Designing applications for on-wall controllers

Abstract

The Application note deals with design of user applications for small on-wall controllers produced by the AMiT company. This document includes descriptions of procedures for programming communication with on-wall controllers on part of the control system made by AMiT and procedures of screen design with links to objects in a Poseidon network.

Author: Zbyněk Říha
Document: ap0047_en_03.pdf

Attachments

File contents: ap0047_en_03.zip

op70c_p1_en_01.dsox	Example of application design for **AMR-OP70C**.
rs_p1_en_03.dso	Example of operation of **AMR-OP70C** in the control system (ARION).
rs_p2_en_01.dso	Example of operation of **AMR-OP70C** in the control system (MODBUS RTU).
op60_p1_en_01.dsox	Example of operation of buttons in **AMR-OP60**.
op70rhp_p1_en_01.dsox	Example of work with the Poseidon interface in **AMR-OP70RHP**.

Contents

Revision history

Version	Date	Changes by	Changes
001	04. 03. 2011	Říha Z.	New document.
002	20. 04. 2012	Říha Z.	Chapter 3.1.9 amended with information on compatibility with programme SAM-PROG. Chapter 4.7 amended with the procedure of application implementation. Related documentation amended. Applications created in DetStudio version 1.7.2. Images and texts modified in accordance with DetStudio version 1.7.2.
003	25. 03. 2019	Říha Z.	AP name changed, AP structure changed, document revised according to the current behaviour of DetStudio 2.0, controller NOA70 replaced by controller AMR-OP70C.

Related documentation

1. Help tab in the EsiDet section of the DetStudio development environment
   file: Esidet_en.chm
2. Help tab in the PseDet section of the DetStudio development environment
   file: Psedet_en.chm
3. Help tab in the Tridet section of the DetStudio development environment
   file: Tridet_en.chm
4. AMR-OP60 – Operation manual
   file: amr-op60_g_en_xxx.pdf
5. AMR-OP70C – Operation manual
   file: amr-op70c_g_en_xxx.pdf
6. AMR-OP70RHP – Operation manual
   file: amr-op70rhp_g_en_xxx.pdf
7. Application note AP0008 – Communication in MODBUS RTU network (PseDet)
   file: ap0008_en_xx.pdf
8. Application note AP0023 – Scripting in DetStudio
   file: ap0023_en_xx.pdf
9. Application note AP0025 – Communication in ARION network – table definition
   file: ap0025_en_xx.pdf
10. Application note AP0041 – Design of graphic controls for NOA7x controllers
    file: ap0041_en_xx.pdf
11. Application note AP0051 – Communication in Poseidon wireless network
    file: ap0051_en_xx.pdf
12. Application note AP0054 – AMREG communication with AMiT control systems (ARION)
    file: ap0054_en_xx.pdf
13. Application note AP0055 – AMREG communication with AMiT control systems (MODBUS RTU)
    file: ap0055_en_xx.pdf

1 Definitions of terms

Register
32bit value (4 bytes). Registers are used to exchange data between on-wall controllers with touch screens and the superior system. Variables are “mapped” in registers. The number of registers therefore always stems from the number of variables transferred. Register numbers in the ARION network should be in a sequence (e.g. 0, 1, 2, ...).

Mapping a variable
Determining the position of a variable within the register for the ARION/MODBUS RTU network. Mapping may vary for Dint and Real variables.

WYSIWYG
Is an acronym of: “What you see is what you get”. This abbreviation indicates the manner of document editing on a computer in which the version displayed on the screen is visually identical to the resulting version of the document.

NOS
NOS stands for Network Operating System. It is an operating system made by AMiT for selected products.

2 On-wall controllers

On-wall controllers come in the following forms:

- blind controllers with no controls (these only measure selected quantities),
- controllers with buttons or with a mechanical knob,
- controllers with touchscreens.

Options of communication and measurement of various quantities depend on the controller type. Controllers communicate with the superior system via the RS485 interface that supports operation of various communication protocols. This application note deals with ARION and MODBUS RTU communication protocols.

Controllers usually come with firmware that is used for, for example, setting the room regulation mode, temperature setpoint correction, speed of the FanCoil unit fans, or issue a command to turn on a device.

Using controllers with a display has the great advantage of **creating one's own graphic design and control algorithms** in the DetStudio development environment. Other functions unrelated to temperature can be also implemented. The programmer is then able to programme e.g. lighting turning on in the room, control blinds or shades, set time plans, etc. Another great advantage is that all these functions are controlled by the single controller, and if e.g. **AMR-OP70C** is used in the room, there is no need to have another controller/switch on the wall to control the lights and so on. Everything is determined by the programme equipment the programmer provides for the controller.

3 Sample application design for AMR-OP70C

The sample design includes:

- viewing the temperature measured by the on-wall controller,
- viewing the CO_2 concentration measured by the on-wall controller,
- setting the desired room mode,
- setting the desired fan mode,
- signalisation of exceeded CO_2 concentration limit,
- providing measured and set values into the network ARION/MODBUS RTU.

The example is available in the annex to this application note. It is the file named op70c_p1_en_xx.dsox.

3.1 Creating a project

When creating a project, select the on-wall controller “AMR-OP7xC”.

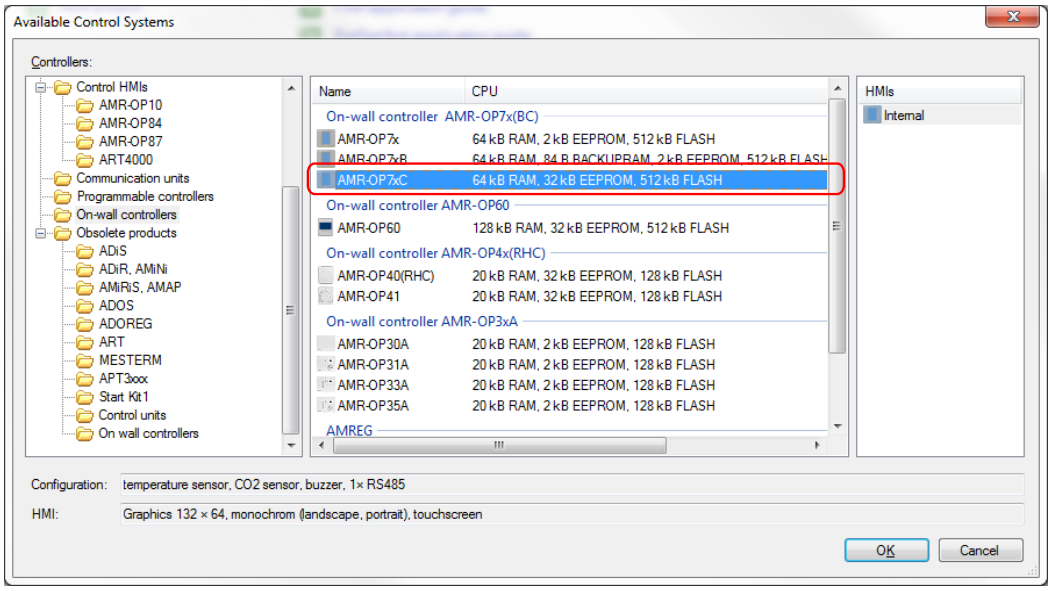

Fig. 1 – Selecting the on-wall controller type

After selecting the on-wall controller type, select “Portrait” orientation (upright).

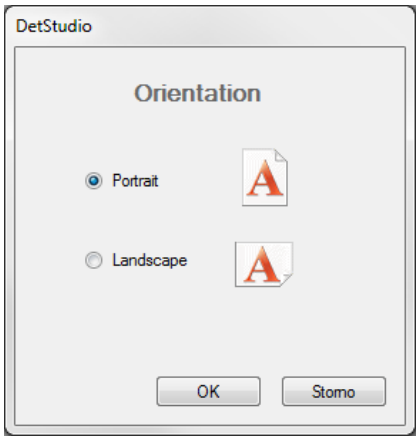

Fig. 2 – Selecting the **AMR-OP70C** screen orientation

3.2 Defining internal variables

The variables that are not supposed to lose their values after the controller restart must be placed into the EEPROM memory. To do so, use the **`EEPROM`** object. After double-clicking the **`EEPROM`** object (in the "Project" window – section "Objects"), the "EEPROM" tab opens for the variables to be inserted with the ***Insert*** key. In the sample application, only the **`Correction`** variable is defined in the EEPROM memory.

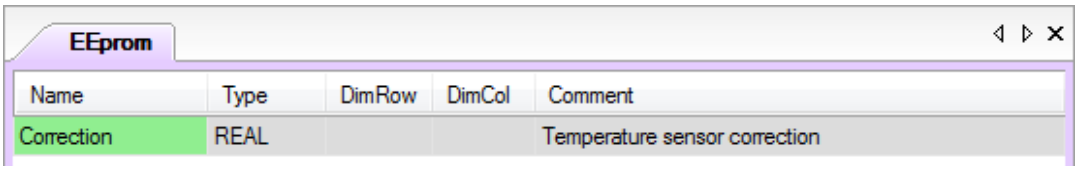

Fig. 3 – Variable in the EEPROM memory

3.3 Definition of the protocol and the variables intended for communication

In the example application, the **AMR-OP70C** is a slave in an ARION or MODBUS RTU network.

3.3.1 Communication protocol definition

Use the **`SerialBusN`** communication object to define the ARION and MODBUS RTU protocol at the same time. To insert the object into the project, select it from the communication objects menu accessible through the context menu – "Add object" – under "Communication".

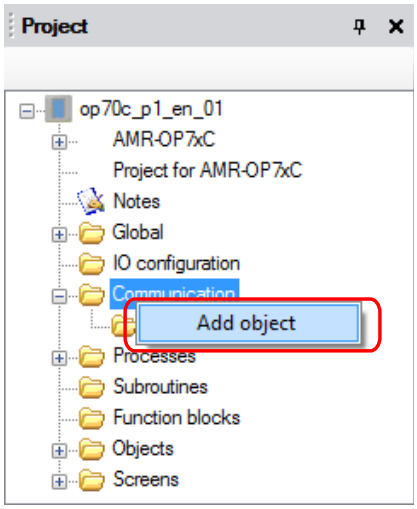

Fig. 4 – Adding objects into the project

After selecting the "Add object", the "Objects and blocks in libraries" list opens; select the **`SerialBusN`** object and confirm the selection by clicking the *"**OK**"* button.

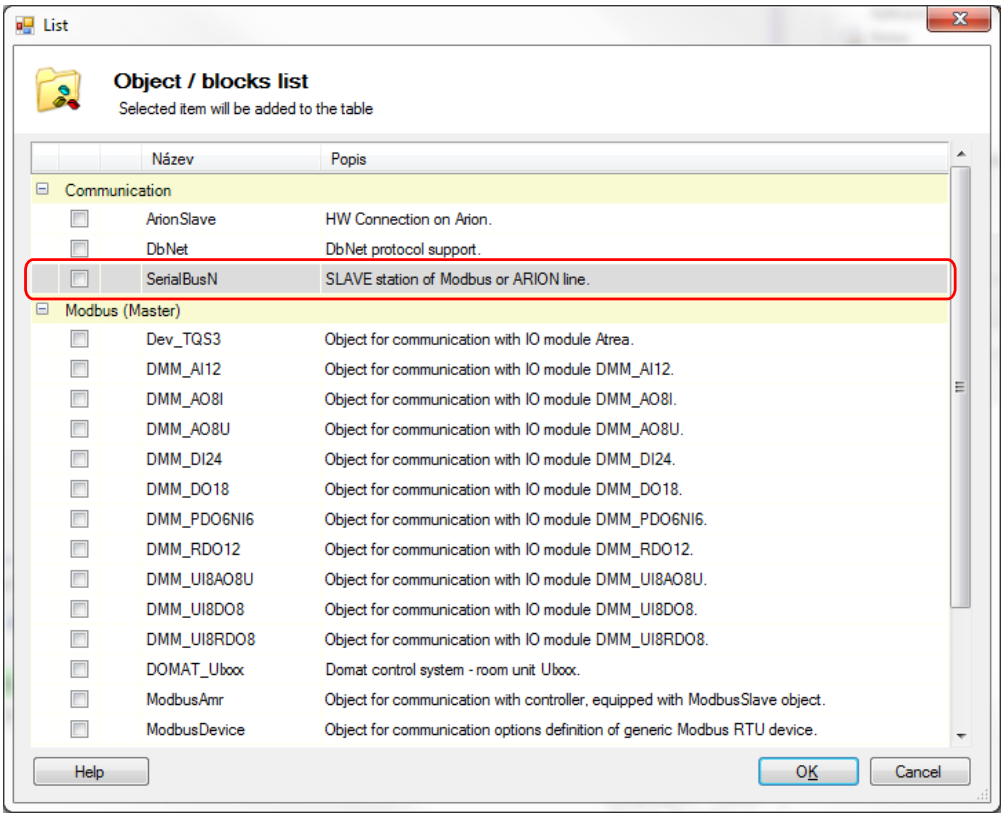

Fig. 5 – Selection of **SerialBusN**

After closing the object list window, **SerialBusN** appears in the "Project" window under "Communication".

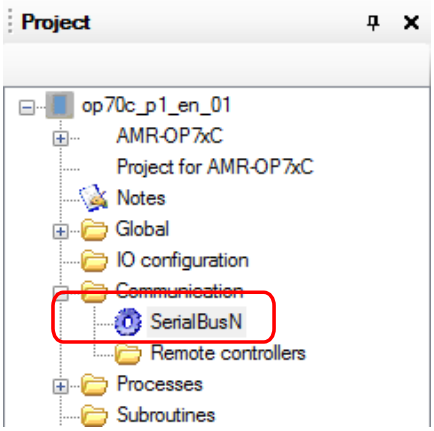

Fig. 6 – **SerialBusN** object in the Communication section

For the time being, hard-code the communication protocol and the communication parameters. The option to change the communication parameters will be dealt with later.

