Communication in the Poseidon Wireless System

Abstract

A description of communication of stations made by AMiT with Poseidon wireless network peripherals made by ENIKA.CZ.

Author: Petr Latina, Zbyněk Říha
Document: ap0051_ap_en_003.pdf

Attachments

File contents: ap0051_ap_en_002.zip

poseidon_p1_en_002.dsox	Examples of algorithms for the operation of transmitters and receivers.

Contents

Revision history

Version	Date	Changes by	Changes
001	29. 07. 2014	Latina Petr	New document
002	27. 01. 2021	Říha Zbyněk	Updated according to DetStudio v 2.x, document structure changed.
003	12. 02. 2021	Říha Zbyněk	Links in chapters 4.4 and 7 edited.

Related documentation

1. Help for the DetStudio development environment – Help for IDE
   file: Ovladani_en.chm
2. Help tab for the DetStudio development environment – Help for EsiDet
   file: Esidet_en.chm
3. **AMR-CP24/01** – Communication unit – Operation manual
   file: amr-cp2401_g_en_xxx.pdf
4. **AMR-CP44/DM RevA** – Communication unit – Operation guide
   file: amr-cp4xdm_reva_g_en_xxx.pdf
5. **AMR-CP46/DM RevA** – Communication unit – Operation manual
   file: amr-cp4xdm_reva_g_en_100.pdf
6. **AMR-OP70RHP/xx** – Programmable on-wall controller – Operation manual
   file: amr-op70rhpxx_g_en_xxx.pdf
7. **AMR-OP71RHP/xx** – Programmable on-wall controller – Operation manual
   file: amr-op71rhpxx_g_en_xxx.pdf
8. **AMR-OP87/Pxx**– Graphic control HMI – Operation manual
   file: amr-op87pxx_g_en_xxx.pdf

1 Definitions of terms

Poseidon®
A wireless network operating at the frequency of 868 MHz; developed by ENIKA.CZ.

Poseidon system
A complex of peripherals communicating in the Poseidon wireless network.

Transmitter
A peripheral of the Poseidon system transmitting commands (e.g. press of a button, transmitted value of a physical quantity sensor – e.g. temperature).

Receiver
A peripheral of the Poseidon system processing the commands received from a transmitter.

ID
A unique identification number of each element in the Poseidon system.

EsiDet
A part of the DetStudio development environment used to create applications for AMiT programmable controllers.

2 Poseidon system basic properties and usage rules

The Poseidon wireless system offers some new options in the field of measurement and control. It is however necessary to take into account certain limitations and properties that do not occur in the classic “wired” automation.

2.1 Using the 868 MHz frequency band

The Poseidon system works in the **868 MHz public frequency band** and its use is subject to specific rules. The rules are defined in the General Authorisation No. VO-R/10/05.2014-3 on use of radio frequencies and on operation of short-range devices, issued by the Czech Telecommunication Office. Technical parameters show e.g. that the station on a frequency of 868 MHz with a maximum radiated power of 25 mW e.r.p. (Effective radiated power), can use the band with duty cycle ≤ 1.0 %. The duty cycle is the portion of time when the device is transmitting at any time in one hour. Put in layman’s terms, it is not possible to occupy the frequency with the transmitting signal indefinitely. That is why all elements of the Poseidon system including the communication unit provide that the duty cycle limit is met automatically and will not allow the user to occupy the band excessively, whether or not intentionally.

2.2 Wireless System Responses

As the previous paragraph shows, a radio channel cannot be viewed like a “wired” connection with an unlimited data stream. Transmitters of the Poseidon system transmit messages in a very short time interval, about 5 ms, immediately upon occurrence of a given event (pressing a button, sending the measured value of temperature, for example temperature, etc.). Conversely, there are repeated requests to read status of some peripherals (position of shutters, dimmer value) by the superior system using stations made by AMiT. To prevent channel overload, AMiT stations have a pre-set **time interval for repeated-status reading**. A similar situation with the time interval occurs with entering values into wireless elements (DALI ballasts, dimmers, shutters control, switch relay, etc.). To prevent repeated transmission of the same value, the AMiT station only sends the values after they change (turn on/off or change of value by more than 1). The communication modes and time sequence are described in more detail in further chapters.

2.3 Using control functions of the Poseidon system

The Poseidon system is designed to process basic control and regulation functions on the level of actual system elements, without the necessity of a superior system. For example, light control can occur directly between transmitters (on-wall buttons, keychains) and receivers (light switches, dimmers). The response to a button press is immediate, within the range of tens of milliseconds. The superior system still has the option to read the current status of the lighting (the light is on/off, current intensity), and furthermore, it is also able to control the receivers from the position of a superior function (central switch). That is why we recommend to **ALWAYS** use all pre-defined higher functions of elements in the Poseidon system that can be configured in the Poseidon Asistent programme, and to use the connection with a superior system for global control and monitoring of status and values of wireless elements.

The following are examples of tasks the Poseidon system processes on the level of transmitters and receivers:

- Controlling lights (switching on/off, dimming, scenes, etc.).
- Timer functions for switching lights on/off and output relays in general.
- Controlling roller blinds and shutters including rotation of specific slats.
- Lighting intensity control on the basis of measured level of outdoor light.
- Actions performed on the basis of motion detection from motion sensors.
- Using priority alarm signals for immediate device response (e.g. protecting the blinds from strong wind).

2.4 Unique device ID

Each peripheral in the Poseidon system has its **unique identification number (ID)** that is assigned to it during production and is not customisable by the user. The links between the peripherals are related to specific IDs and therefore when peripherals change, it is necessary to change these links as well. The unique ID increases safety of the wireless system operation and helps with identification of individual peripherals.

The ID of each Poseidon peripheral is stated on a sticker delivered along with the given peripheral. AMiT stations have the sticker next to the Poseidon interface connector. IDs can be also read via the Poseidon Asistent application (see chapter 4.3.1 “Poseidon Asistent SW”).

2.5 Security and reliability

When transmitting in the given frequency band, data is freely accessible in the entire transmission range. That is why the Poseidon system uses several types of coding and data transmission security measures in its protocol; to increase the transmission reliability and mainly to prevent unauthorised interference into the station controls. Furthermore, the Poseidon Asistent parametrisation programme also includes procedures and functions that allow “invisibility” of given peripherals in the projects, and password protection against unauthorised use. We recommend you pay careful attention to this area.

