregulation menu item *)
6:
    IF thermoregulationMenu.updateCurrentItemValue THEN
        bWarningKiller := sysWriteParINT( ADR( rC01_HeatRecoveryUnitDelta ), thermoregulationMenu.localValue
    END_IF;

    thermoregulationMenu.currentItemValue := rC01_HeatRecoveryUnitDelta;
    thermoregulationMenu.currentItemMin := -500;
    thermoregulationMenu.currentItemMax := 999;
    thermoregulationMenu.currentItemLabel := 'rC01';
    thermoregulationMenu.currentItemDisplayType := DISPLAY_TEMPERATURE;

END_CASE;

thermoregulationMenu( enable := enableThermoregulationMenu );
```

5.9.2. Adding a menu

The user interface for the LCD terminal of an application derived from the AHU baselines may call for the addition of an entire menu.

EXAMPLE 5.24

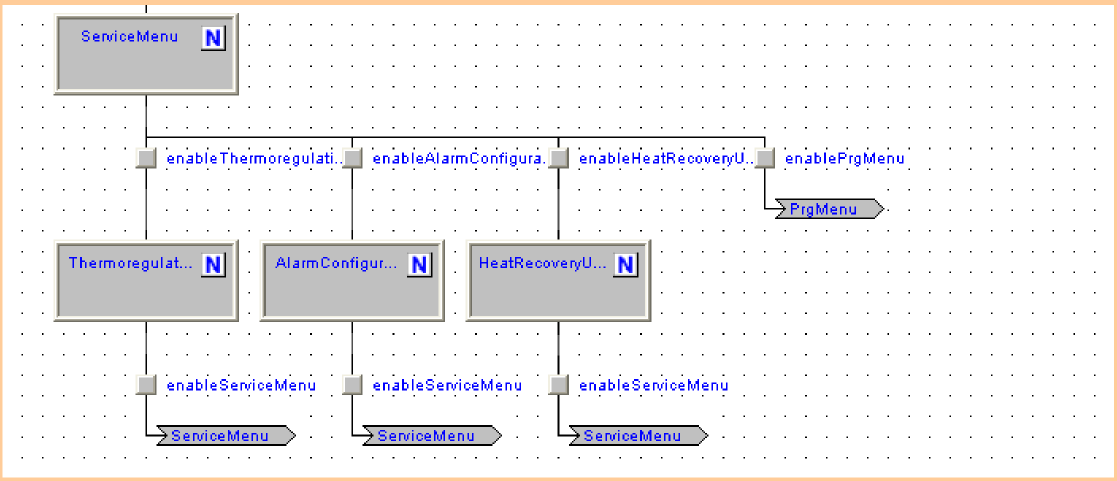
Extending Example 5.22, assume we wish to create a menu dedicated to the configuration of the heat recovery unit (parameters `rC01` and `rC02`), within the password-protected section.

To add a menu we need to adjust the SFC diagram to add the STEP corresponding to the new menu and connect it to the parent menu.

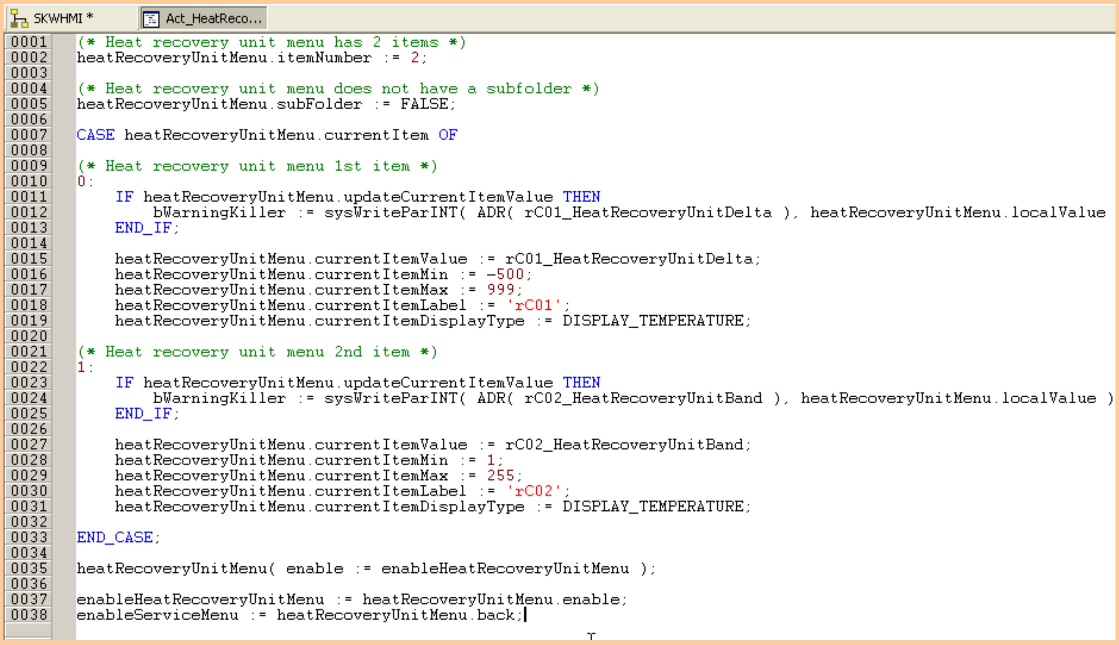
EXAMPLE 5.25

To meet the requirement expressed in Example 5.25, proceed as follows:

1. first of all define a new Boolean variable to enable the menu, `enableHeatRecoveryUnitMenu`;
2. create a new STEP, `HeatRecoveryUnitMenu`, connecting it as the child of `ServiceMenu`;



3. edit `Act_ServiceMenu` so that it manages an extra item corresponding to the new submenu;
4. create and implement the ACTION `Act_HeatRecoveryUnitMenu`, inserting the definition of the menu and its items (parameters `rC01` and `rC02`); in doing this, it may prove useful to create and invoke a suitable block of the SmartHMI library for navigation management.



```
(* Heat recovery unit menu has 2 items *)
heatRecoveryUnitMenu.itemNumber := 2;

(* Heat recovery unit menu does not have a subfolder *)
heatRecoveryUnitMenu.subFolder := FALSE;

CASE heatRecoveryUnitMenu.currentItem OF

(* Heat recovery unit menu 1st item *)
0:
    IF heatRecoveryUnitMenu.updateCurrentItemValue THEN
        bWarningKiller := sysWriteParINT( ADR( rC01_HeatRecoveryUnitDelta ), heatRecoveryUnitMenu.localValue
    END_IF;

    heatRecoveryUnitMenu.currentItemValue := rC01_HeatRecoveryUnitDelta;
    heatRecoveryUnitMenu.currentItemMin := -500;
    heatRecoveryUnitMenu.currentItemMax := 999;
    heatRecoveryUnitMenu.currentItemLabel := 'rC01';
    heatRecoveryUnitMenu.currentItemDisplayType := DISPLAY_TEMPERATURE;

(* Heat recovery unit menu 2nd item *)
1:
    IF heatRecoveryUnitMenu.updateCurrentItemValue THEN
        bWarningKiller := sysWriteParINT( ADR( rC02_HeatRecoveryUnitBand ), heatRecoveryUnitMenu.localValue )
    END_IF;

    heatRecoveryUnitMenu.currentItemValue := rC02_HeatRecoveryUnitBand;
    heatRecoveryUnitMenu.currentItemMin := 1;
    heatRecoveryUnitMenu.currentItemMax := 255;
    heatRecoveryUnitMenu.currentItemLabel := 'rC02';
    heatRecoveryUnitMenu.currentItemDisplayType := DISPLAY_TEMPERATURE;

END_CASE;

heatRecoveryUnitMenu( enable := enableHeatRecoveryUnitMenu );

enableHeatRecoveryUnitMenu := heatRecoveryUnitMenu.enable;
enableServiceMenu := heatRecoveryUnitMenu.back;
```

6. Libraries

The baselines for AHUs make use of various block libraries (FUNCTION and FUNCTION_BLOCK). These libraries can be utilized effectively in the development of applications derived from the baselines.

This chapter contains a detailed description of the blocks contained in the various libraries.

6.1. Actuators

The Actuators library contains the set of actuation logics that are most frequently utilized in HVAC applications.

6.1.1. DelayedStarter_t

Type	FUNCTION_BLOCK
Description	Actuator of a digital output that adds a starting delay and a stopping delay
Inputs	1) `in : BOOL` Value to actuate on the digital output prior to application of the delays. 2) `delayOnStart : UDINT` Starting delay, expressed in milliseconds [ms]. 3) `delayOnStop : UDINT` Stopping delay, expressed in milliseconds [ms]. 4) `delayBetween : UDINT` Minimum pause time between successive starts, expressed in milliseconds [ms].
Outputs	1) `out : BOOL` Value to actuate on the digital output after application of the delays.

6.1.2. StarDeltaStarter_t

Type	FUNCTION_BLOCK
Description	Star-delta starting
Inputs	1) `in : BOOL` Value to actuate on the digital output. 2) `lineStarDelay : UDINT` Star-line delay, expressed in milliseconds [ms]. 3) `starDuration : UDINT` Star-line duration, expressed in milliseconds [ms]. 4) `starDeltaDelay : UDINT` Star-delta delay, expressed in milliseconds [ms].
Outputs	1) `line : BOOL` Line contactor.

	1) `star : BOOL` Star contactor. 1) `delta : BOOL` Delta contactor.

6.2. Alarms

The Alarms library contains the set of management logics of the alarms that are most frequently utilized in the applications.

6.2.1. AutoRearmAlarm_t

Type	FUNCTION_BLOCK
Description	Automatic reset alarm
Inputs	1) `enable : BOOL` Enables the alarm condition test. If `FALSE`, the block output is set to 0 (alarm not active). 2) `condition : BOOL` Alarm condition, active when `condition` equals `TRUE`.