Left-clicking the **SerialBusN** object opens the "Properties" window listing the properties of the communication parameters of the object. Set the **SerialBusN** object properties according to the following picture.

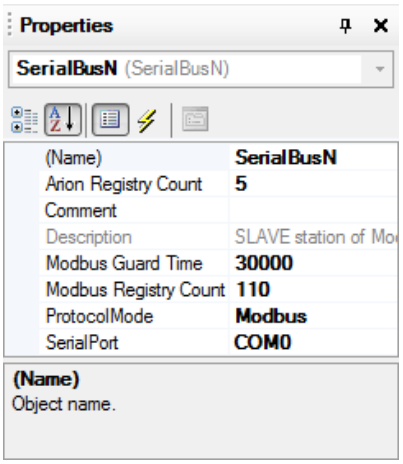

Fig. 7 – Setting communication parameters of the **`SerialBusN`** object

Note
*Leave the communication protocol (**`ProtocolMode`**) as "Modbus" – for the purposes of future updates of the on-wall controller (DetStudio communicates with the on-wall controllers through the MODBUS RTU).*

Set the **`Arion Registry Count`** property to 5 as the example uses:
- 1× DInt register for setting the room mode and the fan mode,
- 1× DInt register for sending the measured values of CO_2 to the superior system,
- 1× DInt register which the superior system uses for setting the CO_2 limit for a warning due to exceeded CO_2 level.
- 1× Real register for sending the temperature values measured by the controller to the superior system,
- 1× Real register which the superior system uses for setting the temperature setpoint to show it on the controller display.

Set the **`Modbus Registry Count`** property to 110. The first 100 registers (0 to 99) are reserved for the system. A single DInt or Real register (size of both is 32 bits) takes two MODBUS registers (as they are 16 bits each).

Other communication parameters can be set in service screens (see the on-wall controller operation manual). In the service screens, the address will be set to 1, the communication speed to 38,400 and even parity for MODBUS RTU.

3.3.2 Definition of variables for data exchange with the control system

As mentioned earlier, the example of controller settings (for the purpose of data exchange with the control system) uses 5 registers per 32 bits. Their descriptions are in the previous chapter.

To set the data exchange with the control system, define six variables (two variables will be mapped in a single register) in the **`SerialBusN`** tab (opened by double-clicking the **`SerialBusN`** object in the "Communication" section of the "Project" window), see the picture below.

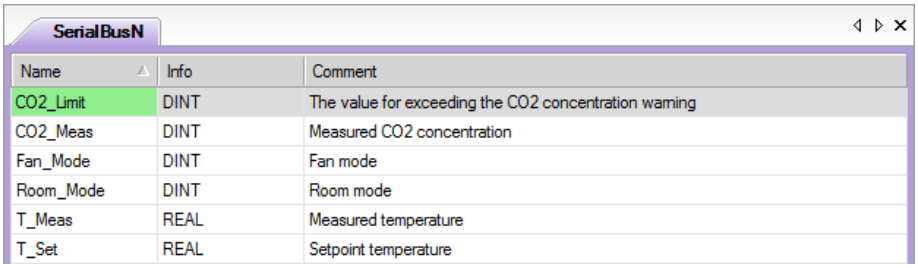

Fig. 8 – Defining variables in the **SerialBusN** object

3.3.3 Mapping variables into registers

Since only 2 bits will be used for the **Room_Mode** variable (room mode) and 3 bits for the **Fan_Mode** variable (fan mode), it is convenient to send both variables into the ARION (MODBUS RTU) network in a single register with 5 bits used up. To map more than one variable into a single register of the ARION (MODBUS RTU) network, use the "Properties" window of the individual variables of the **SerialBusN** object The properties for mapping are **BitLenght**, **BitOffset** and **RegisterArion** (**RegisterModbus**).

Set the variable properties according to the following pictures.

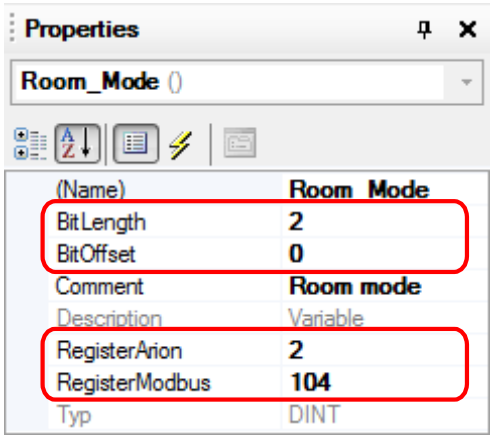

Fig. 9 – Mapping the **Room_Mode** variable into registers

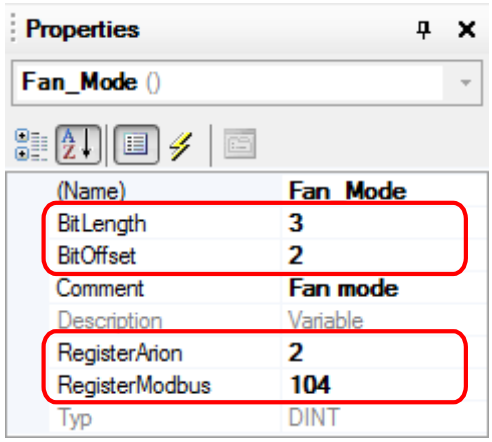

Fig. 10 – Mapping the **Fan_Mode** variable into registers

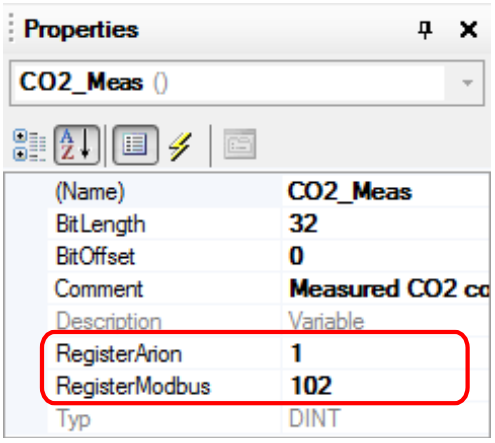

Fig. 11 – Mapping the `CO2_Meas` variable into registers

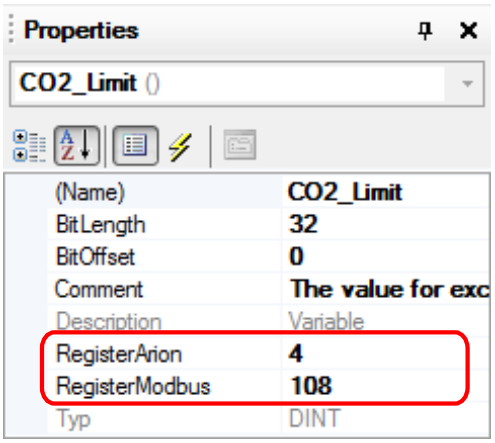

Fig. 12 – Mapping the `CO2_Limit` variable into registers

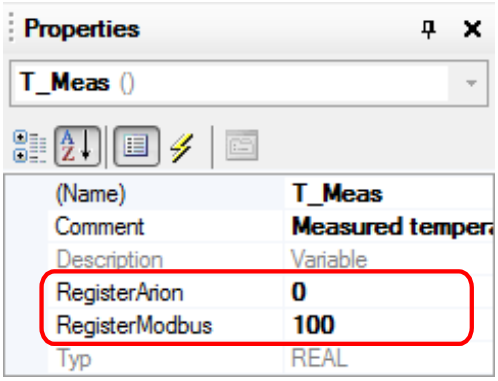

Fig. 13 – Mapping the `T_Meas` variable into registers

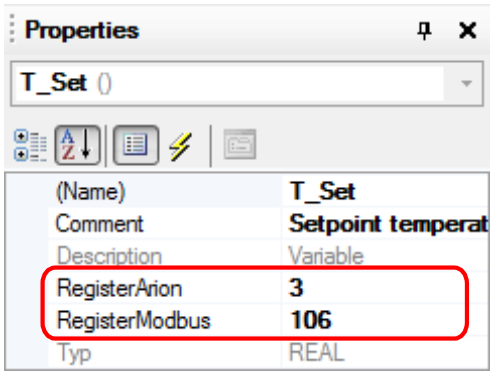

Fig. 14 – Mapping the **T_Set** variable into registers

3.4 Designing the AMR-OP70C the operating application of the controller

To create the operating part of the controller programme that will periodically repeat on the **AMR-OP70C**, use a process called "Process1". Within the process, programme saving of the measured temperature (temperature will be corrected by a correction editable by the user) and saving of the CO_2 concentration into the variables defined in the **SerialBusN** object. (see chapter 3.2 "Defining internal variables").

Note

*The so-called intellisense help (keyboard shortcut **Ctrl+j**) makes it easier to find a specific unit – it contains all available objects, variables and preferences usable for the project. It is possible to list through the list by pressing the up/down arrows. After opening the intellisense help, start typing the object name. This narrows down the list of objects that correspond to the typed letters – confirm the selection by **Enter**.*

Insert the **SerialBusN** object into the process by following the steps in the note above.

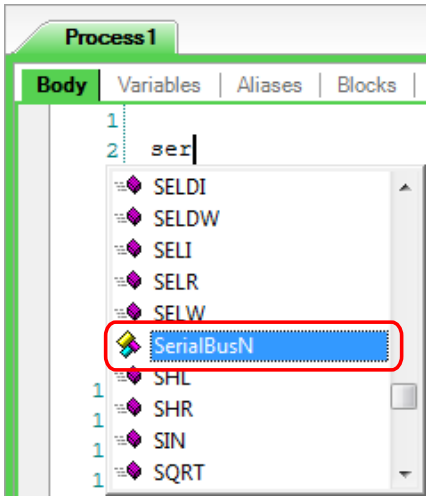

Fig. 15 – Intellisense help

Writing a "." character (full stop) behind the object name, the intellisense list opens again with the list of properties (or methods) that can be used for the given object. For example, save the measured temperature into the **T_Meas** variable. Pick the **T_Meas** variable from the list (a similar method to adding the **SerialBusN** object).

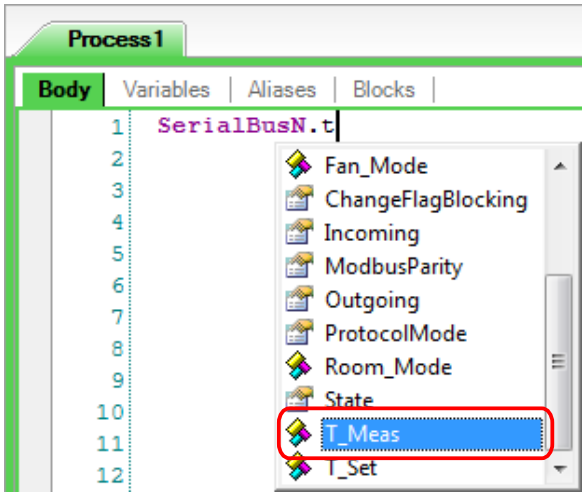

Fig. 16 – Picking the **T_Meas** variable of the **SerialBusN** object

Using the above-mentioned instructions, programme the following code that saves the measured values into the **SerialBusN** object variables.

```
SerialBusN.T_Meas = IO.DeviceTemperature + EEprom.Correction;
SerialBusN.CO2_Meas = IO.CO2;
```

Use the **Correction** variable to correct the measured temperature due to the need for unification of potential differences between temperature measurements from sensors by different manufacturers.

With each activation of "Process 1" (depending on its Period parameter), the application will set the bit no. 1 of the DI channel of the ARION network and the bit no. 1 of the register no. 8 of the MODBUS RTU network to True (write information into one of the registers by the controller). The superior system will list all data from the controller based on bit no. 1. When the write information is set – by the controller – too often, the superior system will try to communicate all data from the controller too often and it may cause significant network traffic. To prevent this state, use, for example, the **Timer** block to set periodicity with which the measured temperatures are written into the register.

Insert the **Timer** block into the project as a global block and set its properties according to the following picture.

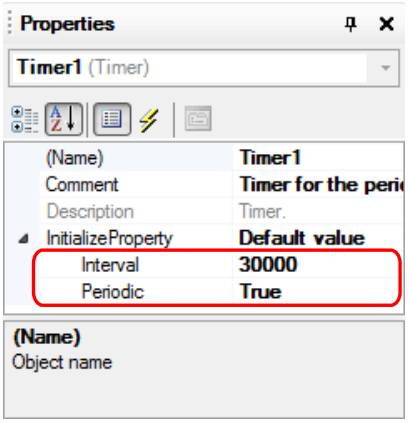

Fig. 17 – Properties of the **Timer1** global block

The periodic process will check whether the timer ticked or not. If yes, the application writes the measured values into registers.

```
Timer1();  // timer operator for periodic reading
if Timer1.Out then  // Timer1 ticked
     SerialBusN.T_Meas = IO.DeviceTemperature + EEprom.Correction;
     SerialBusN.CO2_Meas = IO.CO2;
endif;
```

It is necessary to activate the **Timer1** block by its **Action** parameter. Set the property to True in "ProcessInit" as follows:

```
Timer1.Action=1;
```

To indicate the CO_2 threshold was exceeded, use the **TimerPulse** block (a local versions is sufficient). Write the following expression into the **Input** property of the block:

```
IO.CO2 > SerialBusN.CO2Limit
```

The **Out** property will write directly into the **IO.Buzzer** (the activation of sound notification). Final code will look like this:

```
TimerPulse1(Input = IO.CO2 > SerialBusN.CO2Limit, Out => IO.Buzzer);
```

The duration of the sound notification can be adjusted in the property window of the **TimerPulse1** block (**PulseLength** parameter), see the following picture.