2.6 Reach

Each element of the Poseidon system is designed to provide reliable data transmission into the range of **150 m in free space**. In case of using the system accessories to the peripherals, we can guarantee range of up to 3 km. Actual range is substantially lower in practice, and it depends on the number of obstacles and material through which the signal travels as well as on the intensity of potential local interference from other sources. The function of **signal repeater** can be used conveniently in each receiver powered from the network.

3 AMiT station in the Poseidon system

AMiT station in the Poseidon system may act simultaneously as transmitter and receiver, i.e. it may transmit commands, queries on the device output status or receive information from the network e.g. in the form of analogue values (temperature, humidity).

AMiT stations can:
- mediate the data transfer between the superior system and the Poseidon system,
- act as the superior system for the Poseidon network.

3.1 Communication type

Button pressing emulation
Using the button pressing emulation, the AMiT station is able to send a command in the same way as a physical transmitter (a key-chain button etc.). The AMiT station therefore acts directly as an ENIKA peripheral. Use the Poseidon Asistent SW (see chapter 4.3.1 “Poseidon Asistent SW”) to set the reaction of a selected peripheral to a button press emulated in an AMiT station.

Commanding the receivers
Output status of peripherals that allow receiving commands from AMiT stations can be changed. The difference between commanding mode and the button press emulation-based control is that the controlled peripheral can only be commanded in the “basic mode”. Peripherals in the “basic mode” perform functions predefined by the manufacturer, e.g. a peripheral with a relay output would perform the On/Off function. In the commanding mode, it is impossible to use Poseidon Asistent to set a reaction of a selected peripheral to a command issued by an AMiT station.

Receivers and queries
Peripherals that allow command receiving also allow sending the current output status (or value) upon a query or automatically. Again, this functionality depends on the firmware in a specific peripheral.

Processing received messages
During communication, messages received from transmitters are processed. The received messages may contain the following types of values:
- analogue value (e.g. temperature, humidity, etc.),
- percentage value ranging from 0 % to 100 %,
- binary value (e.g. status occupied = True for a movement sensor or an output state of a relay = True),
- button status on the physical transmitter (e.g. pressed = True).

3.2 Communication time sequence, priority

The following text describes the communication time sequences and priorities in commands and queries in terms of the AMiT station. Time sequences are selected on the basis of a recommendation from the Poseidon communication creator, the ENIKA.CZ company.

Commanding
Commands have the **highest priority**; they may entail a button press emulation or an issued command e.g. to control a relay On/Off. This type of request gets performed practically immediately. If more requests occur, they enqueue and get performed one by one.

The time interval between executing two or more commanding requests is **100 ms**. If there is no response from the commanded peripheral after sending a specific command, a Timeout is assessed and for **5 s** no other command is transmitted (the devices wait for the “air to clear”). Then, another command can be transmitted.

Caution!
The commanding mechanism watches for a change in the value to be sent automatically. If there is no change by value 1 in comparison to the previous value, the command is not transmitted. That ensures the same value will not be transmitted over and over again.

Queries
If all commands are executed, individual peripherals are queried about their status or output values. Queries have a **lower** priority. A "**List**" of queried peripherals is created automatically, in the order the individual objects enter into the project for the AMiT station, see chapter 4.2 "Definition of peripherals".

The query requests are transmitted in sequence for individual peripherals in the same order they enter the list of queried devices. After a query is performed for the last peripheral, the query request cycle starts again from the first peripheral on the list.

The time interval between individual queries is **1 s**. If there is no response from the peripheral after sending a specific query, Timeout is evaluated and for the peripherals wait for **5 s** for the "air to clear". Then, another query can be transmitted.

Note:
At the time of publication of this application note, the Poseidon protocol allows for one query per one output of a specific peripheral; i.e. if a queried peripheral has 8 relay outputs, its query will take 8 s.

Messages from transmitters
Similar to a list of queried peripherals, there is also a list of peripherals from which a message containing e.g. analogue value is expected. This list is created automatically upon creating a project for the AMiT station. These messages are generated by pressing a button or by a temperature or humidity sensor or by a contact state transmitter, etc. The messages are ready for immediate use after recording, e.g. in the form of a measured temperature value.

4 Creating a communication application in the Poseidon system

In order for the AMiT station to communicate in the Poseidon system successfully, an application must be created (according to technical requirements) in **DetStudio**/**EsiDet**. It is generally recommended to send queries between peripherals without using the AMiT station as an intermediary (sending a query first to the AMiT station which then forwards it somewhere else). This leads to unnecessary latency when processing queries. A more effective approach is to create links directly between the peripherals – the AMiT station only receives reports on the measured temperature and the temperature setpoint; it can also affect the state of the controlled peripherals.

All examples from this chapter are also included in the attachment ap0051_ap_en_xxx.zip.

Successful establishment of communication in Poseidon network can be divided into several steps:
- definition of Poseidon communication in the project,
- definition (emulation) of peripherals that are to be communicated with,
- establishment of the link between the AMiT station and peripherals.

Creating a project is the same for the whole portfolio of the AMiT company that contains the Poseidon interface. Within the scope of this application note, the communication will be programmed for graphic control HMI **AMR-OP87/Pxx** with a communication module for the Poseidon 868 MHz system – **AW-P868A**.

4.1 Definition of Poseidon communication in the project

To define Poseidon communication, it is necessary to insert a **`Poseidon`** object into the project. It is the main communication object of the Poseidon network. It is possible to insert it from the list of communication objects window. The “Add object” window is accessible through the context menu of the “Communication” node.

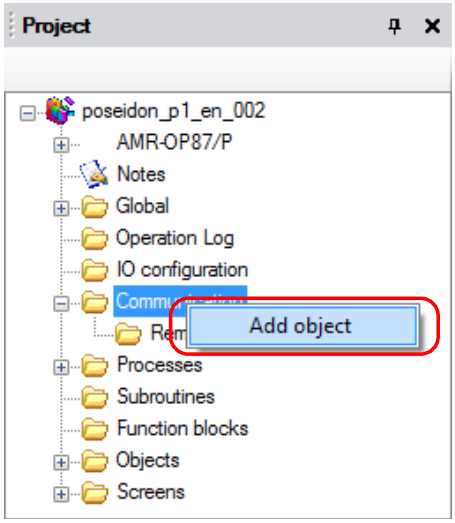

Fig. 1 – Adding objects into the project

After selecting “Add object”, the “Objects and blocks in libraries” list opens; select the **`Poseidon`** object.