Outputs	1) `alarm : USINT` Alarm status: 0 = not active, 1 = active

6.2.2. DelayAutoRearmAlarm_t

Type	FUNCTION_BLOCK
Description	Delayed activation and automatic reset alarm
Inputs	1) `enable : BOOL` Enables the alarm condition test. If `FALSE`, the block output is set to 0 (alarm not active). 2) `condition : BOOL` Alarm condition, active when `condition` equals `TRUE`. 3) `delay : UDINT` Alarm activation delay, expressed in milliseconds.
Outputs	1) `alarm : USINT` Alarm status: 0 = not active, 1 = active

6.2.3. DelayManualRearmAlarm_t

Type	FUNCTION_BLOCK
Description	Delayed activation alarm with manual reset
Inputs	1) `enable : BOOL` Enables the alarm condition test. If `FALSE`, the block output is set to 0 (alarm not active).

	2) `reset : BOOL` Alarms reset command. If `TRUE` and the alarm is in pending manual reset status, it transits to the non activation status (the block output is 0). 3) `condition : BOOL` Alarm condition, active when `condition` equals `TRUE`. 4) `delay : UDINT` Alarm activation delay, expressed in milliseconds.
Outputs	1) `alarm : USINT` Alarm status: 0 = not active, 1 = active, 2 = pending manual reset

6.2.4. ManualRearmAlarm_t

Type	FUNCTION_BLOCK
Description	Manual reset alarm
Inputs	1) `enable : BOOL` Enables the alarm condition test. If `FALSE`, the block output is set to 0 (alarm not active). 2) `reset : BOOL` Alarms reset command. If `TRUE` and the alarm is in the pending manual reset status, it transits to non activation status (block output is 0). 3) `condition : BOOL` Alarm condition, active when `condition` equals `TRUE`.
Outputs	1) `alarm : USINT` Alarm status: 0 = not active, 1 = active, 2 = pending manual reset

6.2.5. ProbeError_t

Type	FUNCTION_BLOCK
Description	Probe error
Inputs	1) `enable : BOOL` Enables the alarm condition test. If `FALSE`, the block output is set to 0 (alarm not active). 2) `probe : INT` Value read by the probe whose operation/presence must be checked.
Outputs	1) `alarm : USINT` Alarm status: 0 = not active, 1 = active

6.3. Regulators

The Regulators library contains the set of regulation logics that are most frequently utilized in HVAC applications.

6.3.1. Hysteresis_t

Type	FUNCTION_BLOCK
Description	Hysteresis
Inputs	1) `clockwise : BOOL` Direction of hysteresis: if `FALSE`, the direction is counter-clockwise; if `TRUE`, it is clockwise. 2) `in : INT` Value to correlate with the thresholds given by parameters `lo` and `hi`. 3) `lo : INT` Hysteresis lower threshold value. 4) `hi : INT` Hysteresis upper threshold value.
Outputs	1) `out : BOOL` Hysteresis status.

6.3.2. OnOffRegulator_t

Type	FUNCTION_BLOCK
Description	Regulator ON/OFF
Inputs	1) `hcMode : USINT` Direction of regulator: if `COOL`, the control action is active until `feedback` exceeds the `setpoint`; if `HEAT`, it is active until `feedback` is below the `setpoint`. 2) `feedback : INT` Measurement of the physical quantity subject to the control action read by the regulation probe and utilized as feedback for correlation with the `setpoint`. 3) `setpoint : INT` Target value of the physical quantity subject to the control action. 4) `diff : INT` Differential to be added to/subtracted from the setpoint for status changeover from OFF to ON: if `hcMode` equals `COOL`, the control action occurs when `feedback` becomes greater than or equal to `setpoint + diff`; if `hcMode` equals `HEAT`, it occurs when `feedback` is below or equal to `setpoint - diff`.
Outputs	1) `out : BOOL` ON/OFF status of the control action.

6.3.3. ProportionalRegulator_t

Type	FUNCTION_BLOCK
Description	Proportional regulator
Inputs	1) `hcMode : USINT` Direction of regulator: if `COOL`, the control action is active until `feedback` exceeds the `setpoint`; if `HEAT`, it is active until `feedback` is below the `setpoint`. 2) `feedback : INT` Measurement of the physical quantity subject to the control action read by the regulation probe and utilized as feedback for correlation with the `setpoint`. 3) `setpoint : INT` Target value of the physical quantity subject to the control action. 4) `band : INT` Dimension of the proportional band: if `hcMode` equals `COOL`, the regulator saturates (control action equals 100%) when `feedback` becomes higher than or equal to `setpoint + band`; if `hcMode` equals `HEAT`, when `feedback` becomes less than or equal to `setpoint - band`.
Outputs	1) `out : INT` Control action level, in parts per thousand [‰].

6.3.4. ThreeStepsRegulator_t

Type	FUNCTION_BLOCK
Description	Regulator with three steps
Inputs	1) `hcMode : USINT` Direction of regulator: if `COOL`, the control action is active until `feedback` exceeds the `setpoint`; if `HEAT`, it is active until `feedback` is below the `setpoint`. 2) `feedback : INT` Measurement of the physical quantity subject to the control action read by the regulation probe and utilized as feedback for correlation with the `setpoint`. 3) `setpoint : INT` Target value of the physical quantity subject to the control action. 4) `diff : INT` Total differential to add to/subtract from the setpoint for activation of the three power steps: if `hcMode` equals `COOL`, the first step is activated when `feedback` becomes greater than or equal to `setpoint + diff/3`, the second step when `feedback` becomes greater than or equal to `setpoint + 2·diff/3`, and the third step when `feedback` becomes greater than or equal to `setpoint +`

	`diff`; if `hcMode` equals `HEAT`, the first step is activated when `feedback` becomes less than or equal to `setpoint - diff/3`, the second step when `feedback` becomes less than or equal to `setpoint - 2·diff/3`, and the third step when `feedback` becomes less than or equal to `setpoint - diff`;
Outputs	1) `step1 : BOOL` ON/OFF status of the first power step. 2) `step2 : BOOL` ON/OFF status of the second power step. 3) `step3 : BOOL` ON/OFF status of the third power step.

6.3.5. TwoStepsRegulator_t

Type	FUNCTION_BLOCK
Description	Regulator with two steps
Inputs	1) `hcMode : USINT` Direction of regulator: if `COOL`, the control action is active until `feedback` exceeds the `setpoint`; if `HEAT`, it is active until `feedback` is below the `setpoint`. 2) `feedback : INT` Measurement of the physical quantity that is subject to the control action read by the regulation probe and utilized as feedback for correlation with the `setpoint`. 3) `setpoint : INT` Target value of the physical quantity subject to the control action. 4) `diff : INT` Total differential to add to/subtract from the setpoint for activation of the two power steps: if `hcMode` equals `COOL`, the first step is activated when `feedback` becomes greater than or equal to `setpoint + diff/2`, the second step when `feedback` becomes greater than or equal to `setpoint +diff`; if `hcMode` equals HEAT, the first step is activated when feedback becomes less than than or equal to `setpoint - diff/2`, and the second step when `feedback` becomes less than or equal to `setpoint - diff`;
Outputs	1) `step1 : BOOL` ON/OFF status of the first power step. 2) `step2 : BOOL` ON/OFF status of the second power step.

6.4. SmartHMI

The SmartHMI library contains blocks for the construction of user interfaces for **Smart** and the relative terminals (SKW, SKP).

6.4.1. LocalMainView_t

Type	FUNCTION_BLOCK
Description	Block for management of LEDs and detection of prolonged pressing of keys ("hot" key function) of the local **Smart** display, in basic display mode.
Inputs	1) `state : USINT` Unit ON/OFF status, utilized by `LocalMainView_t` for management of the stand-by LED and view of the 'OFF' text on the display. 2) `remoteState : BOOL` If `TRUE`, ON/OFF status is determined by a remote setting, and its display (illumination of the stand-by LED or message 'OFF' shown on the display) is intermittent. 3) `mode : USINT` COOL/HEAT operating mode, utilized by `LocalMainView_t` for management of the corresponding LEDs. 4) `remoteMode : BOOL` If `TRUE`, COOL/HEAT operating mode is determined by a remote setting and its display (illumination of COOL or HEAT LED) is intermittent. 5) `economy : BOOL` Activation of Economy mode, utilized by `LocalMainView_t` for management of the corresponding LED. 6) `timeIsEnabled : BOOL` Activation of time frame operation, utilized by `LocalMainView_t` for management of the corresponding LED. 7) `resource1 : BOOL` Enabling of resource no. 1 (depending on application), utilized by `LocalMainView_t` for management of the corresponding LED. 8) `resource2 : BOOL` Enabling of resource no. 2 (depending on application), utilized by `LocalMainView_t` for management of the corresponding LED. 9) `resource3 : BOOL` Enabling of resource no. 3 (depending on application), utilized by `LocalMainView_t` for management of the corresponding LED. 10) `resource4 : BOOL` Enabling of resource no. 4 (depending on application), utilized by `LocalMainView_t` for management of the corresponding LED. 11) `resource5 : BOOL` Enabling of resource no. 5 (depending on application), utilized by

	`LocalMainView_t` for management of the corresponding LED. 12) `resource6 : BOOL` Enabling of resource no. 6 (depending on application), utilized by `LocalMainView_t` for management of the corresponding LED. 13) `resource7 : BOOL` Enabling of resource no. 7 (depending on application), utilized by `LocalMainView_t` for management of the corresponding LED.
Outputs	1) `hotKeyUP : BOOL` Prolonged press of local display UP key detected. 2) `hotKeyDW : BOOL` Prolonged press of local display DW key detected. 3) `hotKeyESC : BOOL` Prolonged press of local display ESC key detected. 4) `hotKeySET : BOOL` Prolonged press of local display SET key detected.

6.4.2. SKWAlarmMenu_t

Type	FUNCTION_BLOCK
Description	Block for navigation from remote keypad of an alarms menu
Inputs	1) `totalAlarms : USINT` Total number of alarms that can be displayed in the menu. 2) `currentAlarmValue : USINT` Value of current alarm (i.e. of the alarm in the `currentAlarm` position): 0 = not active, 1 = active, 2 = pending manual reset. 3) `currentAlarmLabel : STRING` Text identifying current alarm (i.e. the alarm in the `currentAlarm` position).