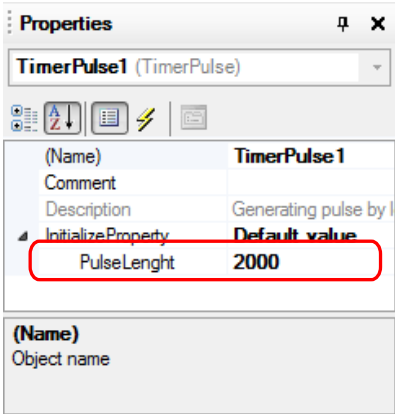

Fig. 18 – Properties of the **TimerPulse1** local block

Set the volume and the frequency of the buzzer directly in the properties window that appears after clicking the **IO.Buzzer** block directly in the structured text of the process.

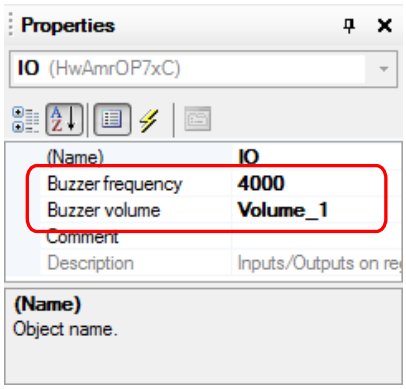

Fig. 19 – Buzzer properties

3.5 Screen design for AMR-OP70C

The process of screen design for **AMR-OP70C** is the same as for any other touchscreen station. As with other touchscreen stations manufactured by AMiT, **this station offers the benefit of choice in terms of screen orientation – either Portrait or LandScape**.

Choose the orientation when creating the project (see 3.1 “Creating a project“) or whenever during the project design (in the Screens section of the “Project settings” window accessible through the “Project” tab in the upper toolbar).

The screen supports both standard text elements available in DetStudio IDE (integrated development environment) or picture/graphic elements. More information on how to process images on **AMR-OP7x** is in the application notes “AP0041 – Designing the graphic elements for the NOA7x series of controllers”.

3.5.1 Setting the heating mode and the fan mode

To allow the user to change the mode, rename the automatically created “Screen1” in the “Screens” tab as “Main”.

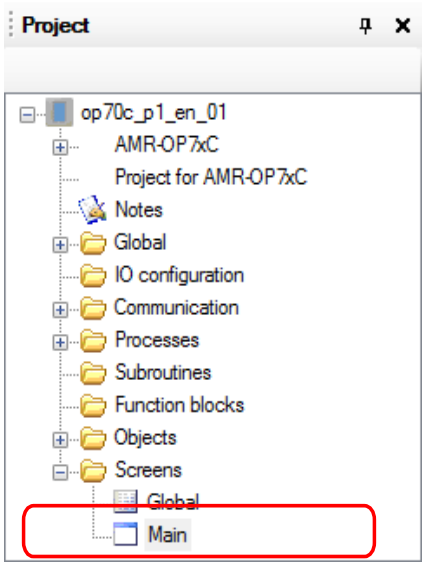

Fig. 20 – Screen name in the “Project” window

To switch between the room heating modes or the fan modes, use, for example, the **`CaseButton`** – the description of this element is in the DetStudio IDE help.

Place two **`CaseButton`** elements onto the screen. Use the sizing handles (eight of them appear directly after the element is placed on the screen or after it is selected by a single click of the left mouse button) to adjust their shape to a square. Adjust their positions according to the following picture.

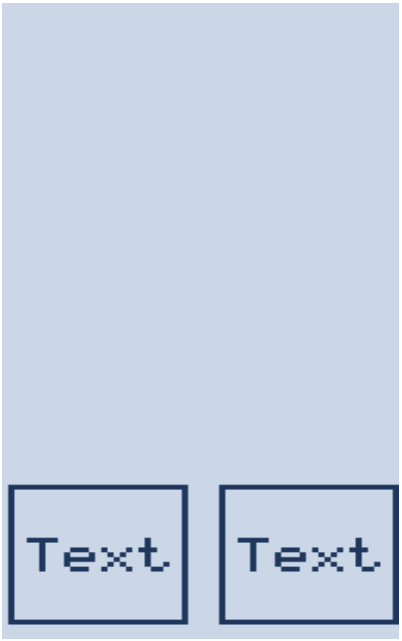

Fig. 21 – Positioning of the **CaseButton** elements on the “Main” screen

The right **CaseButton** element will be used to set the desired heating mode and the left one will be used to set the desired fan mode.

Select the right **CaseButton** element with a left click to display the “Properties” window with the list of its properties. Assign a variable to the element the value of which will change in respect to how many times was the element tapped. Click the **Variable** property in the list of properties, then click the button that appears in the empty field next to the property to open the “Select variable” window. Among other things, the window displays the list of variables. Since the right **CaseButton** will be used for the room mode, choose the **Room_Mode** variable of the **SerialBusN** object.

After confirming the selection, define the appearance of the element through the **Items** property in the “Properties” window.

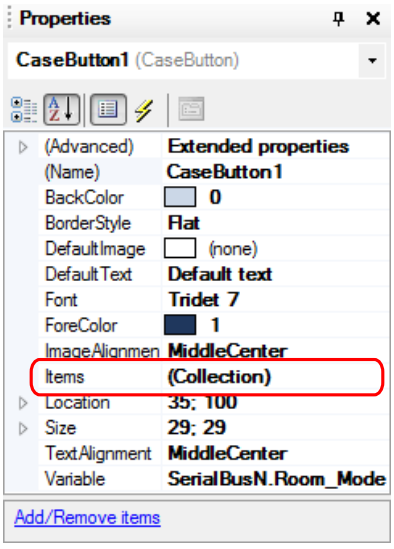

Fig. 22 – Setting the **CaseButton** element appearance

Clicking the ... button next to the aforementioned item opens the "Items" window. Adjust it according to the following picture.

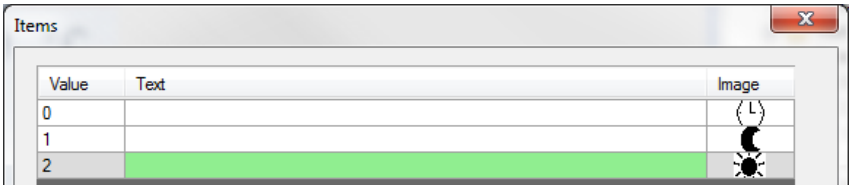

Fig. 23 – Selecting pictures for the individual values of the **Room_Mode** variable

One of the methods for editing the values of individual cells of the items window is to press **F2**.

Set the left **CaseButton** element similarly. In its "Properties" window, set the **Variable** to **Fan_Mode** and its items according to the following picture.

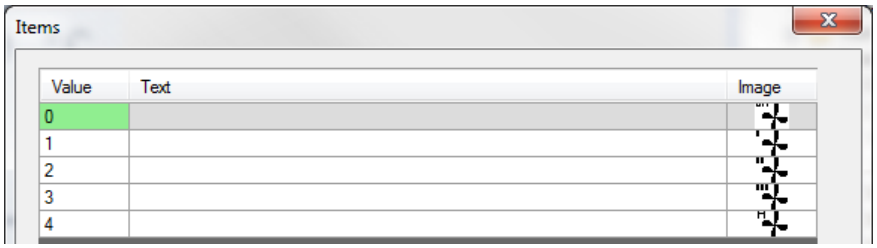

Fig. 24 – Selecting pictures for the individual values of the **Fan_Mode** variable

Icons used in setting the **CaseButton** element are included in the DetStudio installation located at "C:\Users\<user_name>\Documents\DetStudio\Icons\NOAx7x\" as is stated in the "AP0041 – Designing the graphic elements for the NOA7x series of controllers".

3.5.2 Showing the temperature measured by AMR-OP70C

Read the temperature measured by the integrated sensor in **AMR-OP70C** within the operating application and save it into the **SerialBusN.T_Meas** variable (see chapter "3.4Designing the AMR-OP70C the operating application of the controller"). To display the **T_Meas** value, use the **NumericView** element. Drag it onto the screen (position according to the following picture) from the "Basic" section of the "Toolbox" window.

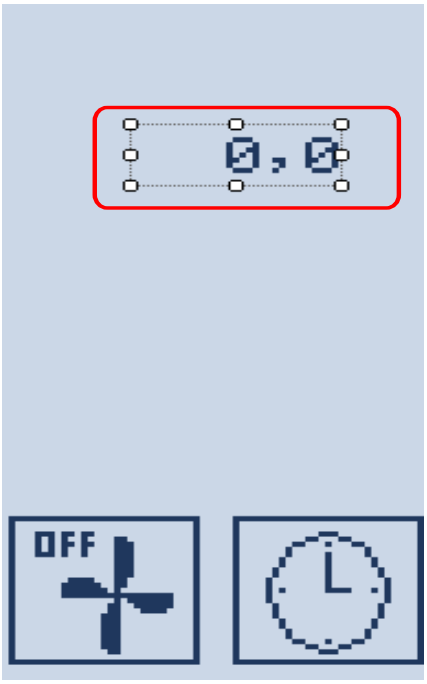

Fig. 25 – Location of the **NumericView** element

Double-clicking the **NumericView** element opens the "Select variable" window – select **SerialBusN.T_Meas** and confirm.

Do not use **IO.DeviceTemperature** directly as the measured value is also corrected by the **EEprom.Correction** variable. The same (corrected) value is sent to the superior system.

After the variable is selected, set the appearance of the **NumericView** variable. Set the **Font** property (display font) to "ADT 21" and leave the **Format** (display mode) as is (i.e. ##.# – real numbers ranging from -9.9 to 99.9). After setting the Location property to "3;24", the **NumericView** element "Properties" window will look like this.

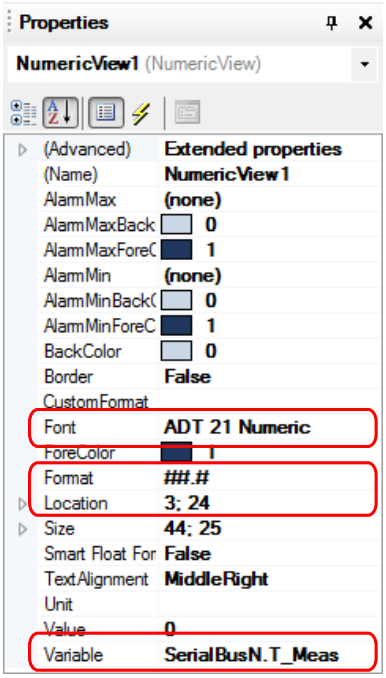

Fig. 26 – Settings of **`NumericView`**

After setting the **`NumericView`** element according to the picture, the “Main” screen should look like this.

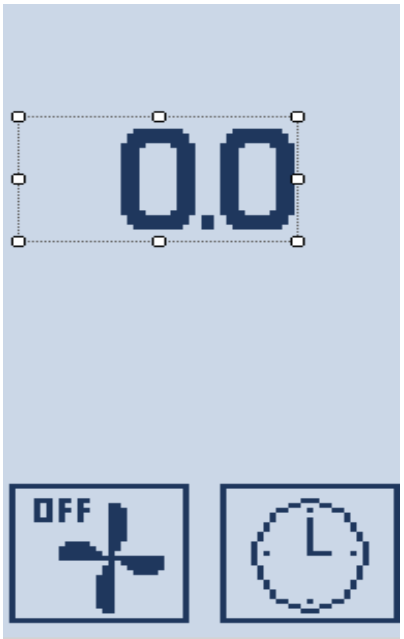

Fig. 27 – “Main” screen with the temperature measured by the integrated sensor

3.5.3 Displaying static text

The user has to know in what units are the values displayed. To save space, display the units in a different size. To achieve that, use the **`Label`** text field located in the “Basic” section of the “Toolbox” window. Place the element at the intended location, double-click it and enter “°C”. The “Main” screen should now look like this.

Fig. 28 – The “Main” screen with static text “°C”

Next, make the screen show the temperature sent by the superior system. As with the previous temperature field, use the **`NumericView`** element. This time, however, set the units in the “Properties” window of the element – look for the **`Unit`** parameter. The “Main” screen should then look like this.

Fig. 29 – The “Main” screen with all the required fields

The above-mentioned example for **AMR-OP70C** is included in the application note attachments – in the op70c_p1_en_xx.dsox file.

3.6 Generating the application for AMR-OP70C

In order for the DetStudio to be able to generate the application, install the “DetStudioTools.exe” package (not included in the DetStudio). This package is available (after signing up) at amitomation.com.

The generation produces a file with the *.bin extension and a name identical to the project name.

3.7 Loading the application in AMR-OP70C

Load the application in **AMR-OP70C** via the MODBUS RTU protocol of the RS485 interface. Use a USB to RS485 converter to connect the PC with the controller (e.g. **SB485s** offered by AMiT). The loading procedure is the same as for the other ethernet-less AMR stations – it is described in the EsiDet help section of the DetStudio IDE help.

4 Operation example of AMR-OP70C in stations with NOS

AMR-OP70C may communicate with the superior system via the ARION or MODBUS RTU protocol. This application note will describe examples with both protocols.

4.1 Example – ARION protocol

This example is available in this application note attachments – in the rs_p1_en_xx.dso file.