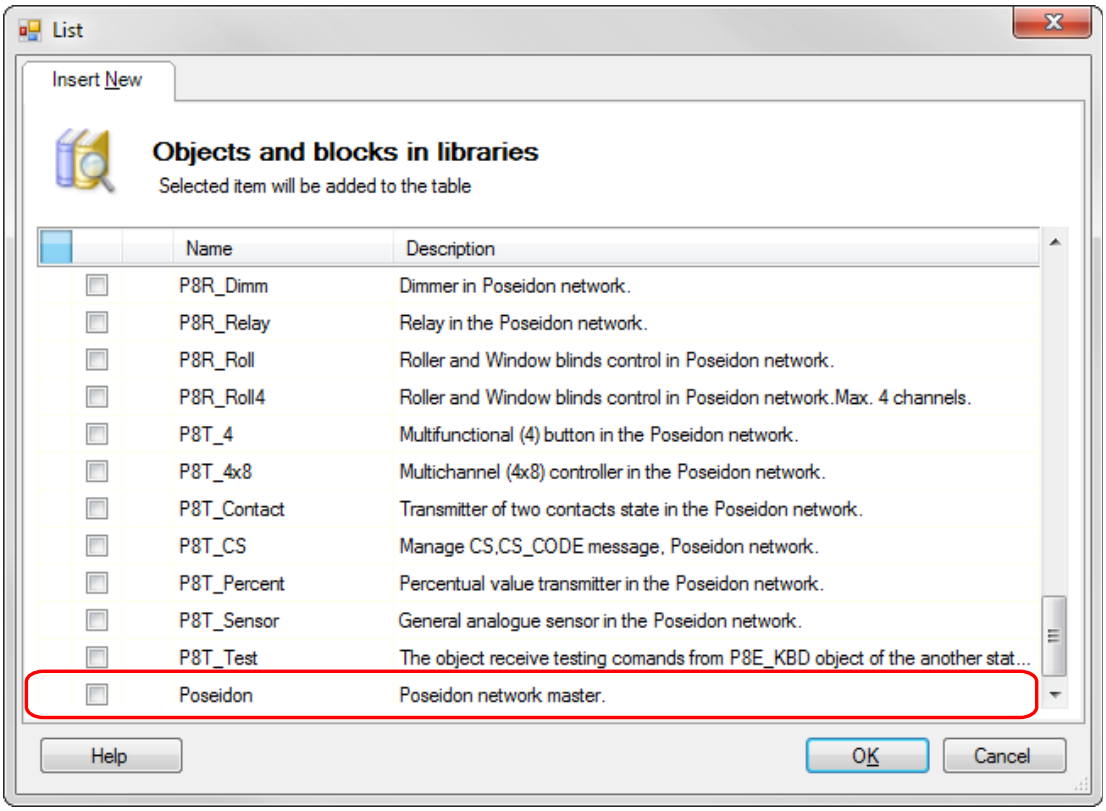

Fig. 2 – Selection of the **Poseidon** object

After confirming the selection (by clicking "***OK***" or by double-clicking the object), the object appears in the "Project" window under "Communication".

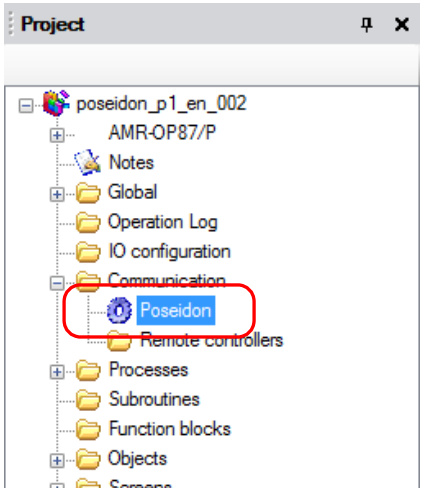

Fig. 3 – **Poseidon** object in the "Communication" section

Left-clicking the **Poseidon** object opens the "Properties" window listing the properties of the communication object. Settings can be left with the default values (the **AsistentEnable** parameter = True) so that it is possible to use the HMI with "Poseidon Asistent". This piece of SW was created by the ENIKA company and it is used for management of the Poseidon wireless network (see chapter 4.3.1 "Poseidon Asistent SW").

4.2 Definition of peripherals

The peripherals can be defined within the project in a similar way to defining the Poseidon communication as such – via the communication object list window. To communicate within the Poseidon network, take advantage of any of the P8x_xxx type objects from the “Poseidon Wireless System” section.

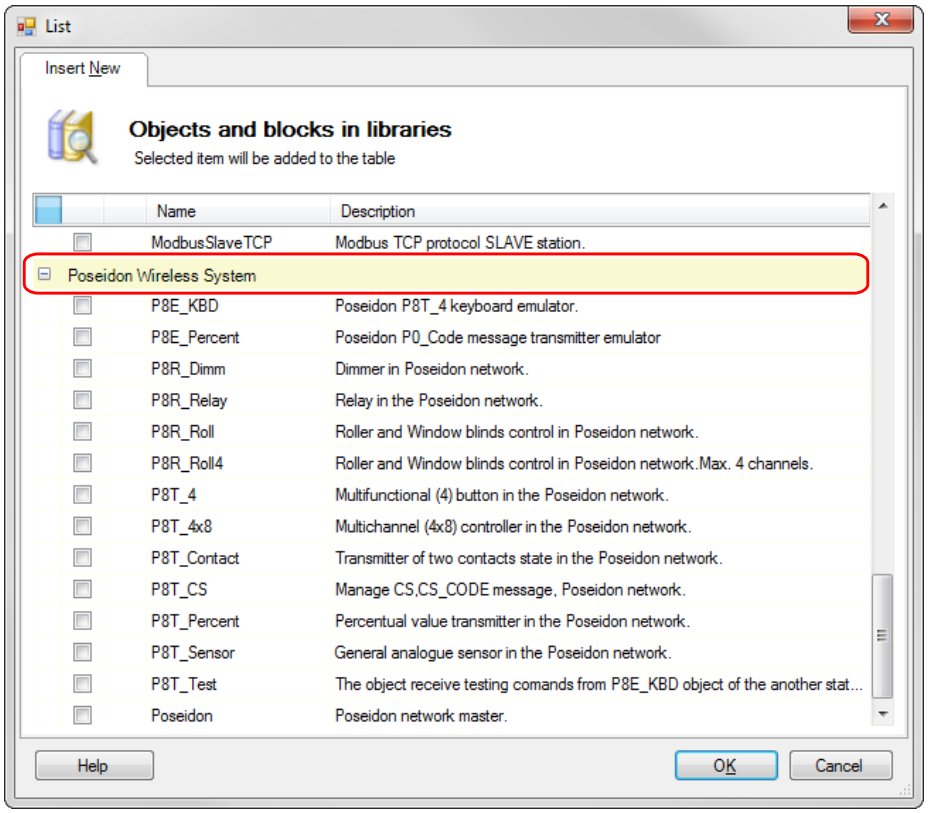

Fig. 4 – “Poseidon Wireless System” section

The list of all available objects – including the peripherals of the Poseidon wireless network for which the objects meant – can be found in the **DetStudio** Help for **EsiDet**.