INOUT variables	1) `currentAlarm : USINT` Current alarm, expressed as a position in the menu, deriving from the navigation operations performed by the user. 2) `enable : BOOL` Enabling of the menu: can be deactivated following a user command or due to timeout.
Outputs	1) `back : BOOL` Returns user to the higher menu level

6.4.3. SKWFolderMenu_t

Type	FUNCTION_BLOCK
Description	Block for navigation from remote keypad of a menu providing access to a list of submenus
Inputs	1) `folderNumber : USINT` Total number of submenus that can be displayed in the menu. 2) `currentFolderLabel : STRING` Text identifying the current submenu (i.e. submenu in the `currentFolder` position).
INOUT variables	1) `currentFolder : USINT` Current submenu, expressed as a position in the menu, deriving from the navigation operations performed by the user. 2) `enable : BOOL` Enabling of the menu: can be deactivated following a user command or due to timeout.
Outputs	1) `back : BOOL` Returns user to the higher menu level 2) `enableCurrentFolder : BOOL` User command for input in the current submenu (i.e. the submenu in the `currentFolder` position).

6.4.4. SKWLeftDisplayMenu_t

Type	FUNCTION_BLOCK
Description	Block for navigation from the remote keypad of a menu of items, with their values shown on the left-hand display
Inputs	1) `itemNumber : USINT` Total number of items that can be displayed in the menu. 2) `currentItemValue : INT` Text identifying current item (i.e. item in `currentItem` position). 3) `currentItemMin : INT` Minimum value of current item (i.e. item in `currentItem` position). 4) `currentItemMax : INT` Maximum value of current item (i.e. item in `currentItem` position). 5) `currentItemLabel : STRING` Text identifying current item (i.e. item in `currentItem` position). 6) `currentItemDisplayType : USINT` Type of value of the current item (i.e. item in `currentItem` position), expressed with constants `DISPLAY_TEMPERATURE`, `DISPLAY_PRESSURE_ONE_DEC`, ... 7) `subFolder : BOOL` If `TRUE`, the menu provides access also to a submenu (not included in

	the `itemNumber` counter value). 8) `subFolderName : STRING` Text identifying the submenu.
INOUT variables	1) `currentItem : USINT` Current item, expressed as a position in the menu, deriving from the navigation operations performed by the user. 2) `enable : BOOL` Enabling of the menu: can be deactivated following a user command or due to timeout.
Outputs	1) `enableSubFolder : BOOL` User command for entry into the submenu. 2) `back : BOOL` Returns user to the higher menu level 3) `updateCurrentItemValue : BOOL` User command to edit the value of the current item (i.e. item in `currentItem` position). 4) `localValue : INT` Value to assign to the current item (i.e., item in `currentItem` position) when `updateCurrentItemValue` is `TRUE`.

6.4.5. SKWMainView_t

Type	FUNCTION_BLOCK
Description	Block for management of LEDs and detection of prolonged pressing of keys ("hot" key function) of the local **Smart** display, in basic display mode.
Inputs	1) `state : USINT` Unit ON/OFF status, utilized by `SKWMainView_t` for management of stand-by LED and view of 'OFF' text on display. 2) `stateSource : USINT` Source of ON/OFF status: 0 = local, 1 = remote, 2 = time frame operation. 3) `mode : USINT` COOL/HEAT/AUTO operating mode, utilized by `SKWMainView_t` for management of the corresponding LEDs. 4) `modeSource : USINT` Source of operating mode COOL/HEAT/AUTO: 0 = local, 1 = remote, 2 = auto. 5) `modeSelection : USINT` Selectable operating modes COOL/HEAT/AUTO (for editing by the user with remote keypad): 0 = COOL only, 1 = HEAT only, 2 = COOL and HEAT, 3 = COOL, HEAT and AUTO, 4 = remote setting. 6) `rightValueType : USINT` Type of value to show on the right-hand display of the LCD terminal,

	expressed with constants DISPLAY_TEMPERATURE, DISPLAY_PRESSURE_ONE_DEC, ... 7) rightValue : INT Value to show on the right-hand display of the LCD terminal. 8) leftValueType : USINT Type of value to show on the left-hand display of the LCD terminal, expressed with constants DISPLAY_TEMPERATURE, DISPLAY_PRESSURE_ONE_DEC, ... 9) leftValue : INT Value to show on the left-hand display of the LCD terminal. 10) leftText : STRING Text to show on the left-hand display of the LCD terminal, if leftValueType equals TEXT. 11) generalAlarm : BOOL If TRUE, the unit is in OFF status from alarm and generalAlarmMsg is shown on the display. 12) generalAlarmMsg : STRING Message to show on display when generalAlarm is TRUE. 13) alarm : USINT Alarms status, SKWMainView_t for management of the corresponding LED. If 0, this means no active alarm; if 1; at least one active alarm; if 2, all active alarms are pending manual reset. 14) economy : BOOL Activation of Economy mode, utilized by LocalMainView_t for management of the corresponding LED. 15) currentTimeProfile : USINT Current time frame operation profile. 16) fanSpeedControl : BOOL If TRUE, management of fans speed from LCD keypad is enabled. 17) fanSpeed : USINT Fans speed, if fanSpeedControl is TRUE.
INOUT variables	1) enable : BOOL Enabling of the menu: can be deactivated following a user command (for entry in a submenu).
Outputs	1) enableSetMenu : BOOL User command for entry into the set menu. 2) enablePrgMenu : BOOL User command for entry into the Prg menu. 3) timeout : BOOL Timeout detected in basic display mode. 4) changeStateRequest : BOOL User command for changeover of unit ON/OFF status. 5) newState : USINT New value of unit ON/OFF status requested by user, if

	`changeStateRequest` is `TRUE`. 6) `changeModeRequest : BOOL` User command for changeover of operating mode COOL/HEAT/AUTO. 7) `newMode : USINT` New value of operating mode COOL/HEAT/AUTO requested by user, if `changeModeRequest` is `TRUE`. 8) `changeFanSpeedRequest : BOOL` User command for fan speed change. 9) `newFanSpeed : USINT` New fan speed requested by user, if `changeFanSpeedRequest` is `TRUE`. 10) `toggleTimeRequest : BOOL` User command to enable/disable time frame operation.

6.4.6. SKWPswProtection_t

Type	FUNCTION_BLOCK
Description	Block for password protection of access to a menu for remote keypad
Inputs	1) `timeout : BOOL` If `TRUE`, access privileges forgotten (due to timeout) and the password must be re-entered to gain access to the menu.
INOUT variables	`1) enable : BOOL` Enabling of the menu: can be deactivated following a user command or due to timeout.
Outputs	1) `enableMenu : BOOL` Menu access rights acquired (user has entered the password correctly). 2) `back : BOOL` Returns user to the higher menu level

6.4.7. SKWRightDisplayMenu_t

Type	FUNCTION_BLOCK
Description	Block for navigation from the remote keypad of a menu of items, with their values shown on the right-hand display
Inputs	1) `itemNumber : USINT` Total number of alarms that can be displayed in the menu. 2) `currentItemValue : INT` Text identifying current item (i.e. item in `currentItem` position). 3) `currentItemMin : INT` Minimum value of current item (i.e. item in `currentItem` position). 4) `currentItemMax : INT`

	Maximum value of current item (i.e. item in `currentItem` position). 5) `currentItemLabel : STRING` Text identifying current item (i.e. item in `currentItem` position). 6) `currentItemDisplayType : USINT` Type of value of current item (i.e. item in `currentItem` position), expressed with constants `DISPLAY_TEMPERATURE`, `DISPLAY_PRESSURE_ONE_DEC`, ... 7) `subFolder : BOOL` If `TRUE`, the menu provides access also to a submenu (not included in the `itemNumber` count value). 8) `subFolderName : STRING` Identification text of submenu.
INOUT variables	1) `currentItem : USINT` Current item, expressed as a position in the menu, deriving from the navigation operations performed by the user. 2) `enable : BOOL` Enabling of the menu: can be deactivated following a user command or due to timeout.
Outputs	1) `enableSubFolder : BOOL` User command for entry into the submenu. 2) `back : BOOL` User command to return to higher menu level 3) `updateCurrentItemValue : BOOL` User command to edit value of current item (i.e. item in `currentItem` position). 4) `localValue : INT` Value to assign to current item (i.e. item in `currentItem` position) when `updateCurrentItemValue` is `TRUE`.

6.5. Thermodynamics

The Thermodynamics library contains calculation functions of physical units of importance in the development of HVAC applications.

6.5.1. CalcEnthalpy

Type	FUNCTION
Description	Approximate calculation of specific enthalpy
Inputs	1) `temperature : INT` Temperature, in tenths of a degree Celsius [°C · 10^{-1}]. 2) `humidity : INT` Relative humidity percentage, in parts per thousand [‰]. 3) `altitude : INT` Height above sea level, in hundreds of metres [m · 10^{2}]
Outputs	Specific enthalpy [(kJ/kg) · 10^{-1}].

6.6. Utils

The Utils library contains generic blocks of utilities that do not fall logically into any of the foregoing libraries.

6.6.1. DynamicSetpoint_t

Type	FUNCTION_BLOCK
Description	Block for management of a dynamic differential to apply to a setpoint
Inputs	1) `enable : BOOL` Enabling of dynamic differential. If FALSE, the differential output equals 0. 2) `hcMode : USINT` Direction of regulator: if COOL, dynamic differential is greater than 0 when feedback is higher than setpoint; if HEAT, when feedback is below setpoint. 3) `maxDifferential : INT` Maximum value of dynamic differential (i.e. maximum value of differential output). 4) `feedback : INT` Measurement of the physical unit to which value of dynamic differential is linked, utilized as feedback for correlation with the setpoint. 5) `setpoint : INT` Value of the physical unit to which value of dynamic differential is linked, for which the differential is cancelled. 6) `band : INT` Proportional band of increase of the value of the differential value up to its maximum, maxDifferential.