4.1.1 Definition of AMR-OP70C in DetStudio

Define **AMR-OP70C** in the ARION table in the AMiT control system (see "AP0025 – Communication in the ARION network – table definition"). To display the table, double-click the "Arion0" item located under Communication / Arion in the project window.

If there is an application with a defined **`SerialBusN`** object in **AMR-OP70C**, define **AMR-OP70C** in the control system application by dragging the **`AMR-OP3x7x`** module into the table (see the following picture).

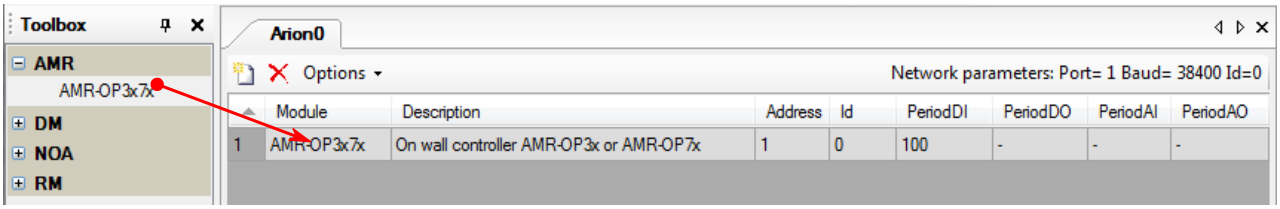

Fig. 30 – Definition **of AMR-OP70C** in the control system

After placing the **`AMR-OP3x7x`** module into the table, define the number of registers available in **AMR-OP70C** in the Properties window of the module. It needs to match the number of registers in the design of the **AMR-OP70C** application (see chapter 3.3.1 "Communication protocol definition"). Since there were 5 registers defined in the application for **AMR-OP70C**, the properties window should look as follows.

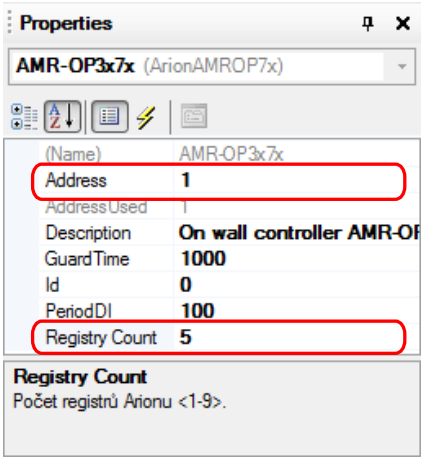

Fig. 31 – Defining the address and the number of registers programmed in **AMR-OP70C**

Other than the defined registers, **AMR-OP70C** also uses an ARION network channel for digital inputs and digital outputs.

In the digital inputs channel, the first three bits are used for the following purposes:
- DI.0 – **AMR-OP70C** was restarted,
- DI.1 – a value was written into one of the variables defined in the **SerialBusN** of **AMR-OP70C,**
- DI.2 – communication failure between **AMR-OP70C** and the control system.

Digital inputs are read with a period defined in the Properties of the **AMR-OP3x7x** module (defined while defining the module in the ARON network table).

The digital outputs channel is used to reset flags set by the channel of digital inputs. As soon as the control system notices the value of one of the digital input registers was set to True, it should perform a corresponding procedure and then reset the input through the digital outputs channel. The individual bits are described below:
- DO.0 – restart flag reset,
- DO.1 – resets the flag set after a **SerialBusN** variable was changed,
- DO.2 – resets the communication failure flag.

4.1.2 Method of communication with AMR-OP70C

There are two methods of communication between **AMR-OP70C** and the control system.
- Simplified method
- Recommended method

In both cases, the communication follows the same structure of communication – a single request handles both writing and reading.

Simplified method
Communication with **AMR-OP70C** is provided by modules **ARI_RegIn** / **ARI_RegOut** – place them into the periodic process. The **ARI_Trig** special module enters the read/write requests.

Recommended method
Communication with **AMR-OP70C** is established via the digital inputs channel. Request for reading from **AMR-OP70C** is entered based on the state of signals of the channel. Therefore, only the DI.1 input is monitored periodically (see previous chapter). **AMR-OP70C** values are read based on the settings of the input – via modules **ARI_RegIn** and **ARI_Trig**.

Write is performed based on the change of a value on the side of the control system through the **ARI_RegOut** module along with the **ARI_Trig** module.

Note
*With **AMR-OP70C**, it is impossible to read only a single register. Using the **ARI_Trig** module always involves all the registers defined in **AMR-OP70C**. However, it is possible to access and save single registers once they are in the control system buffer by using multiple **ARI_RegIn** modules.*

4.1.3 Designing a programme for communication with AMR-OP70C

This guide will use the recommended method of communication between the control system and **AMR-OP70C** (see the previous chapter).

Place the **ARI_State** module into the periodic process – it is used to check whether the controller is communicating with the control system. Additionally, use the **ARI_DigIn** module to check whether the control system wrote into one of the variables mapped into the **SerialBusN** object into the register defined in the controller. Read the registers based on a write flag of a **SerialBusN** variable.

Write the value only when the change is registered and only if there is no read request (writes occur along reads).

```
// state of AMR-OP70C and state of registers transfer
ARI_State 1, OP_state, 4, OP_R_trans

// checking for a change in AMR-OP70C
ARI_DigIn 1, 0, flags, 0x00000000

// writing the temperature setpoint (only when its value changes)
Let @SetPointTmp = (SetPoints[0,0] > (SetPoint_Old[0,0] + 0.1)) or (SetPoints[0,0] <
(SetPoint_Old[0,0] - 0.1))  // a check whether a change occurred or not
Let @LimitCO2 = (SetPoints[1,0] > (SetPoint_Old[1,0] + 0.1)) or (SetPoints[1,0] <
(SetPoint_Old[1,0] - 0.1))  // a check whether a change occurred or not
Let @OP_Writ_Reg = @SetPointTmp or @LimitCO2  // setting the flag for a change or the
lack of thereof

// flag for the change of values issued by the controller, or for a change issued by
the system
Let @OP_RW = (flags > 0) or @OP_Writ_Reg
If @OP_RW  // communication is started based on the flag
     // communication of registers programmed below performs
     // both read and write operations with a single request. That is why data is
     // first saved into a writing buffer and only then it is loaded
     ARI_RegOut 1, 3, 2, SetPoints[0,0], SetPointType[0,0], 5
     // a command for communication of registers of AMR-OP70C
     ARI_Trig 1, 4
     // acknowledging reception of information and resetting the flag for
     // change/restart/loss of service
     Let Flag_reset = 7
     ARI_DigOut 1, 0, 3, Flag_reset, 0x00000000
     // command for flag reset
     ARI_Trig 1, 3
     If @OP_Writ_Reg  // if there is a writing request
          Let SetPoint_Old[0,0] = SetPoints[0,0]  // the written value is saved
          Let SetPoint_Old[1,0] = SetPoints[1,0]  // the written value is saved
     EndIf
EndIf

// buffer is read only after data is communicated
Let @OP_Read_Reg = OP_state.0 and not(OP_R_Trans.0)

If @OP_Read_Reg
     // load measured temperature
     ARI_RegIn 1, 0, 1, T_Meas, NONE[0,0], 5

     // read measured CO2 concentration
     ARI_RegIn 1, 1, 1, CO2, NONE[0,0], 3

     // read room mode and fan mode
     ARI_RegIn 1, 1, 1, OP_Mode, NONE[0,0], 4
EndIf
```

Note

*Using the **ARI_Trig** module issues a command for a simultaneous read and write of registers. If the programmer uses the same variable for both writing and reading and writes into the variable through the **ARI_RegOut**, the application triggers the **ARI_Trig** module and writes into the relevant register in **AMR-OP70C**. If the user changed the value of such register and the control system changed the same value through the **ARI_RegOut** module before the control system acknowledged the change performed by the user, the control system overwrites the value set by the user in **AMR-OP70C** through the **ARI_RegOut** module with the value set by the control system through the **ARI_Trig** module. It is recommended to use different registers for writing and reading to make such issues easier to solve.*

4.2 Example – MODBUS RTU protocol

This example is available in this application note attachments – in the rs_p2_en_xx.dso file.

4.2.1 Definition of AMR-OP70C in DetStudio

Define **AMR-OP70C** in the MODBUS table in the AMiT control system (see "AP0008 – Communication in MODBUS RTU network (PseDet)"). Insert the definition of master communication in MODBUS RTU network into the "Modbus" folder in the project window through the "Modbus" folder context menu. Subsequently, insert the definition of the device (ModbusDevice) with which the control system is supposed to communicate. Do so through the "ModbusMaster0" object context menu.

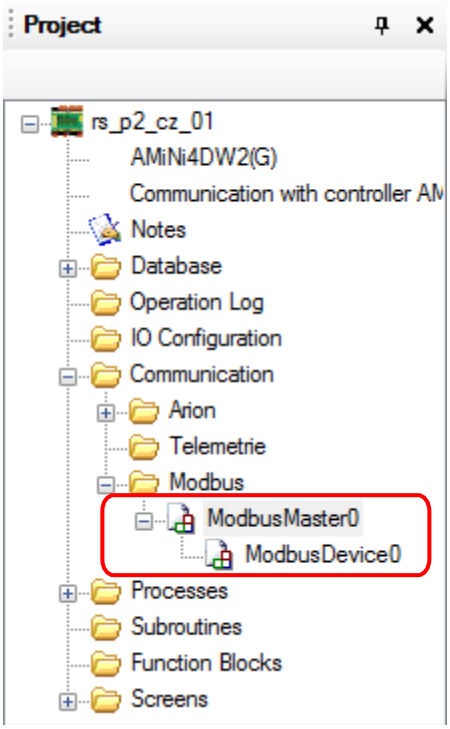

Fig. 32 – Definition **of AMR-OP70C** in the control system

Define the "ModbusMaster0" communication parameters in the object properties window. The communication parameters need to correspond to those defined in **AMR-OP70C**. In the "SerialPort" parameter, set the interface the device uses to communicate through the MODBUS RTU protocol. When requesting communication through RS485, it is necessary to set the property to 1.

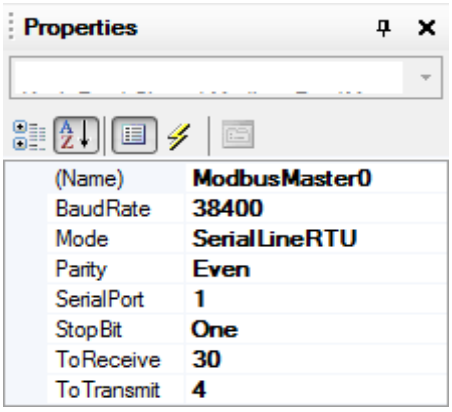

Fig. 33 – Properties of the "ModbusMaster0" object

In the properties window, set the "ModbusDevice0" address to the same address as **AMR-OP70C** has for the MODBUS RTU network. Set the `ClientLabel` property to a different value from "-1" for later evaluation of communication with **AMR-OP70C**.

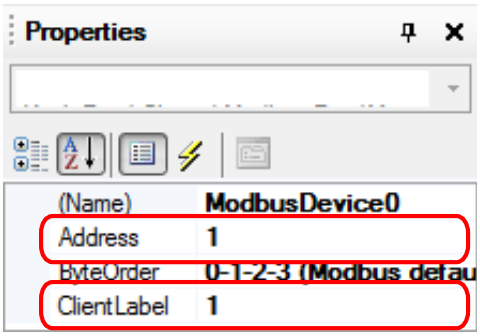

Fig. 34 – Properties of the "ModbusDevice0" object

Double-clicking the "ModbusDevice0" object opens a tab – use the tab to define registers for communication into / out of **AMR-OP70C** as defined in chapter 3.3.2 "Definition of variables for data exchange with the control system". This guide will use the recommended method of communication between the control system and **AMR-OP70C** (see chapter 4.2.2 "Method of communication with AMR-OP70C"). For the purposes of the recommended method of communication, it is also necessary to define the communication of the system register no. 8.

ModbusDevice0

Holding registers | Input registers | Coils | Discrete inputs

Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Write priority	Write function	Label
8 (40009)	8 (40009)	1	Flags	Low	--manual--	normal Modbus	1008
100 (40101)	101 (40102)	2	T_Meas	--manual--	--manual--	normal Modbus	
102 (40103)	103 (40104)	2	CO2	--manual--	--manual--	normal Modbus	
104 (40105)	105 (40106)	2	OP_Mode	--manual--	--manual--	normal Modbus	1104
106 (40107)	107 (40108)	2	T_Set	--manual--	--manual--	normal Modbus	1106
108 (40109)	109 (40110)	2	CO2Limit	--manual--	Auto	normal Modbus	1108

Fig. 35 – Defining registers in the "ModbusDevice0" object

Other than the system register and the CO_2 threshold warning, all communication will be activated manually. Enter any value into the "Label" column of selected registers (see the example image). Use the corresponding values in the module for issuing communication or for evaluation of its state.

Note
The "Auto" priority setting is convenient to use in case there would be a temporary change of settings (e.g. from the control system display). – e.g. setting the threshold for the CO2 warning. The "Auto" priority is not recommended for registers with a mapped variable used in the periodic process. With the priority set to "Auto", every write into the mapped variable – even with the same value – also causes a request for a write into the peripheral. That creates unnecessary network traffic – an example would be the room temperature setpoint. In most cases, the room temperature setpoint.is the output value of the ***DayPlan(2)*** *module. Since the* ***DayPlan(2)*** *module is processed in a periodic process, the write request sets periodically along with the period of the process the module belongs to. That is why, in the example, the label of the write priority is set to "--manual--" for the room temperature setpoint.*

Other than the defined registers, the **SerialBusN** object also provides the MODBUS RTU network with a group of system holding registers.