In the sample application, communication will be occurring among these peripherals:

- P8 R 2 N/K – 2-channel receiver,
- P8 R R I – roller shades built-in receiver,
- P8 R DALI N – built-in receiver with a DALI output,
- P8 T Temp/RH – transmitter of temperature and humidity,
- P8 T 4 – 4-channel on-wall transmitter.

Based on information stated in the **DetStudio** Help for EsiDet, it is first necessary to include the following objects into the project:

- **`P8R_Relay`** for controlling the 2-channel receiver,
- **`P8R_Roll`** for controlling the roller shades receiver,
- **`P8R_Dimm`** for the built-in receiver with a DALI output,
- **`P8T_Senzor`** for the temperature and humidity transmitter (one object per value),
- **`P8T_4`** for the 4-channel on-wall transmitter.

The roller shades transmitter will be also controlled by buttons emulated by the HMI. To emulate the buttons, it is necessary to insert the **`P8E_KBD`** object.

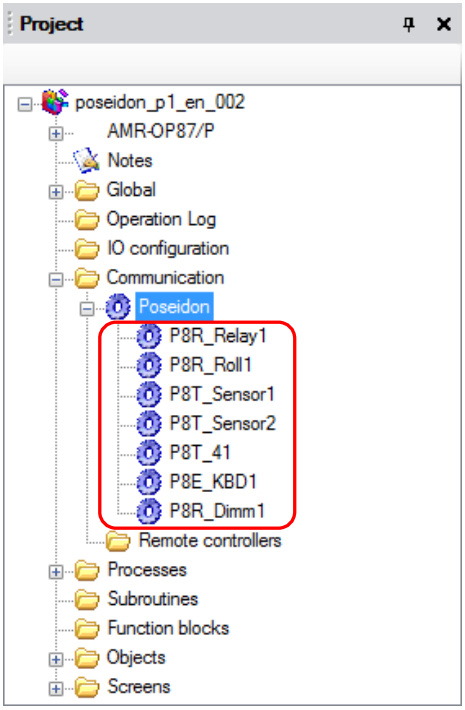

Fig. 5 – Definition of Poseidon network peripherals

Right-click the individual objects to open the “Properties” window to adjust communication settings.

It is necessary to set the channel count to objects **`P8R_Relay`** and **`P8R_Dimm`** (according to the pairing method – see chapter 4.3 “Establishing a link between the AMiT station and peripherals”) so that it corresponds to physical channels present on P8 R 2 N/K and P8 R DALI N. The count of channels is given by the **`ChannelCount`** parameter. If the parameter contains a different number of channels, the communication will not work.

Note
*It is possible to set the **`FailureThreshold`** parameter to **`P8R_xxx`** objects – it is the number of undelivered messages to the peripheral it takes for the communication failure state to occur.*
*Furthermore, for **`P8T_xxx`** objects (that actively send data without user input), it is possible to set the **`TimeOut`** parameter which determines the maximum waiting time for a message form the peripheral.*
Both parameters can be left in the default state.

After inserting objects, it is possible to work with their parameters both via the screens and the processes.

4.2.1 Properties of on-screen objects

When the panel is required to only display data – i.e. it is not supposed to process the data on the control level – received from the peripherals or to send commands (e.g. as a reaction to button press), it is not necessary to programme anything on the process level. It is sufficient to only use the programming and parametrisation of visualisation.

Displaying analogue values

To display analogue values received from a transmitter, use the **`NumericView`** control. Typically, this would be used to display temperature, humidity or the current position of roller shades or blinds.

After inserting a NumericView control onto the screen, assign the corresponding object parameter to the **`Variable`** parameter of the **`NumericView`** control (e.g. the **`Value`** parameter of the **`P8T_Sensor1`** object for temperature or the **`ActualPosition`** parameter of the **`P8R_Roll`** object for the current position of roller shades or blinds). Other parameters of the **`NumericView`** control can be customised to meet the project requirements.

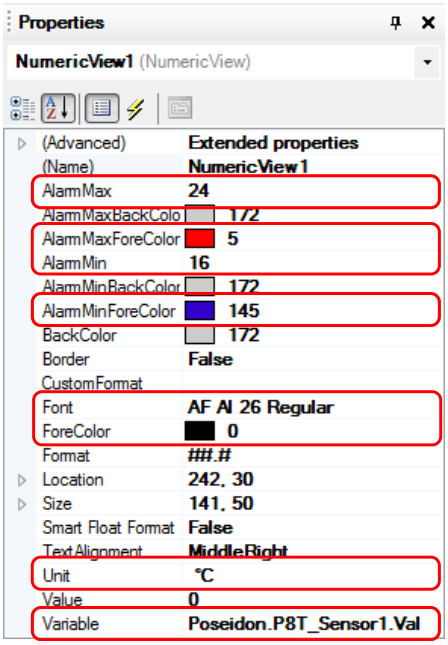

Fig. 6 – Setting **`NumericView`** control parameters

Settings analogue values

To set an analogue value, use the **`NumericEdit`** control. Typically, this would be used to set lighting intensity or the current position of roller shades or blinds.

After inserting a **`NumericEdit`** control onto the screen, assign the corresponding object parameter to the **`Variable`** parameter of the **`NumericEdit`** control (e.g. the **`Out0`** parameter of the **`P8R_Dimm1`** object for setting the lighting intensity via DALI). Other parameters of the **`NumericEdit`** control can be customised to meet the project requirements.