Outputs	1) `differential : INT` Dynamic differential to apply.

6.6.2. PercentageReduction

Type	FUNCTION
Description	Percentage reduction of a value expressed in parts per thousand [‰]
Inputs	1) `in : INT` Value to reduce, expressed in parts per thousand [‰]. 2) `reduction : INT` Value of the percentage reduction to apply, expressed in parts per thousand [‰].
Outputs	Value resulting from the percentage reduction

6.6.3. TimeFrameManager_t

Type	FUNCTION_BLOCK
Description	Time frame operation
Inputs	1) `enable : BOOL` Enable time frame operation.
Outputs	1) `mode : USINT` Mode (COMFORT, ECO or OFF) corresponding to the current time frame.

APPENDIX

1. Block diagram

The block diagram shown in Figure 17 represents the high level software structure of a baseline. See also the same architecture diagram in Figure 1

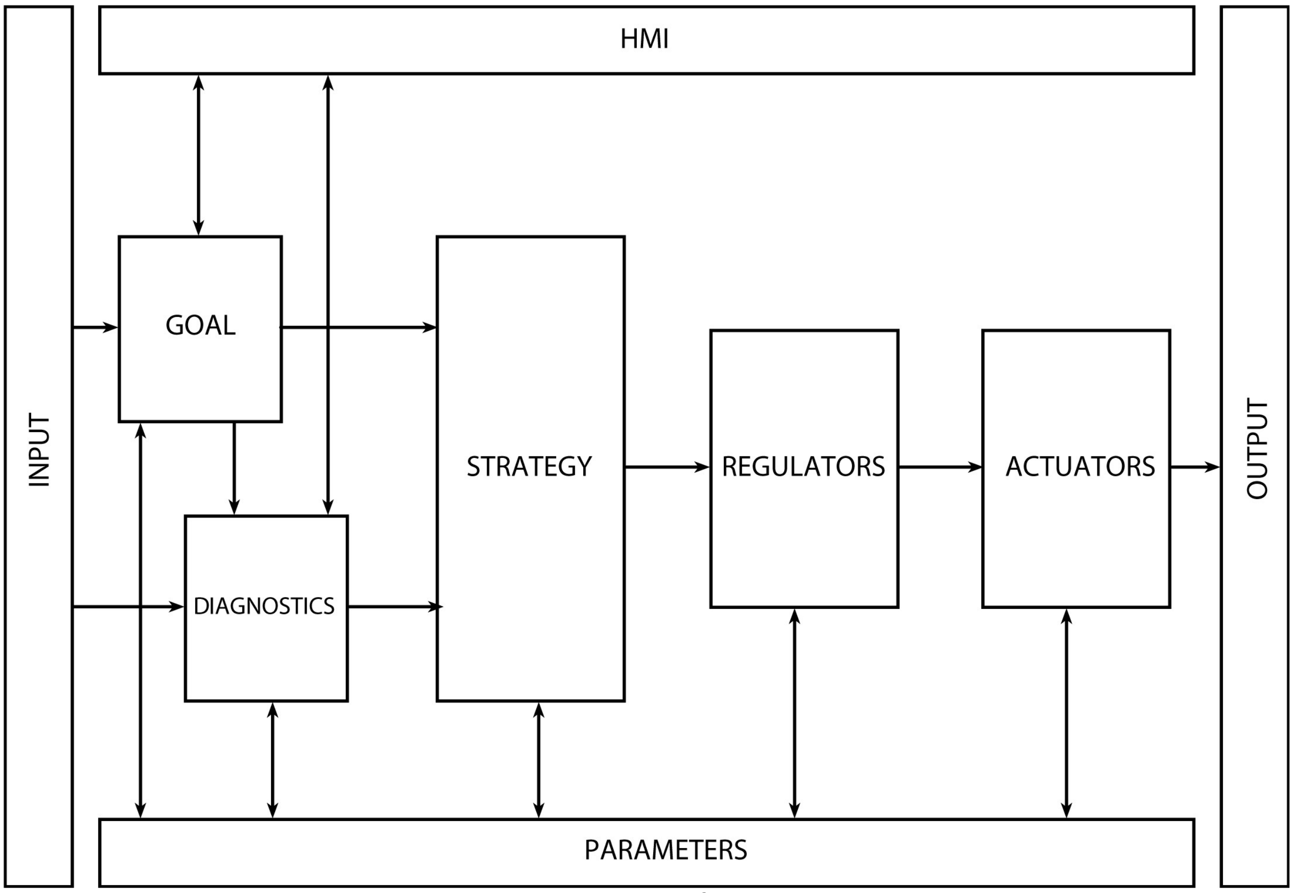


Figure 17: High level software structure

The following headings provide detailed descriptions of the individual items of the diagram and their connections.

1.1. INPUT

1.1.1. Responsibility

The INPUT block translates physical inputs into logical inputs utilized by the application. Specifically, the block:

- maps the physical inputs on the corresponding logical inputs (i.e., INPUT determines which DIs or AIs correspond to which symbolic names utilized in the application);
- converts the physical value of the inputs into the logical value utilized in the application.

1.1.2. Collaborations

The physical inputs read by INPUT are external to the application architecture.
The logical inputs produced by INPUT are utilized by:

- GOAL, with regard to remote setting of the operating mode and activation of Economy mode;
- DIAGNOSTICS, with regard to digital diagnostic inputs (thermal protections, flow switches, pressure switches, etc.) and analog diagnostics (probe error, high/low pressure, etc.);
- STRATEGY and REGULATORS, with regard to regulators feedback.

1.2. PARAMETERS

1.2.1. Responsibility

The PARAMETERS block provides the application with the configuration parameters supplied by USER.

1.2.2. Collaborations

The PARAMETERS outputs are utilized by:

- GOAL (setpoints, time frames, etc.);
- DIAGNOSTICS (alarms configuration);
- REGULATORS (regulators configuration);
- ACTUATORS (actuators configuration).

1.3. GOAL

1.3.1. Responsibility

The GOAL block establishes the aims that the unit controller must pursue.

Specifically, the data resulting from the GOAL block are:

- unit ON/OFF status;
- COOL/HEAT (summer/winter) operating mode of the unit;
- energy saving mode activation status (ECO mode);
- setpoints to follow;
- measurements (feedback) to use in regulation, deriving from the choice of regulation probes.

The GOAL block is necessary to summarize these data starting from the different possible sources, which include parameters and digital and analog inputs.

The GOAL block is not concerned with whether or how the goals can be achieved, delegating this task to the downstream blocks.

1.3.2. Collaborations

To achieve this aim, GOAL makes use of:

- local setting (from PARAMETERS) and remote setting (from INPUT) of the ON/OFF stratus and the requested operating mode;
- setpoint (from PARAMETERS);
- time frames and their activation (from PARAMETERS);
- activation (from INPUT) and settings (from PARAMETERS) of Economy mode;
- dynamic setpoint input (from INPUT) and its settings (from PARAMETERS);
- automatic mode changeover input (from INPUT) and its settings (from PARAMETERS);
- regulation probe selection parameters (from PARAMETERS).

The result of the GOAL calculation is propagated towards other items of the architecture, in particular towards:

- DIAGNOSTICS (unit ON/OFF status);
- STRATEGY (unit ON/OFF status, request type and setpoint to follow, regulation probes);
- HMI (ON/OFF status of the unit and request type).

The above situation is represented schematically in Figure 18.

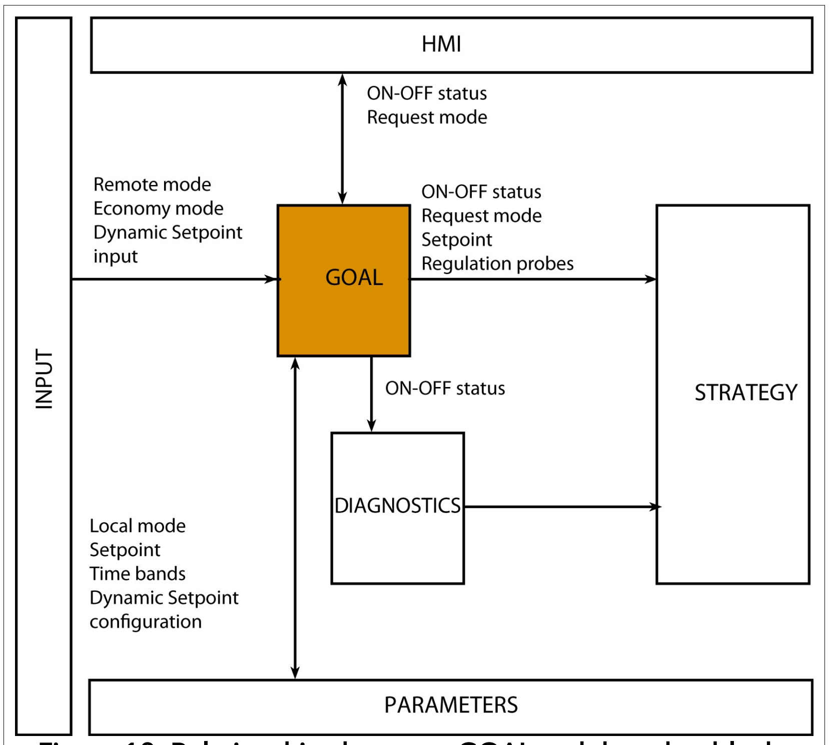


Figure 18: Relationships between GOAL and the other blocks

1.4. DIAGNOSTICS

1.4.1. Responsibility

The DIAGNOSTICS block evaluates the presence of anomaly conditions in the unit and manages activation and deactivation (reset) of the alarms.

1.4.2. Collaborations

DIAGNOSTICS makes use of:

- unit ON/OFF status (from GOAL);
- diagnostic digital inputs (from INPUT) for detection of anomalous conditions in the unit (for example, tripping of a thermal protection);
- analog inputs (from INPUT) for analog diagnostics (e.g. for detection of a missing or faulty probe);
- configuration of alarms activation and resetting logics (from PARAMETERS) (e.g. a delay interval);
- user command for manual alarms reset (from HMI).