Of the system registers, the register no. 8 (SystemStatus) can be used; the first three bits are already used for the following purposes:

- SystemStatus.0 – the controller has been restarted,
- SystemStatus.1 – a value was written in the controller into one of the variables mapped in a register in the **SerialBusN** object,
- SystemStatus.2 – communication failure between the controller and the control system.

The above-mentioned flags are reset by writing True into the same bit of the "SystemStatus" register. As soon as the control system notices that the value of one of the digital inputs registers was set to true, it should perform a corresponding procedure and then reset the flag by writing True into the same bit.

4.2.2 Method of communication with AMR-OP70C

There are two methods of communication between **AMR-OP70C** and the control system.

- Simplified method
- Recommended method

Simplified method
Reading of required values from **AMR-OP70C** occurs with the period set in the definition of the corresponding register or registers. Writing of required values into **AMR-OP70C** occurs with every write into a variable mapped – on the control system side – to the MODBUS RTU network.

In terms of programming of the control system, the stated communication type is very simple. However, it creates a needless network traffic – queries on the set/measured values even in cases when the controller values did not change. Using the simplified method is not recommended for an extensive MODBUS RTU network.

Recommended method
Communication with **AMR-OP70C** uses register no. 8 (SystemStatus). Request for reading from **AMR-OP70C** is entered based on the state of bits of the controller. Therefore, only register no. 8 is monitored periodically. Communication itself occurs only based on the state of bit no. 1. Reading from **AMR-OP70C** occurs based on the state of the bit.

Below, this application note will deal with the recommended method of communication.

4.2.3 Designing a programme for communication with AMR-OP70C

Place the **MdbmReqSt** module into the periodic process – it is used to check whether the controller is communicating system register no. 8. Register no. 8 is used to check whether a change occurred in one of the variables mapped in the **SerialBusN** in register defined in the controller. Read registers based on the flag for writing into a variable in the **SerialBusN**.

Write the required value only when a change was detected.

```
// communication state check
MdbmReqSt 1, 1008, OP_R_Trans, NONE

If not (OP_R_Trans.0)  // in case of no communication taking place
     If Flags > 0  // if the change was entered via the controller
          MdbmReqSt 1, 1104, OP_Read, NONE  // the last register communication state
          If not(OP_Read.0)  // in case the last register is not being communicated
               // if the last register was communicated successfully and is
               // acknowledged
               If OP_Read.1 and @Acknowledge
                    Let Flags = 7  // system register resets
                    MdbmWrite 1, 1008, NONE
                    Let @Acknowledge = false // resets the flag of changes done from
                                             // the controller
               Else
                    // reading the measured temperature, CO2 concentration and the
                    // mode
                    Let @Acknowledge = true // sets the flag for changes done via the
                                            // controller
                    MdbmMark 1, 3, 100, 6, MarkRes
               EndIf
          EndIf
     EndIf

EndIf

// checks whether the system changed the temperature setpoint
Let @SetPointTmp = (T_Set > (T_Set_Old + 0.1)) or (T_Set < (T_Set_Old - 0.1))

// if the system changed a value, or the controller restarted and the acknowledge
// flag is false
If (@SetPointTmp or bool(Flags & 5))
     MdbmWrite 1, 1106, NONE
     Let T_Set_Old = T_Set
EndIf
```

5 Extending application functions for AMR-OP70C

5.1 Setting communication parameters for AMR-OP70C on the display

Setting the **AMR-OP70C** communication parameters is possible (through the **`SerialBusN`** object parameters) directly from the screen window.

This example is available in this application note attachments – in the op70c_p1_en_xx.dsox file.

Caution
The communication parameters are located in an EEPROM memory with a limited number of writes!

In this case, it is required to implement the possibility for dynamic change of communication protocol and dynamic change of communication parameters.

Define 3 new screens for the options to define all communication parameters – see the following picture.

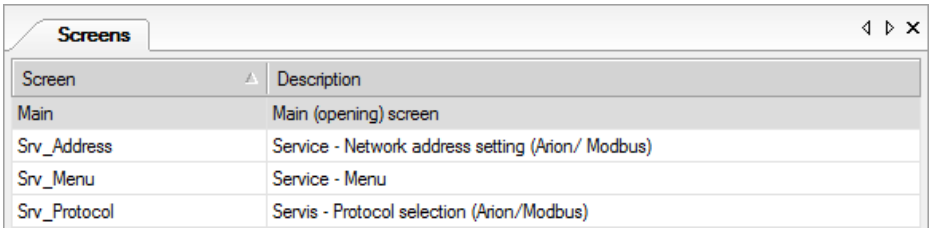

Fig. 36 – Screens for communication parameters settings

5.1.1 Protocol type choice screen

To choose the type of protocol used for data exchange between **AMR-OP70C** and the superior system (e.g. any control system made by AMiT), use the “Srv_Protocol” screen. Programme the choice itself by dragging a **`RadioButton`** from the “Toolbox” window (“General” section) onto the screen. Double-click the **`RadioButton`** element to open the window with definitions of individual items of the element. Set it up according to the following picture.

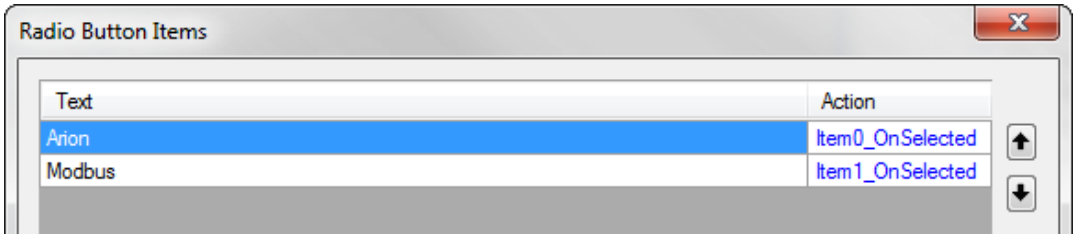

Fig. 37 – Setting the items of the **`RadioButton`** element

After confirming the preferences by pressing ***“OK”***, the scripting part of the screen opens with predefined **`RadioButton1_Item0_OnSelected`** and **`RadioButton1_Item1_OnSelected`** events. These objects will not be used in the application. Delete them and return to the screen design.

For confirming the selection of the protocol and returning to the **`MenuScreen`** containing screen, use the **`Button`** element in the “Toolbox” window (“TouchScreen” section). Drag the element onto the screen and double-click it to open the scripting part of the screen with a predefined **`Button1_OnButtonDown`** event. Programme the event so that the communication protocol type choice is saved and the controller returns to the **`MenuScreen`** containing screen. Based on the

choice of the **RadioButton** element, this event will save either 0 or 1 into the **SerialBusN.ProtocolMode** parameter. Regarding the **SerialBusN** object, 0 means the ARION communication protocol and 1 means the MODBUS RTU communication protocol. At the same time, programme the event to also redirect the user to a different screen. The resulting script should look like this:

```
event Button1_OnButtonDown()
     SerialBusN.ProtocolMode = RadioButton1.SelectedIndex;
     Srv_Menu.Show();
end;
```

Note

*Do not define writing of **SerialBusN.ProtocolMode** within the **ItemX_OnSelected** event of the **RadioButton** element as it would lead to unnecessary writes into the EEPROM memory should the user frequently switch between the communication protocols. That is why it is desirable to perform the write into the **SerialBusN.ProtocolMode** variable only when leaving the screen through the protocol type choice confirmation.*

Set the appearance of the **Button** element in the preferences window (appears after clicking the element) according to the following picture.

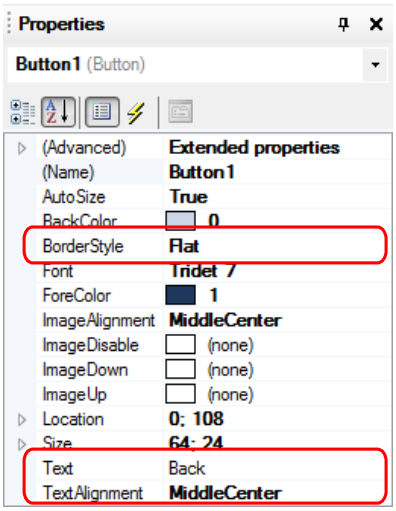

Fig. 38 – Settings of the **Button** element

Next, place a **Label** element into the upper part of the screen. It is located in the “Toolbox” window in the “Basic” section. Double-click the element and type in “Protocol”.

The resulting screen for communication protocol type choice should look like this.

Fig. 39 – Communication protocol type choice screen

Lastly, add a script into the "OnOpen" event of the screen that shows the currently selected communication protocol after the screen is opened. Add the following script into the **OnOpen** event of the screen.

```
RadioButton1.SelectedIndex = SerialBusN.ProtocolMode;
RadioButton1.Refresh();
```

5.1.2 AMR-OP70C address settings screen

To set the address **AMR-OP70C** will use in communication with the network, use the "Srv_Address" screen. The range of usable addresses will be set based on the chosen communication protocol.

Prior to programming the **AMR-OP70C** address setting, copy (**Ctrl+c**) the "Back" button and the "Protocol" **Label** from the "Srv_Protocol" screen and paste (**Ctrl+v**) them on the "Srv_Address" screen (on the same place). Change the "Protocol" text of the **Label** element to "Address".

Setting the address itself will be done by holding one of the two buttons placed onto the screen from the "TouchScreen" section of the "Toolbox" window. These buttons will increment/decrement the address. Furthermore, place onto the screen a **NumericView** element (without a linked variable). Organise and set the elements according to the following picture.

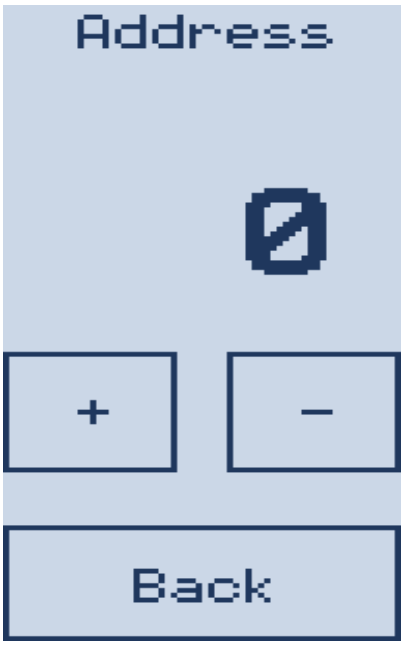

Fig. 40 – Address setting screen

As with the "Srv_Protocol" screen, do not save the chosen address into the **SerialBusN.Address** property right after it is set but rather when the user leaves the screen. To make the *„+"* and *„-"* buttons operable, use their **OnButtonPress** event. To go to the events associated with individual elements, press the ⚡ button in the "Properties" window of the given element. If the "Properties" window lists the properties of the desired **Button**, clicking the ⚡ button displays the following list of events.

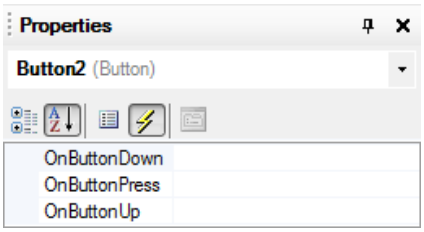

Fig. 41 – **Button** element properties

Left-clicking the empty field next to the **OnButtonPress** event displays the … button. Clicking … opens the scripting part of the screen and creates a corresponding event (**OnButtonPress**). Use this event to programme the incrementation of address **AMR-OP70C** according to the selected protocol. The resulting operation code for tapping and holding the button should look like this:

```
event Button2_OnButtonPress()  // "+" button operation
     If SerialBusN.ProtocolMode.0 then  // MODBUS protocol is chosen
          If NumericView1.Value < 247 then
               NumericView1.Value = NumericView1.Value + 1;
          Else
               NumericView1.Value = 1;
          EndIf;
     Else  // ARION protocol is chosen
          If NumericView1.Value < 63 then
               NumericView1.Value = NumericView1.Value + 1;
          Else
```

```
                NumericView1.Value = 1;
            EndIf;
        EndIf;
        NumericView1.Refresh();
end;
```

The process is similar for the address decrementing button. The decrementing button operation code should look like this.

```
event Button3_OnButtonPress()  // "-" button operation
        If SerialBusN.ProtocolMode.0 then  // MODBUS protocol is chosen
            If NumericView1.Value > 1 then
                NumericView1.Value = NumericView1.Value - 1;
            Else
                NumericView1.Value = 247;
            EndIf;
        Else  // ARION protocol is chosen
            If NumericView1.Value > 1 then
                NumericView1.Value = NumericView1.Value - 1;
            Else
                NumericView1.Value = 63;
            EndIf;
        EndIf;
        NumericView1.Refresh();
end;
```

In the next step, programme the operation of the "Back" button. Use the **OnButtonDown** event of the "Back" button – first save the set address value into the **Address** property of the **SerialBusN** object and then insert the code for leaving the screen for the "Srv_Menu" screen. The resulting code should look like this:

```
event Button1_OnButtonDown()  // "Back" button operation
        SerialBusN.Address = NumericView1.Value;
        Srv_Menu.Show();
end;
```

Lastly, add a script into the **OnOpen** event of the screen that shows the currently chosen **AMR-OP70C** address after the screen is opened. Add the following script into the **OnOpen** event of the screen.

```
NumericView1.Value = SerialBusN.Address;
NumericView1.Refresh();
```

5.1.3 Communication speed settings screen

To set the speed **AMR-OP70C** will use to communicate with the network, define a "Srv_Speed" screen by duplicating the already existing "Srv_Protocol" screen.