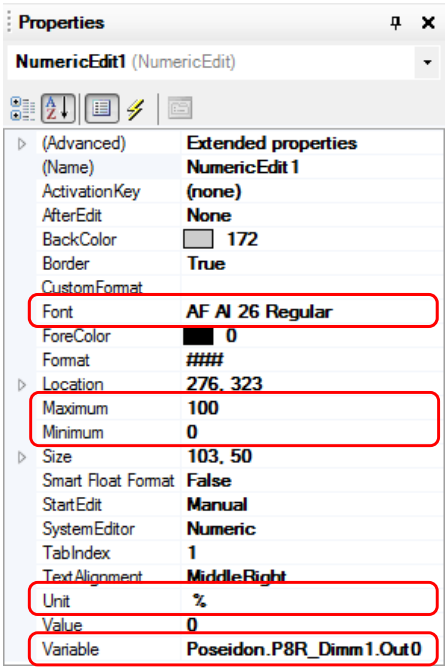

Fig. 7 – Setting **`NumericEdit`** control parameters

Displaying binary values

To display binary values received from a peripheral, use the **`BitSwitchView`** control. Typically, this can be used to view the state of relay or motion detection information from a PIR sensor.

After inserting a **`BitSwitchView`** control onto the screen, assign the corresponding object parameter to the **`Variable`** parameter of the **`BitSwitchView`** control (e.g. the **`ActualOut0`** parameter of the **`P8T_Relay1`** object to display the relay 1 state). Other parameters of the **`BitSwitchView`** control can be customised to meet the project requirements.

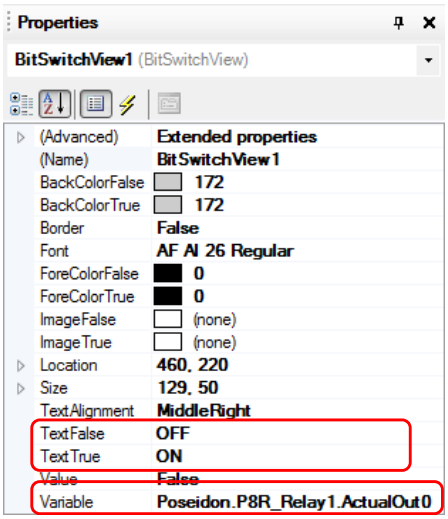

Fig. 8 – Setting **`BitSwitchView`** control parameters

Setting binary values

Setting of binary values can be divided to two groups:

- commanding,
- button pressing emulation.

Commanding

Commanding uses the **P8R_xxx** object that is directly meant – within the scope of **DetStudio** – for controlling the selected peripheral (e.g. **P8R_Relay** object is specifically meant to operate the P8 R 2 N/K peripheral).

To set a required state of the binary output via an on-screen control, use **BitSwitchButton**. After inserting a **BitSwitchButton** control onto the screen, assign the corresponding object parameter to the **Variable** parameter of the **BitSwitchButton** control (e.g. the **Out0** parameter of the **P8T_Relay1** object to set the desired relay 1 state on the P8 R 2 N/K peripheral). Other parameters of the **BitSwitchButton** control can be customised to meet the project requirements.

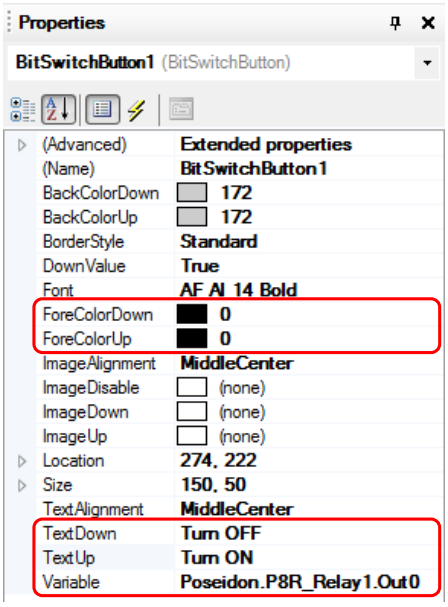

Fig. 9 – Setting **BitSwitchButton** control parameters

Button pressing emulation

Emulation of button presses uses the **P8E_KBD** object for communication. It simulates the press of a wireless button. The AMiT station therefore acts as a wireless button when the **P8E_KBD** object is used. The button press emulation function can be used for example for controlling roller shades or blinds via graphic controls placed on the screen of the AMiT station.

It is not convenient to use **BitSwitchButton** control to work with the **P8E_KBD** object. After setting the required behaviour of the wireless button, it is necessary to set a flag to the **P8E_KBD** that will send a corresponding message to the Poseidon network. The function can be implemented on screens within the **DetStudio** using only controls that support the so-called scripting (the **BitSwitchButton** control doesn't support scripts). It is, however, possible to use the **Button** or **ToggleButton** controls.

After inserting a **Button** control onto the screen, it is possible to use its "OnButtonDown" event. To switch to the list of events of the **Button** control, press the yellow lightning bolt icon in the "Parameters" window (when a **Button** control is selected).

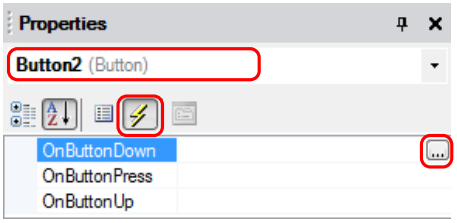

Fig. 10 – **Button** control parameters

Pressing the "**...**" button creates a corresponding event and switches the main window to the scripting section.

Insert the required code into the pre-defined events.

```
event Button1_OnButtonDown()
     Poseidon.P8E_KBD1.OFF1 = true; // Information about the bottom left button being
pressed
     Poseidon.P8E_KBD1.ON1 = false; // Cancels the information about the upper left
button being pressed
     Poseidon.P8E_KBD1.Transmit = true; // Issue to send the information about the
button press
end;

event Button2_OnButtonDown()
     Poseidon.P8E_KBD1.OFF1 = false; // Cancels the information about the bottom left
button being pressed
     Poseidon.P8E_KBD1.ON1 = true; // Information about the bottom left button being
pressed
     Poseidon.P8E_KBD1.Transmit = true; // Issue to send the information about the
button press
end;
```

The reaction to an emulated button press is given by the peripheral that was linked to the emulated buttons. In case of a link to a P8 R R I (blinds or shutter control), this may cause for example closing/opening the shutters or blinds. The function is given by the settings in the Poseidon Asistent SW (see chapter 4.3.1 "Poseidon Asistent SW").