The status (active or inactive) of the alarms is transmitted by DIAGNOSTICS to:

- HMI, for signalling of the active alarms and the facility to reset them;
- STRATEGY, for execution of exceptional procedures for faults management.

The diagram in Figure 19 summarizes the DIAGNOSTICS collaborations.

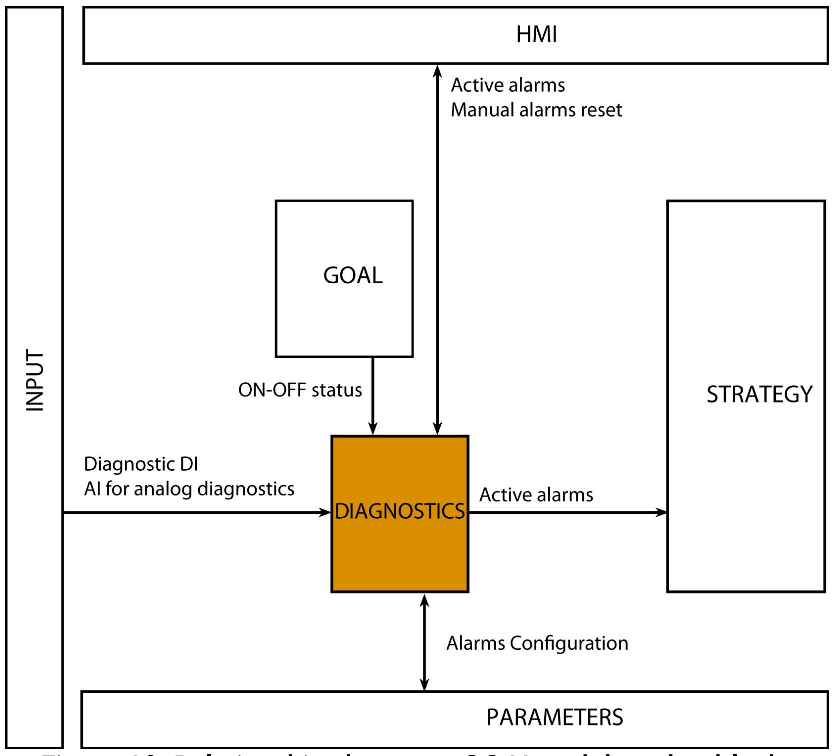


Figure 19: Relationships between GOAL and the other blocks

1.5. STRATEGY

1.5.1. Responsibility

The STRATEGY block establishes the strategy to use to achieve the control goal. In detail, this consists of:

- evaluating the existence of requests to be fulfilled;
- assigning priority to pending requests;
- choosing the strategy to use to fulfill the request from among the various possible strategies;
- managing the request, on the basis of the chosen strategy.

1.5.2. Collaborations

STRATEGY receives the following information as an input:

- ON/OFF status of the unit, the user requests and relative setpoints (from GOAL);
- diagnostic information that calls for a centralized fault management procedure (from DIAGNOSTICS);
- the status of all regulators (from REGULATORS);
- the status of the actuators of interest to the strategy (from ACTUATORS).

The strategy adopted is translated into commands towards REGULATORS/ACTUATORS.
The above situation is represented schematically in Figure 20.

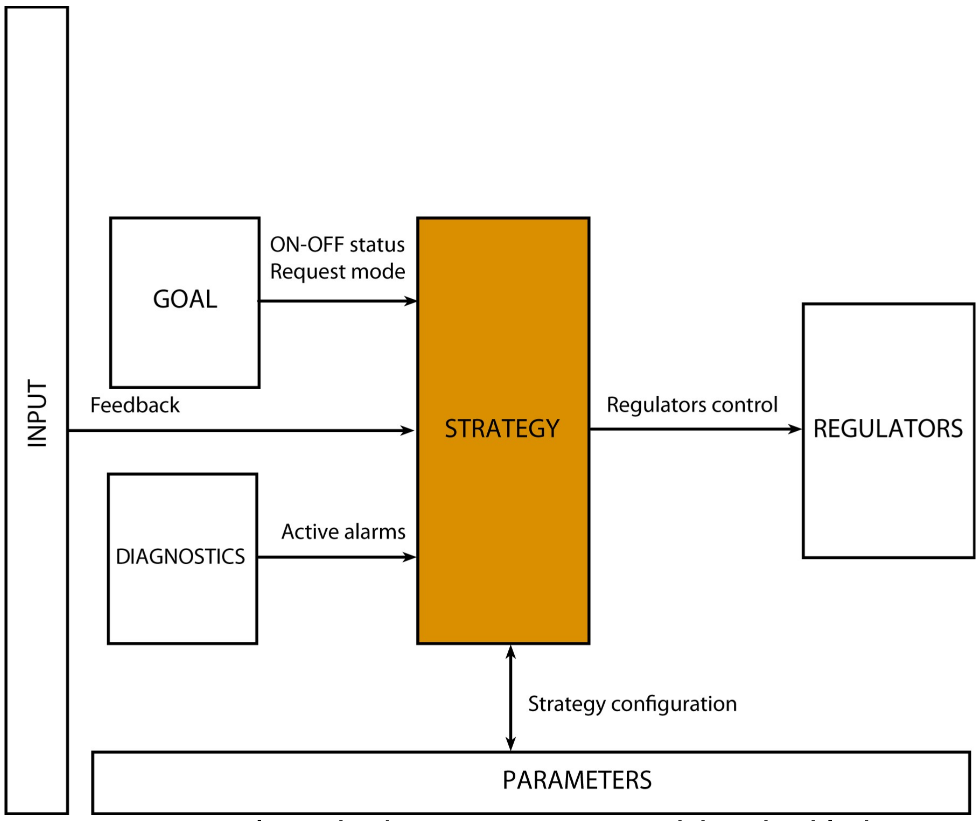


Figure 20: Relationship between STRATEGY and the other blocks

1.6. REGULATORS and ACTUATORS

1.6.1. Responsibility

The values assumed by the physical outputs of the controller are established by the group of REGULATORS and ACTUATORS blocks, which bring together the logic of regulation and of outputs actuation, respectively.

The REGULATORS and ACTUATORS blocks collectively perform the task of fulfilling the requests expressed by the STRATEGY block in terms of setpoints or percentage values.

1.6.2. Collaborations

REGULATORS receives at input:

- the regulator setpoints (from STRATEGY);
- feedback information for regulation (from INPUT) (for example, temperature probes, pressure probes, etc.);
- possible configurations of regulation logics delegated to USER (from PARAMETERS);
- control data - commands for activation, by-pass, etc. (from STRATEGY).

The REGULATORS outputs are used by:

- ACTUATORS, as values to implement on the unit outputs.

ACTUATORS receives at input:

- possible configurations of actuation logics delegated to USER (from PARAMETERS);
- the values of the control actions to implement (from REGULATORS).

The ACTUATORS outputs are used by:

- STRATEGY, in order to change the strategy adopted on the basis of the actuators status;
- OUTPUT, for physical implementation.

The situation described is represented schematically in Figure 21.

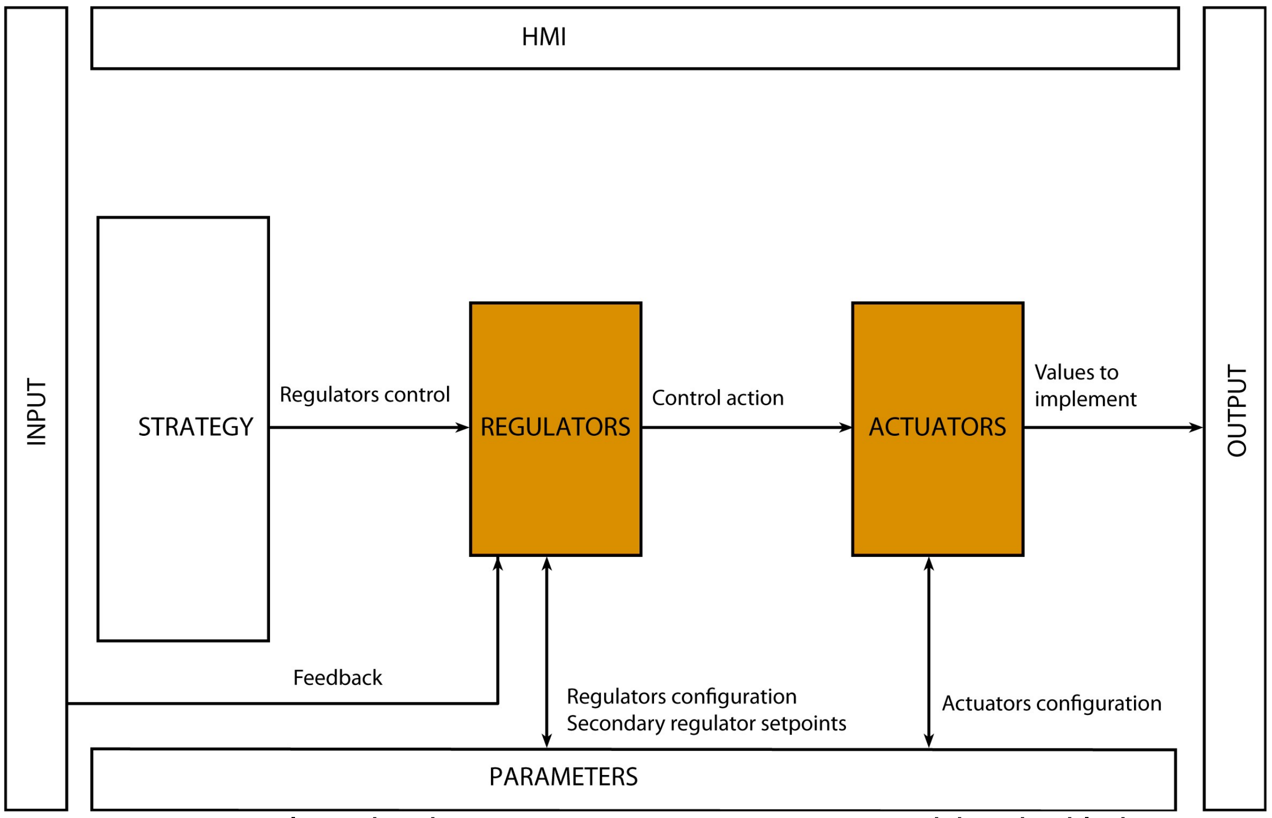


Figure 21: Relationships between REGULATORS, ACTUATORS and the other blocks

1.7. OUTPUT

1.7.1. Responsibility

The OUTPUT block implements the logical outputs on the physical outputs, supplying, in particular:

- mapping between logical and physical outputs (to which DOs or AOs the logical outputs correspond);
- conversion of the logical value utilized in the application into the corresponding physical value.