To do so, select the **Function/Duplicate screen** option in the screen toolbar.

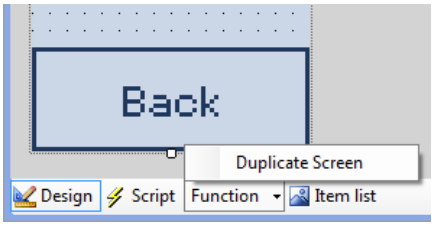

Fig. 42 – Duplicate screen function

After duplicating the screen, change the **Label** element text to “Speed”. Set the **RadioButton** element items (after double-clicking the element) according to the following picture.

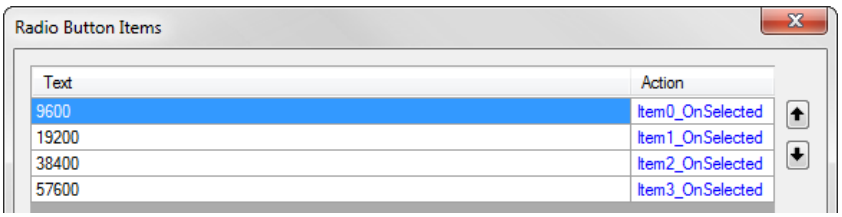

Fig. 43 – Setting the items of the **RadioButton** element

After confirming the preferences by pressing *“**OK**”*, the scripting part of the screen opens with predefined **RadioButton1_Item0_OnSelected**, **RadioButton1_Item1_OnSelected**, **RadioButton1_Item2_OnSelected** and **RadioButton1_Item3_OnSelected** events. These events will not be used in the application. Delete them and return to the screen design.

Adjust the script of the **Button1_OnButtonDown Button** event according to the following picture.

```
event Button1_OnButtonDown()
     If  RadioButton1.SelectedIndex == 0 then
          SerialBusN.BaudRate = 9600;
     Else
          If  RadioButton1.SelectedIndex == 1 then
               SerialBusN.BaudRate = 19200;
          Else
               If  RadioButton1.SelectedIndex == 2 then
                    SerialBusN.BaudRate = 38400;
               Else
                    SerialBusN.BaudRate = 57600;
               EndIf;
          EndIf;
     EndIf;
     Srv_Menu.Show();
end;
```

Set the **RadioButton** object properties according to the following picture.

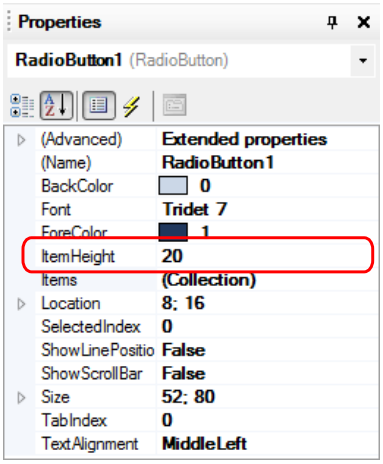

Fig. 44 – Settings of the **RadioButton** element properties

The resulting screen for communication speed settings should look like this.

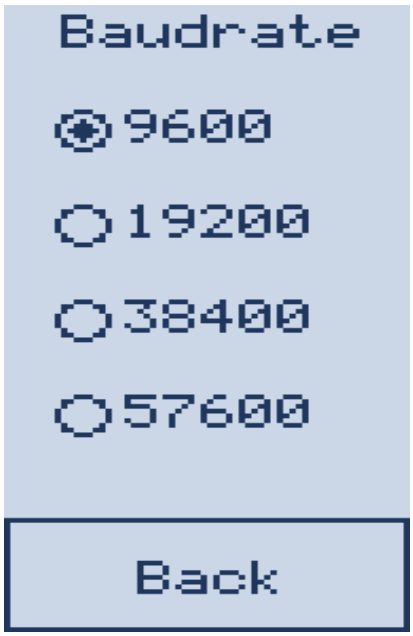

Fig. 45 – Communication speed settings screen

Edit the **OnOpen** event script as follows.

```
If SerialBusN.BaudRate == 9600 then
     RadioButton1.SelectedIndex = 0;
Else
     If SerialBusN.BaudRate == 19200 then
          RadioButton1.SelectedIndex = 1;
     Else
          If SerialBusN.BaudRate == 38400 then
               RadioButton1.SelectedIndex = 2;
          Else
               RadioButton1.SelectedIndex = 3;
          EndIf;
     EndIf;
EndIf;
```

5.1.4 Parity settings screen

To set the parity **AMR-OP70C** will use to communicate with the MODBUS RTU network (parity cannot be changed in ARION network), define a “Srv_Parity” screen by duplicating the already existing “Srv_Protocol” screen.

To do so, select the **Function/Duplicate screen** option in the screen toolbar.

After duplicating the screen, change the **Label** element text to “Parity”. Set the **RadioButton** element items (after double-clicking the element) according to the following picture.

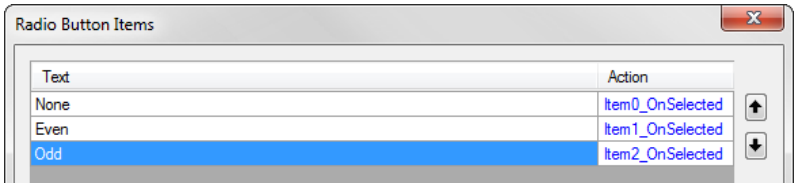

Fig. 46 – Setting the items of the **RadioButton** element

After confirming the defined items by pressing "***OK***", the scripting part of the screen opens with predefined **RadioButton1_Item0_OnSelected**, **RadioButton1_Item1_OnSelected** and **RadioButton1_Item2_OnSelected** events. These events will not be used in the application. Delete them and return to the screen design.

Adjust the script of the **Button1_OnButtonDown Button** event according to the following picture.

```
event Button1_OnButtonDown()
      SerialBusN.ModbusParity = RadioButton1.SelectedIndex;
      Srv_Menu.Show();
end;
```

Set the **RadioButton** object properties according to the following picture.

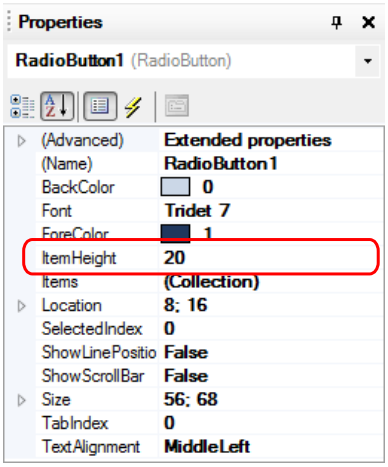

Fig. 47 – Settings of the **RadioButton** element properties

The resulting screen for communication speed settings should look like this.

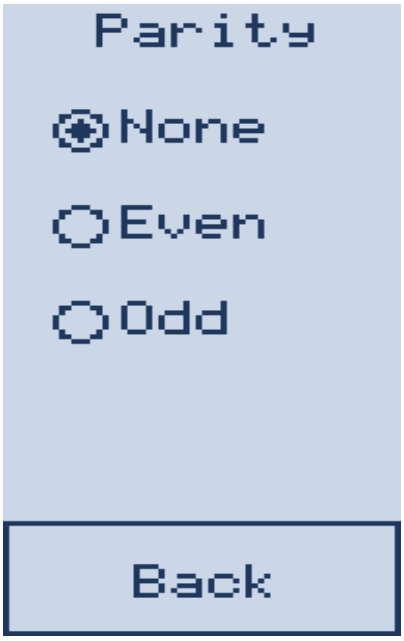

Fig. 48 – Parity settings screen

Parity settings is possible only for a MODBUS RTU based communication. Programme the script to show the parity setting only when the MODBUS RTU protocol is selected (and add text informing the user about it). Use the **MultilineLabel** element as the warning text will be longer than one line. Drag it on the screen and edit the text to "Only for Modbus". Additionally, add the following code to the **OnOpen** event of the "Srv_Parity" screen.

```
If SerialBusN.ProtocolMode.0 then  // if the MODBUS protocol is chosen
     MultilineLabel1.Visible = false;  // hides text
     RadioButton1.Visible = true;  // parity setting is visible
     RadioButton1.Enabled = true;  // parity setting is possible
     RadioButton1.SelectedIndex = SerialBusN.ModbusParity;  // shows current parity
                                                            // settings
Else  // if ARION is chosen
     MultilineLabel1.Visible = true;  // shows text
     RadioButton1.Visible = false;  // hides parity setting
     RadioButton1.Enabled = false;  // parity setting is disabled
EndIf;
Srv_Parity.Refresh();
```

The code guarantees that the warning displays when the ARION protocol was chosen, see the following picture. Otherwise, the parity settings will be displayed (the **RadioButton** element will display current parity).

Fig. 49 – The screen with parity settings for the ARION protocol

5.1.5 Service menu screen

The “Srv_Menu” page will be used to navigate between the individual settings pages. Open it and drag onto it the **`MenuScreen`** element from the “General” section of the “Toolbox” window. Afterwards, double-click it and set the individual items of the properties window according to the following picture.

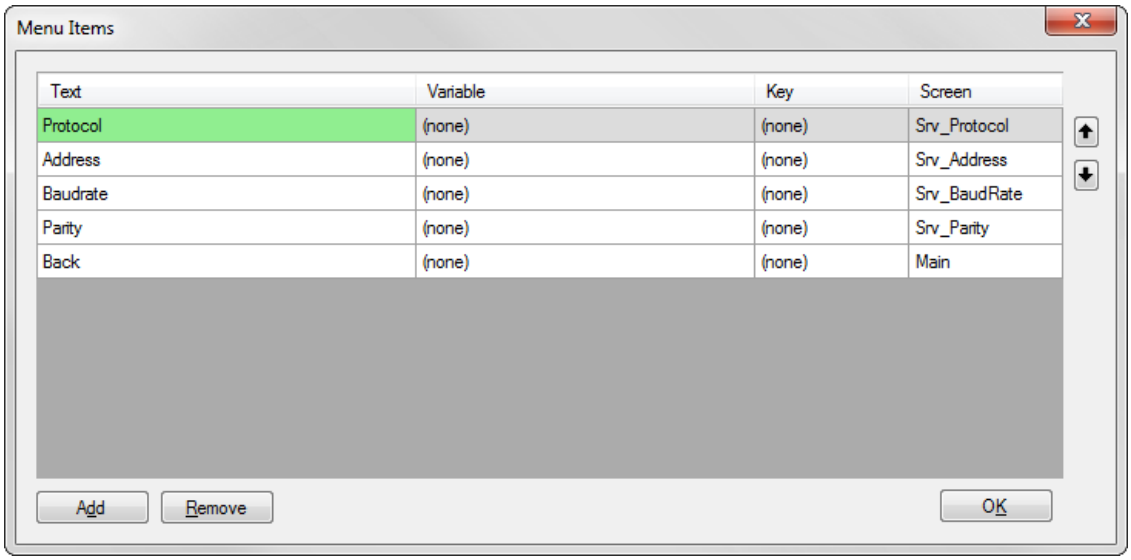

Fig. 50 – Setting the items of the **`MenuScreen`** element

Adjust the final appearance of the **`MenuScreen`** element in the “Properties” window according to the following picture.

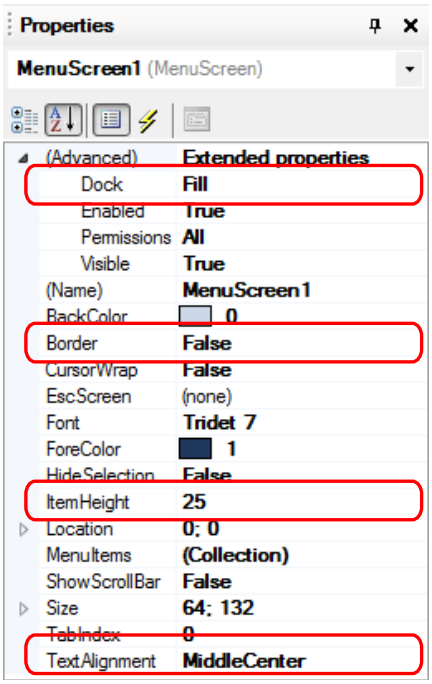

Fig. 51 – Settings of the **`Menu`** element

After changing the appearance, the “Srv_Menu” screen should look like this:

Fig. 52 – Service menu screen

5.2 Displaying the control system time on AMR-OP70C

If displaying the control system time on **AMR-OP70C** is required, it is necessary to programme it on both **AMR-OP70C** and in the control system.

5.2.1 Adjusting the control system application (ARION)

Adjust the global ARION network settings in the control system (the "Properties" window). In the Arion0 network properties window, change the value of **`TimeBroadcast`** to True. That causes control system time-frames to be broadcast into the ARION network.

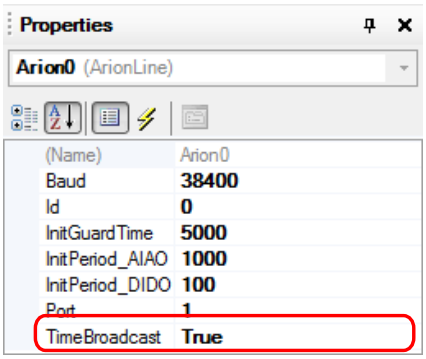

Fig. 53 – Time broadcast settings

5.2.2 Adjusting the control system application (MODBUS RTU)

In the control system, it is necessary to read the time value of the control system and define writing into the system registers 2 and 3 (Time registers) of the controller.

Use the **`GetTime`** module to get the system time of the control system. To send the time to the controller, use a special format (i.e. DB-Net). To receive the time value, define a Long variable and use it as a property of the **`Time`** module of the **`GetTime`**.