4.2.2 Parameters of objects in processes

Using the parameters of the Poseidon communication objects in processes is similar to using other communication objects or variables.

Processing an analogue value from a transmitter and sending to a receiver

To process measured value or humidity, it is possible to only call the **Val** parameter of the **P8T_Sensor1** (measured value) or **P8T_Sensor2** (measured humidity) objects. The quantity that is transferred via individual **P8T_Sensorx** objects is given by the type of the wireless transceiver and by the number of quantities that it sends into the network (see section "Pairing transmitter peripherals" in chapter 4.3.2 "Pairing mode").

Should the station switch a wireless relay based on the measured temperature, it is possible to implement this function by programming the **Hyst** module as follows:

```
Hyst1(In = Poseidon.P8T_Sensor1.Val, Out => Poseidon.P8R_Relay1.Out1);
```

The above-mentioned code causes the P8 R 2 N/K peripheral relay 2 to close when the temperature measured by the wireless temperature transmitter rises above the limit (defined by the **Limit** parameter of the **Hyst1** block) increased by a half of the value set in the **Hysteresis** parameter of the **Hyst1** block.

To get the real state of the output, use the **ATSO** or **OutputsRead** parameter. **ATSO** is True when the peripheral sent the current states of its outputs. **OutputsRead** is True when all output states of the given peripheral are read using queries (see **DetStudio** Help for EsiDet).

Current states of outputs are saved to **ActualOutx** parameters.

Using the **OutputsRead** and **ActualOutx** parameters is convenient e.g. when a Poseidon network peripheral is controlled from multiple places at the same time (typically a wireless button and an AMiT station).

Processing a wireless button press

Upon a button press or release, the button transmitters send information about a button being pressed or released, respectively. When the user presses a button for a short time, it may happen that the AMiT station does not register the event. That is why it is recommended to place the button press processing into a process with a period of 100 ms or less. Furthermore, it is quite common to use one button of a peripheral as an ON switch and another button of the peripheral as an OFF switch. This needs to be treated by the code in the station.

An example of code addressing the issue of ON and OFF switch from a single peripheral – **P8T_4** in this case – is below.

```
RS1(
     Set = Poseidon.P8T_41.ON1,
     Reset = Poseidon.P8T_41.OFF1,
     Output => @Status
);
```

The **@Status** alias is True or False depending on whether the user pressed the upper left or bottom left button of the 4-button transmitter.

Controlling shutters

AMiT stations can control shutters by two methods:

- by emulating a button (see section “Button pressing emulation” in chapter 4.2.1 “Properties of on-screen objects”),
- or by commanding (sending the required value of opening/pitch of the shutters).

Sending the required value of opening/pitch of the shutters via the processes can be used e.g. when the shutters need to react to the current position of the sun. The position of the Sun can be gained by using the **Astro** object (see the **DetStudio** Help for EsiDet). The relationship between the position of the Sun and the shutters is completely determined by the user algorithm.

4.2.3 Using screen controls in combination with processes

In some cases, it is convenient to use screen controls in combination with other functions. Typically, it can be used to change the state of a control on the screen (e.g. a switch) based on the real state of the peripheral output (e.g. state of a relay).

State of a control based on the real state of a peripheral output

Switching based on the real state can be used e.g. when the peripheral is controlled from more than one place (wireless button, PIR sensor, AMiT station etc.). When implementing the control status change based on the current state of a peripheral output (**ActualOutx** parameter), it is not possible to use just the **BitSwitchButton** used in section “Setting binary values” of chapter 4.2.1 “Properties of on-screen objects”. It is necessary to use e.g. the **ToggleButton** or **SelectButton** control (or similar ones) that can be also controlled via the screen script. It is furthermore necessary to programme a function that will – after waiting for a pre-determined amount of time after a command – read the current state of the output. It will then optionally

change the state of the on-screen button. It is also possible to use one **TimerOff** block and two aliases for the same task.

Controlling a binary output with reading of real state

To control a binary output, use the **ToggleButton** control. After inserting a control onto the screen, it is possible to use its "OnToggleButtonDown" and "OnToggleButtonUp" events. To switch to the list of events of the **ToggleButton** control, press the yellow lightning bolt icon in the "Parameters" window (when a **ToggleButton** control is selected).

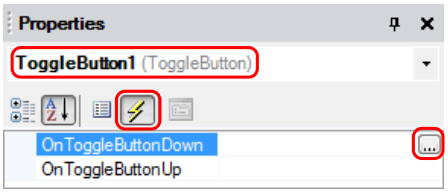

Fig. 11 – **ToggleButton** control events

Pressing the "**...**" button creates a corresponding event and switches the main window to the scripting section.

Insert the required code into the pre-defined events. In case of the sample application, a flag (needs to be defined as alias **@Change** in the process) is set about the request for a state change and the required state of the relay (flag about the change request will be used for launching the **TimerOff** block timer).

```
event ToggleButton1_OnToggleButtonDown()
     Process1.@Change = true;  // Setting the change flag
     Poseidon.P8R_Relay1.Out0 = true;  // Setting the required relay state
end;
event ToggleButton1_OnToggleButtonUp()
     Process1.@Change = true;  // Setting the change flag
     Poseidon.P8R_Relay1.Out0 = false;  // Setting the required relay state
end;
```

Furthermore, in the periodic process, it is necessary to define a **TimerOff** block that will delay the trailing edge of the **@Change** alias. The flag is therefore set with every state change caused by the HMI and it is set to False within the process immediately afterwards. During the **@Change** alias trailing edge delay (the length can be set via the **Delay** parameter), the **@P8R2NK_CH1** alias will be set according to the required relay state. When the timer is ended, the **@P8R2NK_CH1** alias state will be set according to the real state of the relay.

```
TimerOff1(Input = @Change);  // Delay of the changes flag
If TimerOff1.Out then
     @Change = false;  // Cancelling the change flag
     @P8R2NK_CH1 = Poseidon.P8R_Relay1.Out0;  // Alias according to the required relay
state
Else
     @P8R2NK_CH1 = Poseidon.P8R_Relay1.ActualOut0; // Alias according to the real
relay state
     If Poseidon.P8R_Relay1.ActualOut0 != Poseidon.P8R_Relay1.Out0 then
     // Synchronisation of the current state with the request from the panel
          Poseidon.P8R_Relay1.Out0 = Poseidon.P8R_Relay1.ActualOut0;
     EndIf;
EndIf;
```

The **@P8R2NK_CH1** alias will be used for displaying the state of the on-screen button. After changing the required relay state, the button displays the required relay state (for the duration of the **TimerOff Delay**). After the pre-set time elapses, the displayed button state will correspond to

the current relay state. This functionality is addressed by the following code. It is necessary to insert the code into the "OnRefresh" event of the screen where the button is inserted.

```
ToggleButton1.Checked = Process1.@P8R2NK_CH1;
```

The code above also deals with a situation when the relay state gets changed by e.g. a wireless button. The button on the panel automatically changes its status according to the relay state set by the wireless button.