1.7.2. Collaborations

The logical outputs utilized by OUTPUT are calculated by ACTUATORS.
The physical outputs produced by OUTPUT are external to the system.

1.8. HMI

1.8.1. Responsibility

The HMI block manages the application user interface, in particular with reference to:

- signalling of unit ON/OFF status;
- the operating mode (HEAT, COOL, etc.);
- signalling of active services;
- the application menu;
- keys management.

1.8.2. Collaborations

HMI gathers the following data to view:

- unit ON/OFF status (from GOAL);
- operating mode (from GOAL);
- the set of active services (from OUTPUT).

Data on the activities of the user managed by HMI are used by:

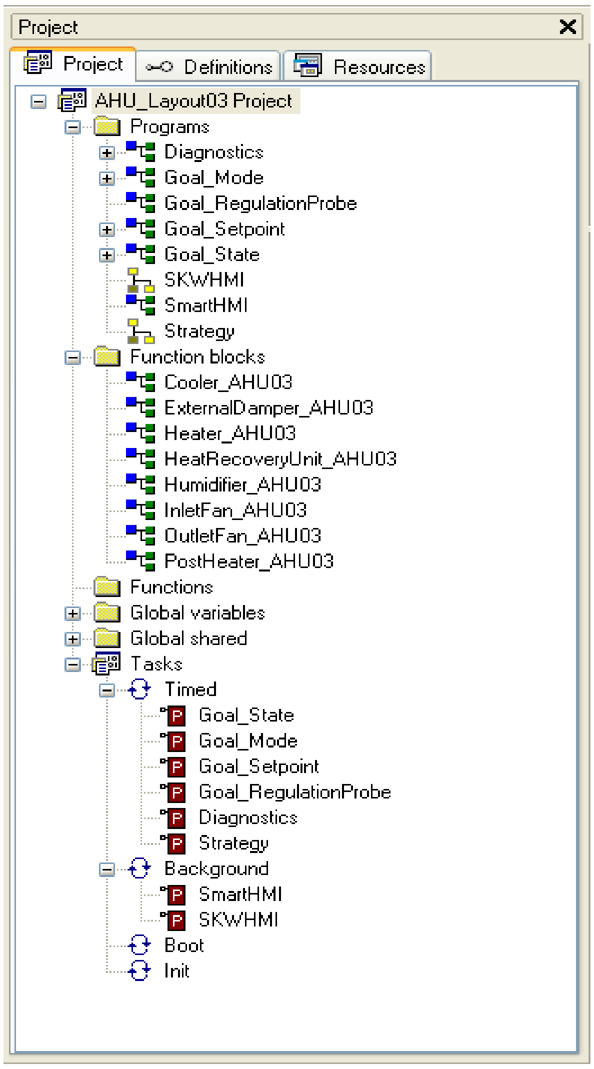
- DIAGNOSTICS, with regard to manual alarms reset.

2. Application project structure

The previous chapter describes the architecture of a baseline from an abstract standpoint: in this chapter, the architecture is explained and described in terms of items of an **Application** project.

2.1. Tasks, programs and functional blocks

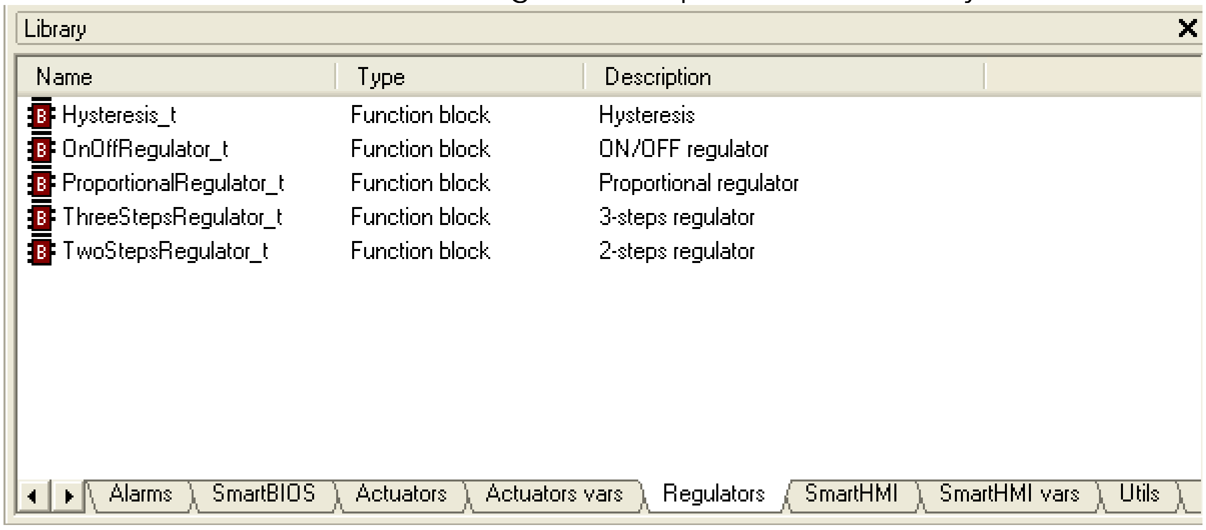
All the blocks described in the previous chapter, with the exception of REGULATORS and ACTUATORS, translate into programs (PROGRAM, in accordance with standard IEC61131-3). This means, specifically, that the exchange of data between blocks occurs by means of global variables, whether they are parameters (EEPROM Parameters), states (Status variables), Alarms, I/Os (I/O mapping) or internal variables (Global variables).
The REGULATORS and ACTUATORS blocks are grouped into functional blocks that represent the physical components of the air handling unit, created as global variables and invoked within the PROGRAM STRATEGY.
All the programs that make up the baseline are executed by the Timed task with the exception of those dedicated to the HMI (significant only for **Smart**).



```
AHU_Layout03 Project
    Programs
        Diagnostics
        Goal_Mode
        Goal_RegulationProbe
        Goal_Setpoint
        Goal_State
        SKWHMI
        SmartHMI
        Strategy
    Function blocks
        Cooler_AHU03
        ExternalDamper_AHU03
        Heater_AHU03
        HeatRecoveryUnit_AHU03
        Humidifier_AHU03
        InletFan_AHU03
        OutletFan_AHU03
        PostHeater_AHU03
    Functions
    Global variables
    Global shared
    Tasks
        Timed
            Goal_State
            Goal_Mode
            Goal_Setpoint
            Goal_RegulationProbe
            Diagnostics
            Strategy
        Background
            SmartHMI
            SKWHMI
        Boot
        Init
```

2.2. Libraries

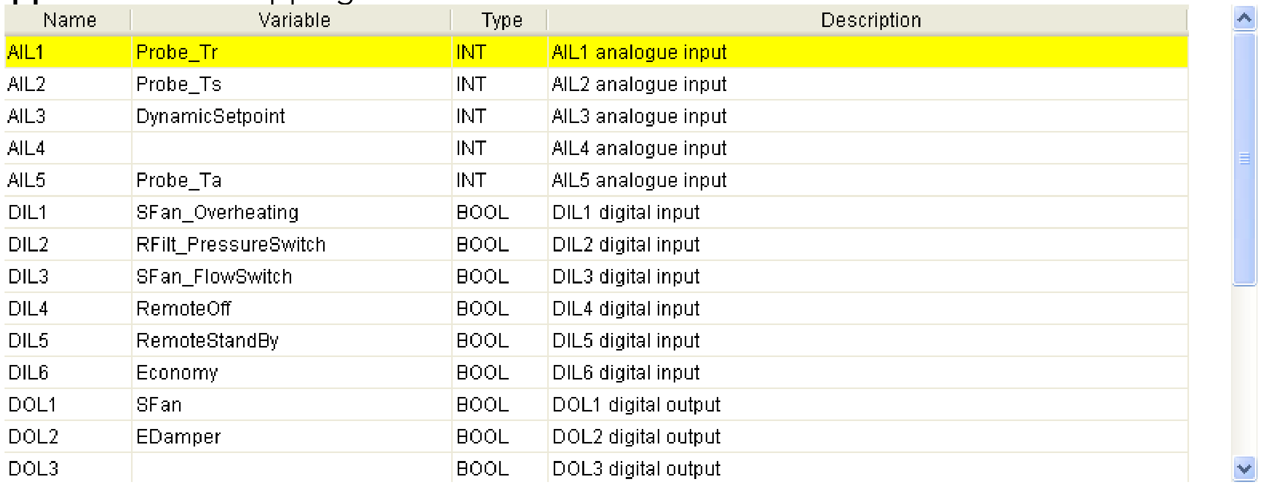
The baseline application makes use of numerous library blocks, subdivided into separate libraries depending on their role (regulators in the regulators library, actuators in the actuators library, etc...).
The number of blocks present in a library may increase at the same time as the need to implement new logics for new applications.
The DEVELOPER can alter the behaviour of an application significantly with simple replacements of a block with an analogous block present in the library.