Place the module into the process with a period corresponding to the required period of writing into the controller. In case of the rs_p2_en_xx.dso example project, it is the "Proc01" process with a period of 5,000 ms.

```
GetTime DBNet_Time, NONE, NONE
```

To write the **`DBNet_Time`** variable value, use the "ModbusDevice0" table. Since there is no requirement for reading the register into the control system, the reading priority should be set to "--manual--". Set the writing priority to "Auto".

ModbusDevice0

Holding registers | Input registers | Coils | Discrete inputs

Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Write priority	Write function	Label
2 (40003)	3 (40004)	2	DBNet_Time	--manual--	Auto	normal Modbus	
8 (40009)	8 (40009)	1	Flags	Low	--manual--	normal Modbus	1008
100 (40101)	101 (40102)	2	T_Meas	--manual--	--manual--	normal Modbus	
102 (40103)	103 (40104)	2	CO2	--manual--	--manual--	normal Modbus	
104 (40105)	105 (40106)	2	OP_Mode	--manual--	--manual--	normal Modbus	1104
106 (40107)	107 (40108)	2	T_Set	--manual--	--manual--	normal Modbus	1106
108 (40109)	109 (40110)	2	CO2Limit	--manual--	Auto	normal Modbus	1108

Fig. 54 – Definition of writing into the system register 2 and 3 (Time)

Note

The aforementioned method of writing time values will be possible only for communication with a single controller. With the "Auto" priority chosen, after the time is written into the controller the write request flag is reset. When there are more occurrences of this line in other tables (other "ModbusDeviceX"), write operations related to them would never take place. The first table resets the write request flag which means that the request doesn't reach any subsequent tables. A solution is to use the "--manual--" write priority in conjunction with the ***`MdbmWrite`*** *module.*

5.2.3 Adjusting the application for AMR-OP70C

In **AMR-OP70C** it is sufficient to place a **`DateTimeView`** element onto a screen and to adjust the display format. For the purposes of this example, place the element onto the "Main" screen. Double-click it to tie it to the **`NowLong`** variable of the **`DateTime`** object. The screen should then look like this:

Fig. 55 – Time displaying screen

5.3 Navigating between screens

To programme the transitions from one screen to another – e.g. from the main screen to the correction settings screen, or from the main screen to the service screen with communication parameters settings – use the **`ImageMap`** element. Implement the transition so that when the user only touches the section of the screen highlighted in the picture below, the measured temperature correction settings screen opens. When the area is held for approx. 10 s, the service menu is displayed instead.

Fig. 56 – The screen transition area

Insert the **ImageMap** element into the required section and adjust its size so that it matches the size of the given area. With the "Add/Remove region" item in the properties window of the element, define a region that takes up all the space of the element.

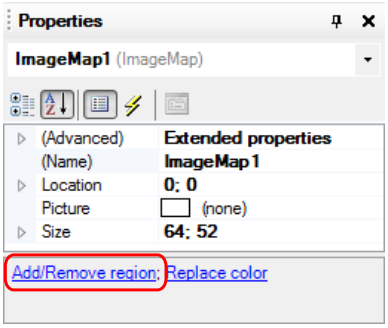

Fig. 57 – **ImageMap** element properties

In terms of programming the transition, use the **OnRegionUP0** event in the **ImageMap** element (to go to the correction settings) and **OnRegionPress0** (to go to the service menu).

Create a new screen named "Correction". It will be used to set the measured temperature correction (it will edit the **Temp_SensorCorrection** variable saved in the EEPROM memory via the **NumericEdit** element). Adjust the appearance of the screen according to the following picture.

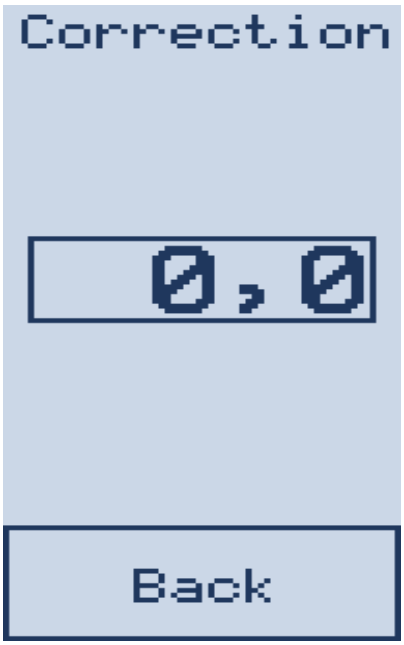

Fig. 58 – Correction setting screen

As was mentioned above, use the **OnRegionUp0** event of "Region0" of the **ImageMap** element located on the "Main" screen to transition to this screen. The event should then look like this:

```
event ImageMap1_OnRegionUp0()
     Correction.Show();
end;
```

After holding the screen for approx. 10 s, the screen should transit to the screen with the service menu ("Srv_Menu"). To do that, use the **OnRegionPress0** event of the **ImageMap** element (still the same **ImageMap** element, but the second event out of its two). At the same time, define an integer variable (e.g. named **Time**) that will be used to count the number of times the **OnRegionPress0** event was called. Pressing and holding "region0" of the **ImageMap** will call the **OnRegionPress0** every 200 ms. The transition to the service menu screen will occur only after region0 of the **ImageMap** element is held pressed for 10 s. Holding the region will increment the **Time** variable value until it reaches 50 (50 × 200 ms = 10 s.). As soon as the **Time** variable value reaches 50, reset it and transition to the service menu screen. The final code of the **OnRegionPress0** event should look like this:

```
event ImageMap1_OnRegionPress0()
     Time = Time + 1;
     If Time > 50 then  // 10 s pause for the press to take effect
          Time = 0;
          Srv_Menu.Show();
     EndIf;
end;
```

It is necessary to programme **Time** value resetting even into the **OnRegionUp0** event. It is required in case the user releases the button too soon and the service menu screen has yet to appear.

```
event ImageMap1_OnRegionUp0()
     Correction.Show();
     Time = 0;
end;
```

6 Appendix A

6.1 Using standalone communication objects

If there is no requirement for switching between ARION and MODBUS RTU, it is possible to use the following objects for communication through the given protocol:

- **ArionSlave,**
- **ModbusSlave**.

Each object requires a different programming attitude.

6.1.1 Object ArionSlave

The **ArionSlave** is substantially different from the **SerialBusN** in terms of sending data into the ARION network. The **SerialBusN** object maps so-called registers (types DInt or Real) from the ARION network. These registers can contain variables or properties of various objects. The number of registers the **SerialBusN** object provides into the ARION network is limited to 9.

The **ArionSlave** provides the following signals into the ARION network:

- 24 signals – type DI,
- 24 signals – type DO,
- 24 signals – type AI,
- 24 signals – type AO.

On the side of the control system, it is necessary to configure the **ArionSlave** object containing device into the “Arion0” table (see the Application note AP0025 – Communication in the ARION network – table definition) via the **ArionDevice** module (in the “DM” section of the “Toolbox”). DI or DO signals are consequently processed with the **ARI_DigIn** or **ARI_DigOut** module.

The analogue values are transmitted as 14-bit integers. Based on the sign of the value, it is possible to transmit either a value in the range of 0 to 16,383 (unipolar) or -8,192 to 8,191 (bipolar). The polarity of the value needs to be defined in the control system through the “Arion0” table in the **ArionDevice** module definition using the **ModeAI** and **ModeAO** modules. The data is then processed as an integer value using the **ARI_NumAI** or **ARI_NumAO** module.

As the text above suggests, it is necessary to transform a potential Real variable to an Integer in case it is required to send such value into the control system. Only then it is possible to send it through the AI or AO channels.

When there is a requirement to transfer a value from an analogue input to the control system or to transfer a value from the system to an analogue output, it is necessary to use the **ArionIO** object in any **AMR-xxx**. In case the **ArionIO** object is used on the **AMR-xxx** side, the data provided by the **ArionIO** object on the control system side is processed by the **ARI_AnIn** or **ARI_AnOut** module.

It is suitable to use the **ArionSlave** object in cases when there is a requirement for sending an analogue value that is directly sent to analogue outputs or that is directly read from the analogue input. In other cases (transferring variables or Real type object properties) it is better to use the **SerialBusN** object.

More examples and further information is in the Application note AP0054 – AMREG communication with AMiT control systems (ARION).

6.1.2 Object ModbusSlave

The ModbusSlave object is in its use similar to the **`SerialBusN`** object. As opposed to the SerialBusN object, the ModbusSlave object allows the use of holding registers as well as input registers, coils or digital inputs for mapping data into the MODBUS RTU network. Since DetStudio v2.1, it is possible to create a reference of a MODBUS object to the required property or variable straight in the ModbusSlave table. Alternatively, it is possible to create a register that will be used in the application code.

More examples and further information is in the Application note AP0055 – AMREG communication with AMiT control systems (MODBUS RTU).

7 Appendix B

7.1 Operation of AMR-OP60 buttons

The on-wall controller allows operation control via a set of 4 buttons. The buttons are mapped to the display elements as ***Esc***, ***Tab***, ***F1*** and ***Enter*** – see the following picture.

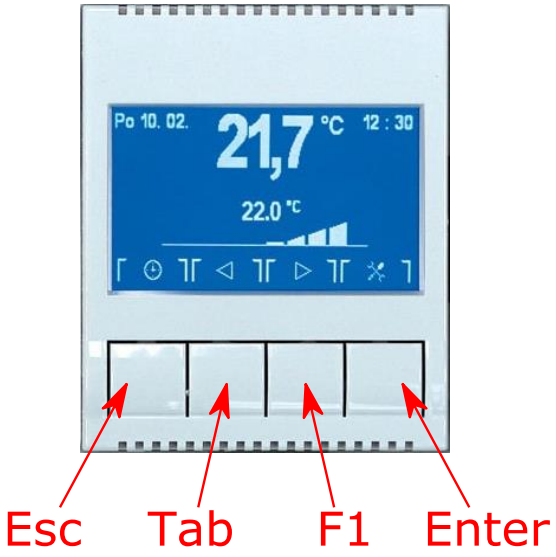

Fig. 59 – Description of **AMR-OP60** buttons

There are several methods of control of the on-screen buttons:

1. by using the **`Keyxxx`** buttons (suitable for changing modes with signalisation of the currently selected mode),
2. by using the **`OP60kbd`** element with predefined buttons (suitable for entering required values via the **`NumericEdit`** or **`NumericUpDn`** elements),
3. by using the **`OP60kbd`** element with user-defined buttons (suitable for use with elements with special usage – e.g. **`RadioButton`**).

A programme with the aforementioned methods of use is included in this Application note – in the op60_p1_en_xx.dsox file.

7.1.1 Keyxxx elements usage

It is suitable to use the **`Keyxxx`** elements when it is required to change the mode of the device as well as showing the currently set mode. The **`KeyScreen`** element falls into the same category – it can be used to transition between screens.

In the example file op60_p1_en_xx.dsox, there is a “Main” screen with four **`Key`** elements in use (K_1, K_2, K_3 and K_4). The element “K_1” switches the room mode, the elements “K_2” and “K_3” change the room temperature setpoint correction and the element “K_4” switches between the user and service settings.

Operating the Key element (“K_1”)

“K_1” will change the required room mode (switching between the day plan, comfort and energy saving). The currently selected mode will be displayed via the **`CaseImage`** element above the button.

CaseImage ("CI_RMode")

It is necessary to define the images used for the **CaseImage** modes. Do so via the **Images** property in the properties of the element. In the **Variable** property of the **CaseImage** element, select **SerialBusN.Room_Mode**.

Key ("K_1")

In the code, use the "OnKeyDown" event to operate the element.

```
event K_1_OnKeyDown()  // room mode change
     if SerialBusN.Room_Mode < 2 then
          SerialBusN.Room_Mode = SerialBusN.Room_Mode + 1;
     else
          SerialBusN.Room_Mode = 0;
     endif;
     CI_RMode.Refresh();
end;
```

Operating the Key elements ("K_2" and "K_3")

These elements will change the value of the room temperature setpoint correction in the range of -100 to 100. The currently chosen value will be displayed via the **CaseImage** element above the button.

CaseImage ("CI_Corr")

It is necessary to define the images used for the **CaseImage** correction levels. Do so via the **Images** property in the properties of the element. In the **Variable** property of the **CaseImage** element, select **SerialBusN.Correction**.

Key ("K_2" a "K_3")

In the code, use the "OnKeyPress" event to operate these elements. Displaying the icons above the buttons can be done via the **Image** element.

```
event K_2_OnKeyPress()  // temperature setpoint correction (lowering the setpoint)
     if SerialBusN.Room_Mode == 0 then  // changes the correction only with the
                                        // Day plan chosen
          if SerialBusN.Correction > -100 then  // if the correction is above -100
               SerialBusN.Correction = SerialBusN.Correction - 20;  // the value is
                                                                    // deducted
          else
               SerialBusN.Correction = -100;
          endif;
          CI_Corr.Refresh();
     endif;
end;

event K_3_OnKeyPress()  // temperature setpoint correction (increasing the setpoint)
     if SerialBusN.Room_Mode == 0 then  // changes the correction only with the
                                        // Day plan chosen
          if SerialBusN.Correction < 100 then  // if the correction is below 100
               SerialBusN.Correction = SerialBusN.Correction + 20;  // the value is
                                                                    // added
          else
               SerialBusN.Correction = 100;
          endif;
          CI_Corr.Refresh();
     endif;
end;
```

Operating the Key element ("K_4")

The element "K_4" will make it possible to:

- display the sensor correction user settings screen (short press),
- display the service screen with communication selection (long press).

Displaying the icons above the button can be done via the **Image** element.