Controlling an analogue output with reading of real state

To control the analogue output by buttons, use the **SelectButton** control.

After inserting the control on the screen, it is possible to define:
- number of buttons,
- the value that the button switching depends on,
- images that are to display when a button is pressed/released (not necessary).

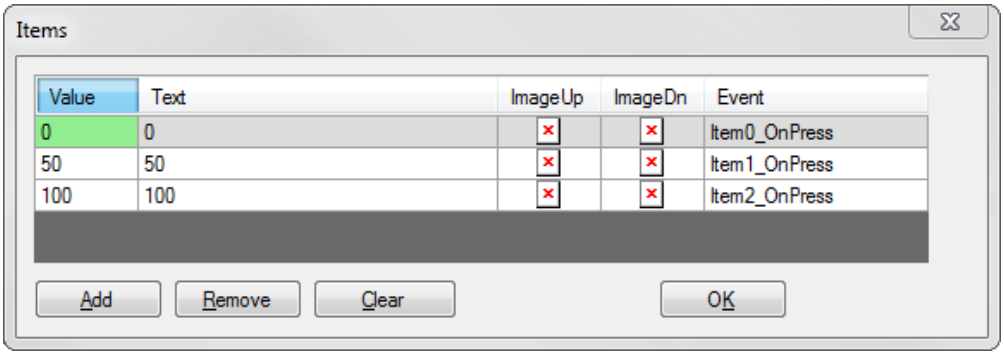

Fig. 12 – Setting buttons of the **SelectButton** control

After confirming the defined buttons by pressing "***OK***", an event will be created for each button. These events will look like this: "SelectButton1_ItemX_OnPress()". The following process is similar to controlling a relay via a **ToggleButton**.

Insert the required code into the pre-defined events. In case of the sample application, a flag (needs to be defined as alias **@ChangeDali** in the process) is set about the request for a state change and the required state of the relay (flag about the change request will be used for launching the **TimerOff** block timer).

```
event SelectButton1_Item0_OnPress()
     Process1.@ChangeDali = true;
     Poseidon.P8R_Dimm1.Out0 = 0.0;
end;

event SelectButton1_Item1_OnPress()
     Process1.@ChangeDali = true;
     Poseidon.P8R_Dimm1.Out0 = 50.0;
end;

event SelectButton1_Item2_OnPress()
     Process1.@ChangeDali = true;
     Poseidon.P8R_Dimm1.Out0 = 100.0;
end;
```

Furthermore, in the periodic process, it is necessary to define a **TimerOff** block that will delay the trailing edge of the **@ChangeDali** alias. The flag is therefore set with every state change caused by the HMI and it is set to False within the process immediately afterwards. During the **@ChangeDali**

alias trailing edge delay (the length can be set via the **Delay** parameter), the **ActualOut** variable value will be set according to the lighting intensity setpoint. When the timer ends, the **ActualOut** variable value will be set according to the real value of lighting intensity.

```
TimerOff2(Input = @ChangeDali); // Delay of the change flag
If TimerOff2.Out then
     @ChangeDali = false;  // Cancelling the change flag
     ActualOut = Real_To_Int(Poseidon.P8R_Dimm1.Out0);
Else
     ActualOut = Real_To_Int(Poseidon.P8R_Dimm1.ActualOut0);
EndIf;
```

The **ActualOut** value will be used for displaying the state of the on-screen buttons. To change the required value of lighting intensity, the status of buttons will be set according to the required lighting intensity (during the **TimerOff** block **Delay**). After the pre-set time elapses, the button state will correspond to the current lighting intensity. This functionality is addressed by the following code. It is necessary to insert the code into the **SelectButton** "OnRefresh" event on the screen where the button is inserted.

```
SelectButton1.Value = Process1.ActualOut;
```

The code above also deals with a situation when the required value state gets changed by another wireless peripheral. The states of the buttons on the HMI are automatically changed according to the lighting intensity set by another wireless peripheral (only when the set intensity equals to the value set in the **SelectButton** control).

4.3 Establishing a link between the AMiT station and peripherals

As stated above, each peripheral of the Poseidon system has its unique ID defined by the manufacturer. In order to operate the peripheral using the communication unit correctly, this ID must be filled in the appropriate objects and a link with the AMiT station must be entered into the receiver memory.

The link with the AMiT station can be entered into the peripheral in two ways:
- using the Poseidon Asistent SW from the ENIKA.CZ company,
- using the pairing mode.

4.3.1 Poseidon Asistent SW

The Poseidon Asistent SW can, among other things, read IDs of peripherals, create links to them and it enables advanced settings of them. When the links are established via Poseidon Asistent, it is necessary to set the read IDs into the corresponding parameters of **P8x_xxx** objects in the DetStudio project.

A PC with a Poseidon Asistent SW copy installed can connect to the wireless network via:
- a USB configuration transceiver – P8 TR USB – made by ENICA.CZ,
- an AMiT station with Ethernet and Poseidon interfaces.

That means that **any AMiT station with both an Ethernet interface and a Poseidon interface can be used** not only as a communication gateway for the Poseidon network of peripherals but also **as a configuration gateway for the Poseidon network**.

To enable communication of Poseidon Asistent through an AMiT station, the station needs to be running an application containing a **Poseidon** object with the **AsistentEnable** parameter set to True.

Caution
While a PC is connected to the AMiT station via Poseidon Asistent, communication with Poseidon peripherals in the network, as defined in the station programme, stops. It will resume only after the Poseidon Asistent SW stops communicating.

Stations without an Ethernet interface (e.g. **AMR-OP70RHP**) cannot function as a configuration gateway for Poseidon Asistent.

More information on the Poseidon Asistent SW is available at www.enika.cz.

If the Poseidon Asistent SW is not available, the link between the AMiT station and peripherals can be made through the process described below.