Library

Name	Type	Description
Hysteresis_t	Function block	Hysteresis
OnOffRegulator_t	Function block	ON/OFF regulator
ProportionalRegulator_t	Function block	Proportional regulator
ThreeStepsRegulator_t	Function block	3-steps regulator
TwoStepsRegulator_t	Function block	2-steps regulator

2.3. INPUT

The conversion of the physical value into the corresponding logical value utilized within the application is performed by the BIOS on the basis of the values of the I/O configuration parameters.
Mapping between physical inputs and logical inputs is performed statically with the **Application** I/O Mapping table.



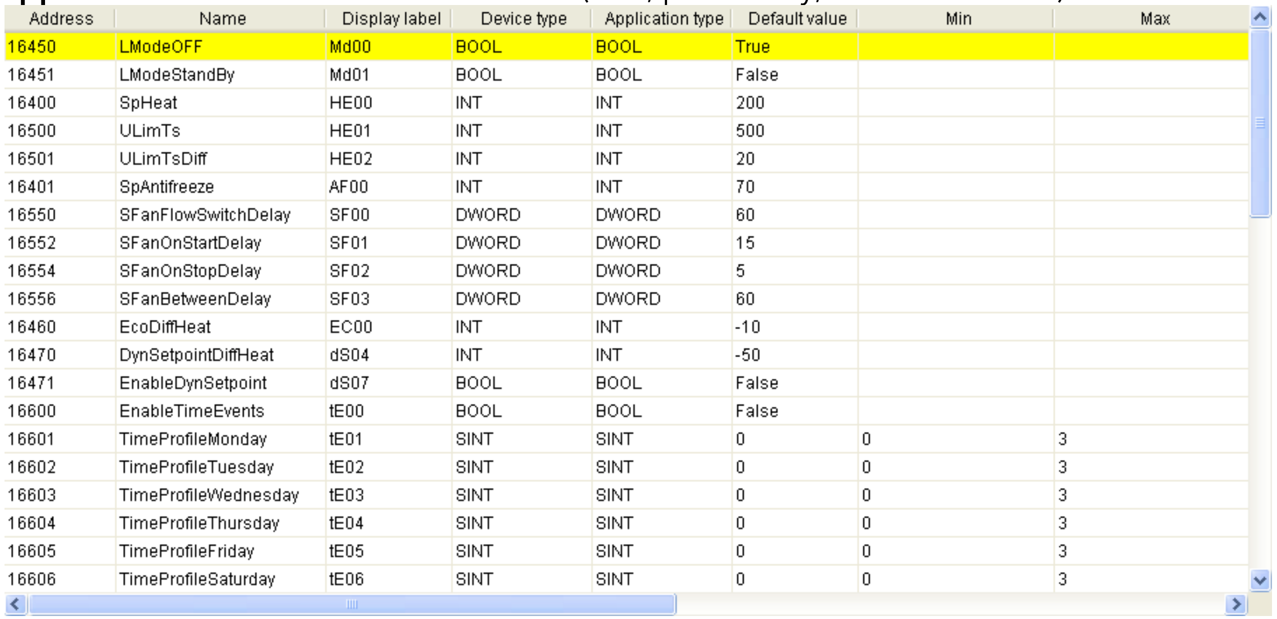
Name	Variable	Type	Description
AIL1	Probe_Tr	INT	AIL1 analogue input
AIL2	Probe_Ts	INT	AIL2 analogue input
AIL3	DynamicSetpoint	INT	AIL3 analogue input
AIL4		INT	AIL4 analogue input
AIL5	Probe_Ta	INT	AIL5 analogue input
DIL1	SFan_Overheating	BOOL	DIL1 digital input
DIL2	RFilt_PressureSwitch	BOOL	DIL2 digital input
DIL3	SFan_FlowSwitch	BOOL	DIL3 digital input
DIL4	RemoteOff	BOOL	DIL4 digital input
DIL5	RemoteStandBy	BOOL	DIL5 digital input
DIL6	Economy	BOOL	DIL6 digital input
DOL1	SFan	BOOL	DOL1 digital output
DOL2	EDamper	BOOL	DOL2 digital output
DOL3		BOOL	DOL3 digital output

2.3.1. Possibilities of action of the DEVELOPER

The DEVELOPER can take action in the context of I/O mapping with the utmost simplicity.

2.4. PARAMETERS

The parameters are made available to the application by means of their definition in the **Application** EEPROM Parameters tables (and, potentially, Status variables).



Address	Name	Display label	Device type	Application type	Default value	Min	Max
16450	LModeOFF	Md00	BOOL	BOOL	True		
16451	LModeStandBy	Md01	BOOL	BOOL	False		
16400	SpHeat	HE00	INT	INT	200		
16500	ULimTs	HE01	INT	INT	500		
16501	ULimTsDiff	HE02	INT	INT	20		
16401	SpAntifreeze	AF00	INT	INT	70		
16550	SFanFlowSwitchDelay	SF00	DWORD	DWORD	60		
16552	SFanOnStartDelay	SF01	DWORD	DWORD	15		
16554	SFanOnStopDelay	SF02	DWORD	DWORD	5		
16556	SFanBetweenDelay	SF03	DWORD	DWORD	60		
16460	EcoDiffHeat	EC00	INT	INT	-10		
16470	DynSetpointDiffHeat	dS04	INT	INT	-50		
16471	EnableDynSetpoint	dS07	BOOL	BOOL	False		
16600	EnableTimeEvents	tE00	BOOL	BOOL	False		
16601	TimeProfileMonday	tE01	SINT	SINT	0	0	3
16602	TimeProfileTuesday	tE02	SINT	SINT	0	0	3
16603	TimeProfileWednesday	tE03	SINT	SINT	0	0	3
16604	TimeProfileThursday	tE04	SINT	SINT	0	0	3
16605	TimeProfileFriday	tE05	SINT	SINT	0	0	3
16606	TimeProfileSaturday	tE06	SINT	SINT	0	0	3

With regard to **Smart**, considering the absence of a parameters database on the target (unlike **Evolution**), the PARAMETERS block can include a dedicated IEC61131-3 PROGRAM that is responsible for validating the values of the application parameters. This logic must be isolated from the remainder of the application, in such a way as to facilitate possible porting on **Evolution**.

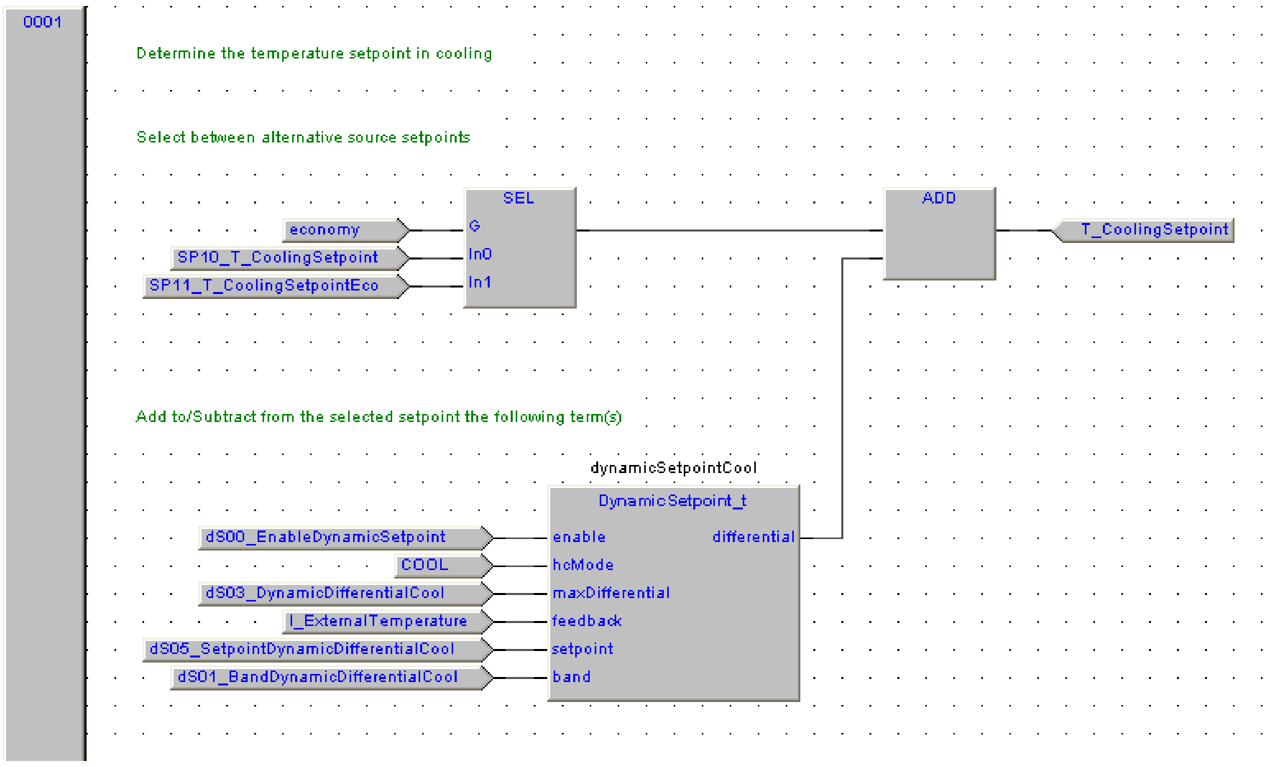
2.4.1. Possibilities of action of the DEVELOPER

The DEVELOPER can easily take action on the definition of parameters (editing of the default value, the range of permissible values, etc.) or add new ones.

2.5. GOAL

Working with an AHU baseline, the GOAL block is composed of a series of programs written in FBD language, the names of which start with `Goal_`.

The results of the GOAL block are made available as global data to the remaining parts of the application, in the folder *Global shared* > *Variables*.