Since the button will be used for navigation to two screens, the **KeyScreen** cannot be used to implement the transition. The required function needs to be programmed using the "OnKeyPress" (long press) and "OnKeyUp" (short press) events of the Key element. "OnKeyDown" cannot be used as it is always called only before the "OnKeyPress".

Define the screens for the user settings of correction (for example named "U_Correction") and the communication protocol selection screen (for example named "S_Protocol").

The operating script of those screens should look like this:

```
// displaying user settings
event K_4_OnKeyUp()
     Time = 0;  // resetting the counter for a long press
     U_Correction.Show();
end;

// displaying system settings
event K_4_OnKeyPress()
     Time = Time + 1;  // with every event, the counter value increments
     if Time > 50 then  // if the counter value reached the required value
          Time = 0;  // the counter resets
          S_Protocol.Show();  // and the protocol selection screen is displayed
     endif;
end;
```

7.1.2 Using OP60kbd with predefined buttons

The **Op60kbd** element is used with the predefined keys, for example, in the sample application code of the "U_Correction" screen. The default settings of the element were adapted – the **ImageButton2** and **ImageButton3** images were deleted as the "Tab" and "F1" are not in use. More detailed description of the predefined keys function of the **Op60kbd** element is in the DetStudio Help (Screen design section).

To return back to the previous screen, use the "OnButton1KeyDown" event of the **OP60kbd** element. The code should look like this:

```
event OP60kbd1_OnButton1KeyDown()
     Main.Show();
end;
```

7.1.3 Using OP60kbd with user-defined buttons

The **Op60kbd** element is used with the user-defined keys, for example, in the sample application code of the "S_Protocol" screen. The screen also uses the **RadioButton** button that allows the user to switch the on-wall controller communication protocols (ARION/MODBUS). **RadioButton** is not controlled by the keys available in the **Op60kbd** element in its default settings. It is therefore necessary to change the keys that cannot be used for controlling the **RadioButton** element for keys that are required by the element (see the DetStudio Help (Screen design section)). The change can be implemented by editing the **KeyCodeButtonX** property. It is therefore necessary to replace the "Tab" key with an "Up" key and the "F1" key for a "Down" key. Change the icons in the properties – **ImageButton2** and **ImageButton3**.

When the screen appears, the **RadioButton** element needs to display the current setting. To implement that, use the "OnOpen" event of the "S_Protocol" screen.

```
event S_Protocol_OnOpen()
     S_Protocol.FocusFirstControl();
     RadioButton1.SelectedIndex = SerialBusN.ProtocolMode;  // loads current setting
     RadioButton1.Refresh();
end;
```

To exit the screen without saving changes, use the first button ("Esc"). The operation code of its „OnButton1KeyDown" event should look like this:

```
event OP60kbd1_OnButton1KeyDown()
     Main.Show();
end;
```

To save the choice and exit the screen, use the fourth button ("Enter"). The operation code of its „OnButton4KeyDown" event should look like this:

```
event OP60kbd1_OnButton4KeyDown()
     SerialBusN.ProtocolMode = RadioButton1.SelectedIndex;
     Main.Show();
end;
```

8 Appendix C

8.1 AMR-OP70RHP operation with the Poseidon© interface

Among other things, the **AMR-OP70RHP** on-wall controller is equipped with a Poseidon interface. Further information on the Poseidon network operation is in the DetStudio / EsiDet Help and in the application note AP0051 “Communication in the Poseidon Wireless System”.

Caution
*The example can be used only for setting basic connections. If more complex configurations are required (DALI network configuration, configuration of time for opening/closing shutters or blinds, etc.), it is necessary to use the mentioned SW tool – Poseidon Assistant – in conjunction with HW (**AMR-CP2x**, **AMR-CP4x**, **P8 TR IP**, **P8 TR USB**) that allows Poseidon configuration.*

This example is available in this application note attachments – in the op70rhp_p1_en_xx.dsox file.

8.1.1 Designing the Poseidon operation code

To insert an object for Poseidon configuration into the **AMR-OP70RHP** application, follow similar steps as for the other protocols. Right-click the “Communication” folder in the “Project” window and select “Add object”. In the list of objects, select the main **`Poseidon`** object and the peripherals the controller is supposed to communicate with.

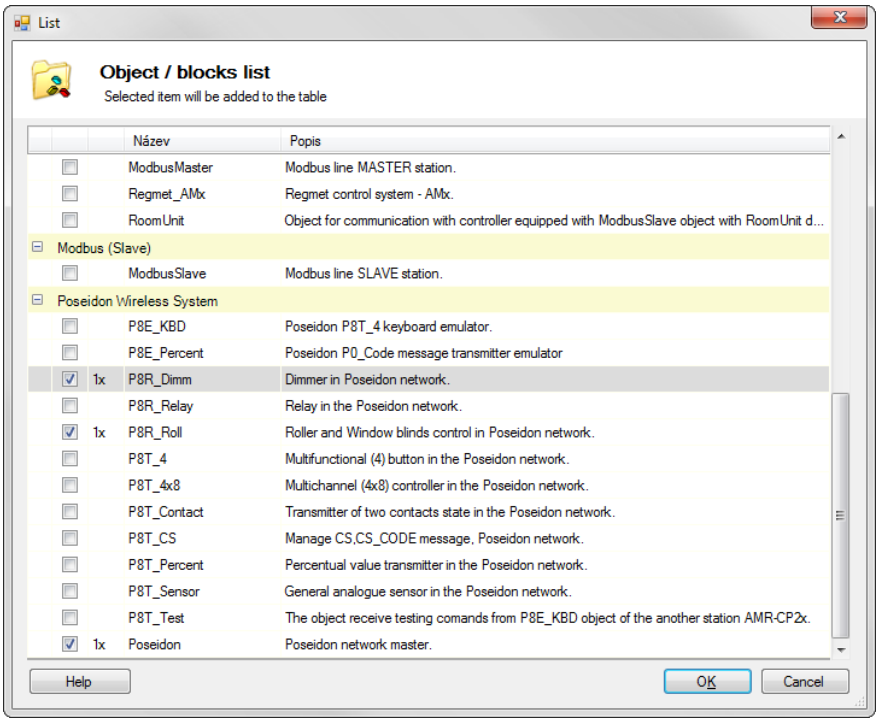

Fig. 60 – Selection of objects for the Poseidon network

The important properties of the P8x_xxx objects for making the network work:

- ID – Poseidon ID of the peripheral – stated on the peripheral label,
- InitDevice – a command for sending or receiving a special frame for establishing a connection,
- Error – the value of a potential error message.

Inserting the objects that manage the operation of peripherals within the Poseidon network causes the above-mentioned properties to be available for any object for communication with a Poseidon network peripheral. It is therefore possible to use them directly on the controller screens.

8.1.2 Screen for creating connections within the Poseidon network

To design a screen for creating connections within the Poseidon network, use the **NumericEdit** element for entering the device ID of the desired peripheral. It is convenient to use the **CustomFormat** property to display the peripheral ID value in hexadecimal format (as the ID is stated in hexadecimal format on the peripherals as well). Therefore, insert "%IX" into the **CustomFormat** property. To allow the controller to use A to F characters for the hexadecimal format, set the **SystemEditor** property to "NumericUpDn".

The command for creating the connection between the controller and the peripheral can be implemented via the **BitSwitchButton** element – add a link to the **InitDevice** property of the respective Poseidon network object into the **Variable** property. Change the text of **TextDown** and **TextUp** properties to "Pairing" and "Pair". Upon press, the button sends a special frame to establish the connection. At the same time, "Pairing" is displayed. This state lasts until the connection between the controller and the peripheral is established or until the command times out.

Communication failure can be displayed via the **NumericView** element – add a link to the **Error** property of the respective Poseidon network definition object into the **Variable** property.

For two receivers in the Poseidon network, the final screen (Landscape mode) should look like this:

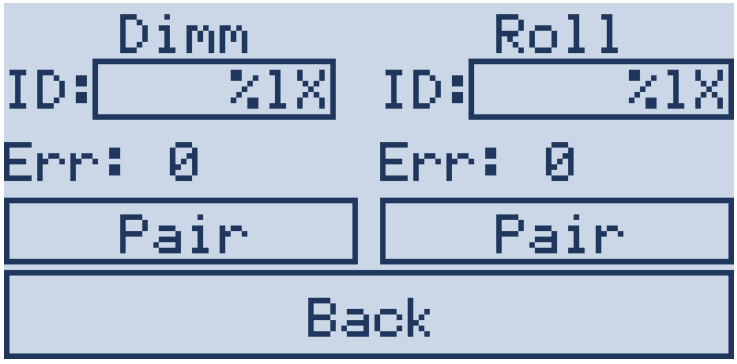

Fig. 61 – Screen for creating connections

8.1.3 Lighting settings screen

Should the user require lighting setting in precisely defined steps, use the **SelectButton** element. After dragging in onto the screen, define the individual item images of the **Items** property according to the following picture.

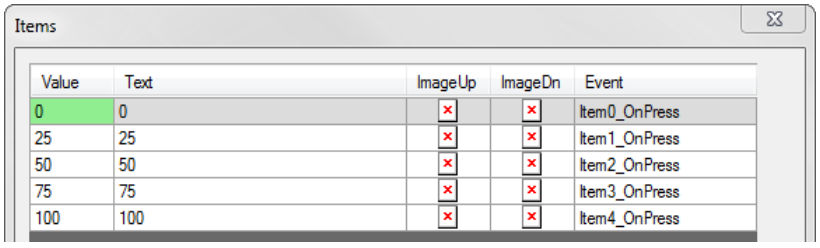

Fig. 62 – SelectButton items images

Implement the setting of the lighting levels, for example, through the **P8R_Dimm** object (analogue control of lighting levels). Insert the **OutX** property of the **P8R_Dimm** object into the **Variable** property of the **SelectButton** element.

Furthermore, place a Label element onto the screen and change its text to “Lighting”.

The currently set level of lighting can be loaded through the **ActualOutX** property of the **P8R_Dimm** object. Display the value via the **NumericView** element – add a link to the **ActualOutX** property into the **Variable** property.

In addition, define a transition to the parameter screen for establishing connections. Follow the same instructions as in chapter 5.3 “Navigating between screens”.

To implement the transition from the current screen to the screen for control of another Poseidon peripheral, use the **ButtonScreen** element. Insert a link to the desired screen into the element’s **GoToScreen** property and change the **Text** property according to the desired screen, e.g. “Blinds”.

The resulting screen should look like this:

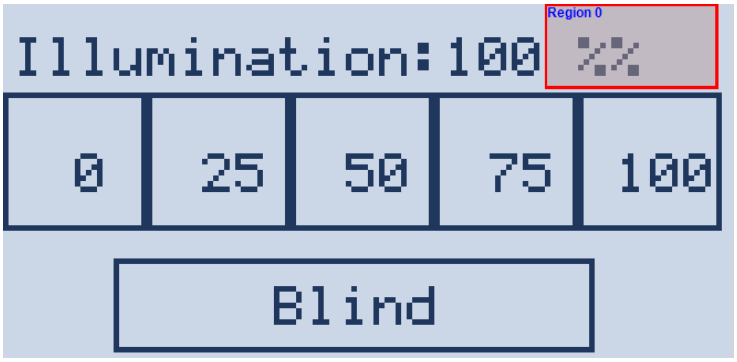

Fig. 63 – Lighting settings screen

8.1.4 Blinds settings screen

Create the screen by duplicating the lighting settings screen. Afterwards, change all text fields on the screen to match the new elements.

Change **Label** text to “Position”.

Add a link to the **Position** property of the **P8R_Roll** element into the **Variable** property of the **SelectButton** element.

The currently set position of blinds can be loaded through the **ActualPosition** property of the **P8R_Roll** object. Therefore, add a link to the **ActualPosition** property into the **Variable** property of the **NumericView** element.

To return to the lighting settings screen, insert a link to that screen into the **GoToScreen** property. Additionally, change the **Text** property accordingly, e.g. “Lighting” (transition to the lighting screen).

The resulting screen should look like this:

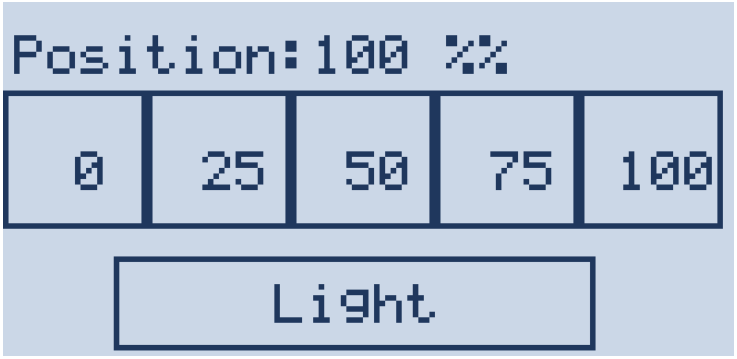

Fig. 64 – Blinds position settings screen

9 Technical support

The AMiT Technical Support Department provides all information regarding communication in ARION network. The Technical support is best contacted via e-mail at **support@amit.cz**.

10 Warning

In this document, AMiT, spol. s r. o. provides information as it is, and the company does not provide any warranty concerning the contents of this publication and reserves the right to change the documentation content without any obligation to inform anyone or any authority about it.

This document can be copied and redistributed under the following conditions:

1. The whole text (all pages) must be copied without making any modifications.
2. All redistributed copies must retain the AMiT, spol. s r. o. copyright notice and any other notices contained in the documentation.
3. This document must not be distributed for profit.

   The names of products and companies used herein may be trademarks or registered trademarks of their respective owners.