0001
Determine the temperature setpoint in cooling
Select between alternative source setpoints
SEL
economy
SP10_T_CoolingSetpoint
SP11_T_CoolingSetpointEco
G
In0
In1
ADD
T_CoolingSetpoint
Add to/Subtract from the selected setpoint the following term(s)
dynamicSetpointCool
DynamicSetpoint_t
dS00_EnableDynamicSetpoint
COOL
dS03_DynamicDifferentialCool
I_ExternalTemperature
dS05_SetpointDynamicDifferentialCool
dS01_BandDynamicDifferentialCool
enable
hcMode
maxDifferential
feedback
setpoint
band
differential

2.5.1. Possibilities of action of the DEVELOPER

The GOAL block is useful for the following DEVELOPER tasks:

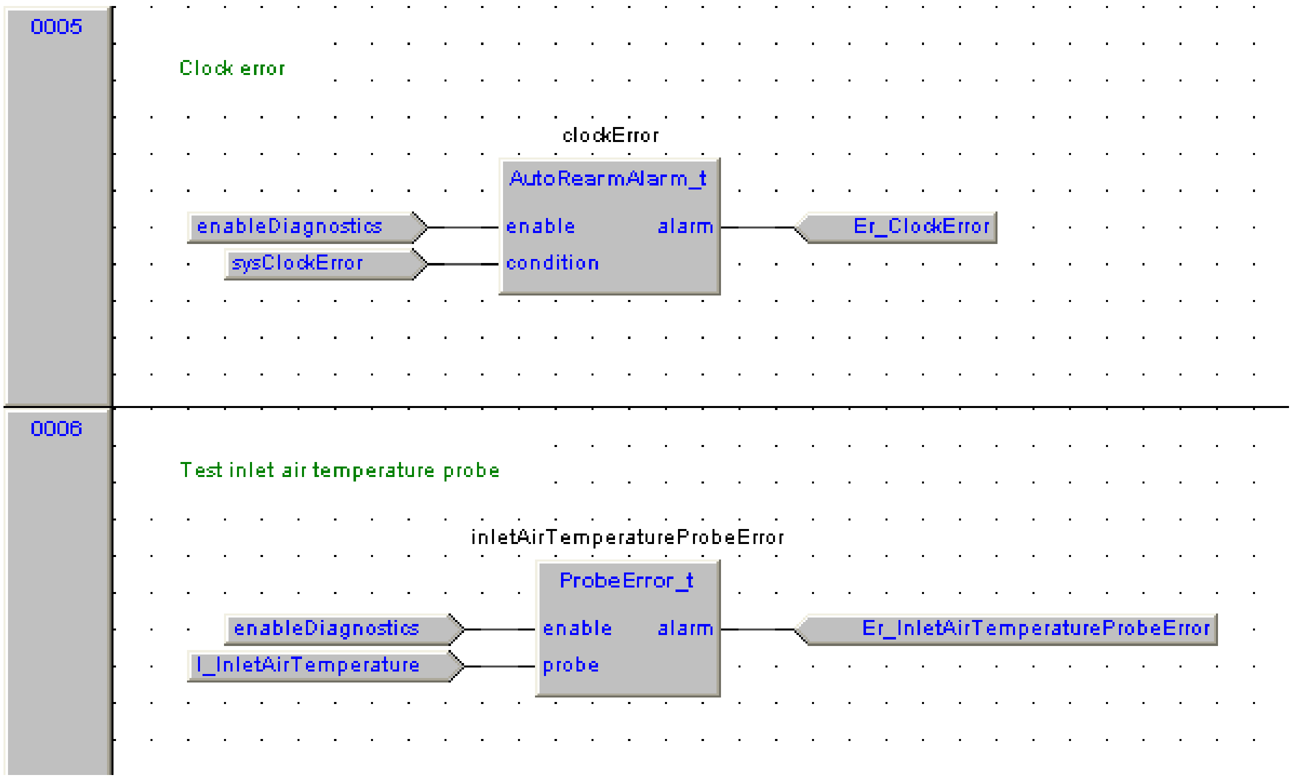
- replacement of a given logic (for example, priority among inputs that determine the operating mode) with an alternative present in the library;
- extension or simplification of the logical networks connected to pins of the blocks (for example, to add a condition to enable time frame operation);
- introduction of new factors for calculation of the setpoints to follow.

2.6. DIAGNOSTICS

Working with an AHU baseline, the DIAGNOSTICS block is constituted by the program of the same name, written in FBD language.

The results of the DIAGNOSTICS block are made available as alarm variables, grouped therefore in the folder *Global shared > Alarms*.

Each network of the `Diagnostics` program is responsible for assigning a value to an individual alarm.



2.6.1. Possibilities of action of the DEVELOPER

The DEVELOPER can easily:

- modify the logics for detection and resetting of alarms (for example, adding or removing delays, making such logics parametric, requesting manual resetting of a specific alarm, etc.);
- add the management of new alarm conditions.

2.7. STRATEGY

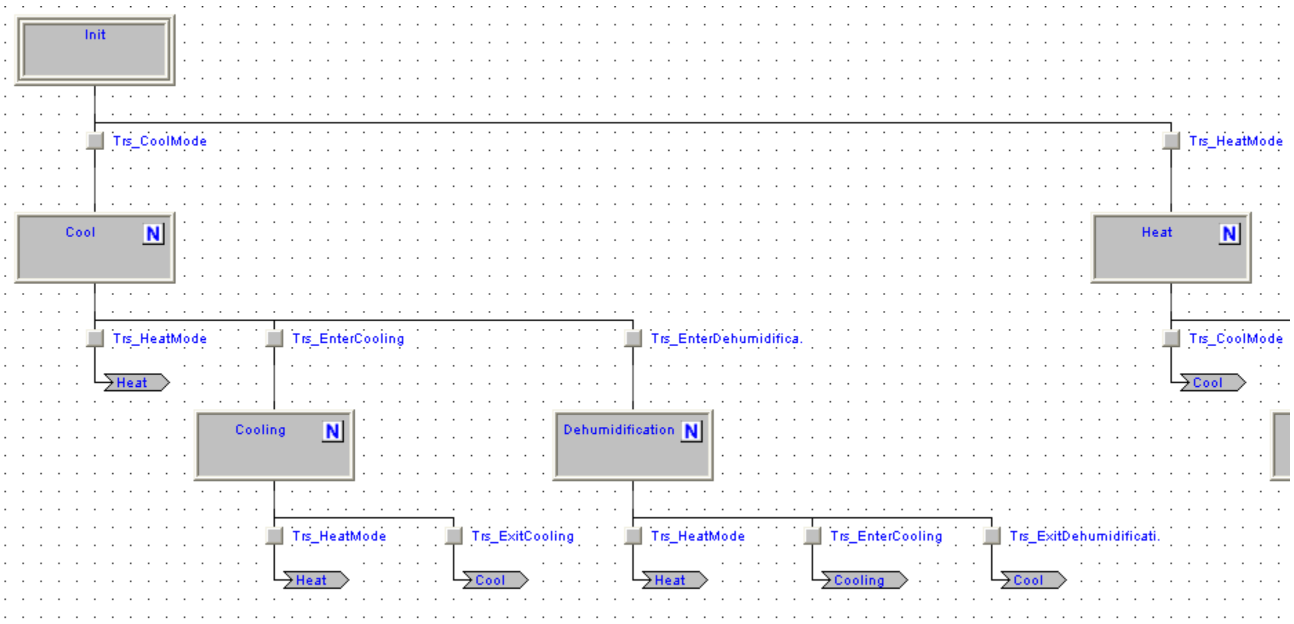
Working with an AHU baseline, the STRATEGY block is constituted by the program of the same name, written in SFC language.

The STRATEGY block is a model of the unit's state machine, which defines:

- the actions to be performed (i.e. the strategy to adopt) for each machine state;
- the possible transitions between states and conditions in which they occur.

The chosen programming language (SFC) allows efficient translation of this state machine:

- the strategies correspond to ACTIONS of the SFC program, implemented in FBD language: each strategy contains an FBD network for each unit component;
- the transitions correspond to SFC program TRANSITIONS implemented in ST language.



2.7.1. Possibilities of action of the DEVELOPER

With reference to the above, in terms of DEVELOPER actions, STRATEGY can:

- manage pre-alarm situations or emergency situations that are not envisaged by the Air Handling Unit baselines, and also provide additional strategies to fulfill the control goals;
- modify the individual strategies already provided for, in terms of actions on the REGULATORS and ACTUATORS blocks.

2.8. REGULATORS and ACTUATORS

The REGULATORS and ACTUATORS blocks are grouped together in functional blocks, which represent the physical components of the air handling unit.

Each strategy determines the inputs of all the blocks and causes their execution.

2.8.1. Possibilities of action of the DEVELOPER

The DEVELOPER has considerable freedom of action in REGULATORS and ACTUATORS, for example, to:

- replace regulation and actuation logics (e.g. with other library blocks);
- insert additional logic on the regulator block pins;
- replace the physical inputs with parametric inputs, or vice versa.
- build complex actuators composed of several actuator blocks although managed as a single actuator by STRATEGY, coordinated by a logical block that is concerned with subdividing requests originating from STRATEGY over different actuators.

Note that is certain cases it may be necessary, or more appropriate, to modify STRATEGY to manage a complex actuator on that level.

2.9. OUTPUT

Mapping between logical outputs and physical outputs is performed statically with the **Application** I/O Mapping table.
Conversion of the logical value utilized within the application into the corresponding physical value is performed by the BIOS on the basis of the values of the I/O configuration parameters.

2.10. HMI

Considering **Smart**, the HMI block is partly automated (local application menu), and partly implemented in dedicated IEC61131-3 PROGRAMS that are responsible for signalling of ON/OFF status, of the operating mode, and of the active services, of the keys management and management of the menu for the remote terminal.
The DEVELOPER can easily act on the entire HMI block.
The HMI logic is isolated from the rest of the application, in such a way as to facilitate possible porting on **Evolution**.

Eliwell Controls Srl
Via dell' Industria, 15 Z. I. Paludi
32010 Pieve d' Alpago (BL) - Italy
Telephone +39 (0)437 986 111
Fax +39 (0)437 989 066
Sales:
+39 (0)437 986 100 (Italy)
+39 (0)437 986 200 (other countries)
saleseliwell@invensys.com
Technical helpline: +39 (0)437 986 250
eliwell.freeway@invensys.com
www.eliwell.it

ISO 9001

9MA10043 06/11