4.3.2 Pairing mode

Pairing receiver peripherals

Pairing can be illustrated on objects **`P8R_Relay`** and **`P8R_Roll`**. Before the actual pairing, an application generated with the given objects needs to be installed in the AMiT station. The receivers mustn't be linked to the AMiT station and they need to be visible in the Poseidon system at the moment of pairing (see the user's guides for the individual peripherals from the ENIKA.CZ company).

The pairing process is performed as follows.

- Connect both the AMiT station and the receiver to the power supply.
- In the DetStudio project, open the "Inspector 1" panel via the "Debugging / Inspector 1" menu. Insert the **`P8R_Relay1`** object into the open "Inspector 1" window (by pressing the "**+**" button).

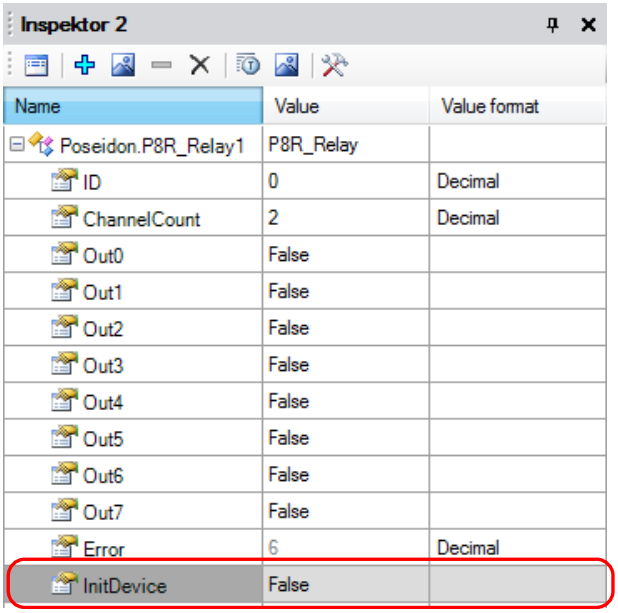

Fig. 13 – Object **`P8R_Relay1`** and some of its parameters in the "Inspector 1" window

- In the following step, press and release a programming button **1×** on the receiver (see the User's Guide for the individual peripherals by the ENIKA.CZ company) – this sets the receiver into programming mode.

- After releasing the programming button on the receiver, the **InitDevice** parameter of the **P8R_Relay1** object is set to “1” (True) in the “Inspector 1” window. That action launches the pairing sequence on the AMiT station side and in case the pairing is successful, the **ID** parameter shows the read value of the receiver ID. The value corresponding to the physical number of relay output is also automatically entered into the **ChannelCount** property, see the following figure.

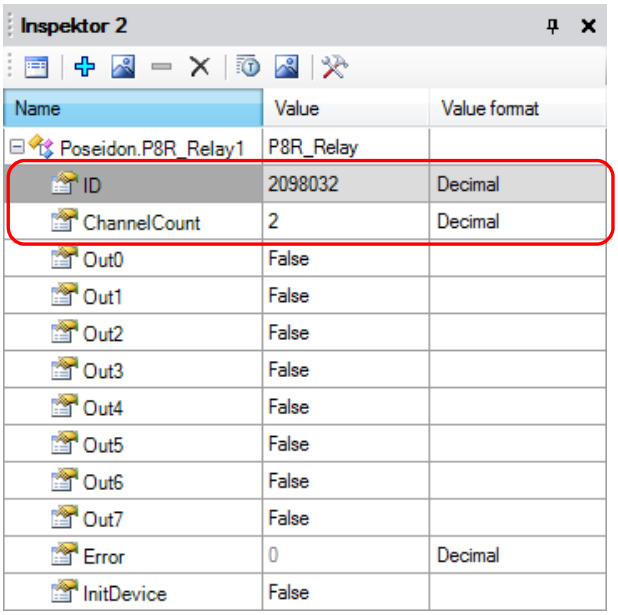

Fig. 14 – Read ID of the receiver after a successful pairing procedure

It is now possible to communicate with the receiver in the Poseidon network, i.e. to monitor / change relay output statuses, e.g. directly in the “Inspector 1” window, or the status can be changed using the screens or algorithms recorded in one of the periodic processes and monitor the changes with the Inspector.

Pairing transmitter peripherals

Pairing can be illustrated on objects **P8T_Sensor**. Before the actual pairing, an application generated with the given objects needs to be installed in the AMiT station.

The pairing process is performed as follows.

- Connect the AMiT station to the power supply. Transmitters are usually battery-powered; the battery has to be inserted in the transmitter and has to have a sufficient voltage level.
- In the DetStudio project, open the “Inspector 1” panel via the “Debugging / Inspector 1” menu. Insert the **P8T_Sensor1** object into the open “Inspector 1” window (by pressing the “**+**” button).

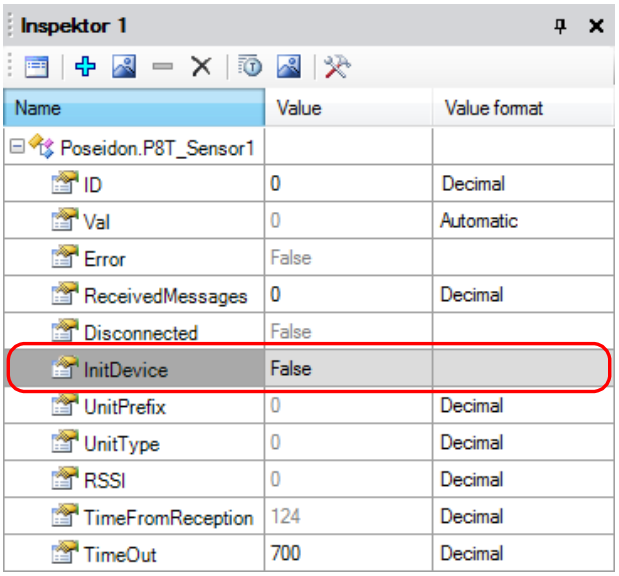

Fig. 15 – Object **`P8T_Sensor1`** and its parameters in the "Inspector 1" window

- The **`InitDevice`** variable of the **`P8T_41`** object is set to value "1" (True) in the "Inspector 1" window. Therefore, an incoming pairing sequence sent by the transmitter is expected.
- Activate the pairing sequence on the side of the transmitter (see the documentation for the given transmitter available at www.enika.cz). In case of the temperature and humidity transmitter, press the bottom button on the transmitter (two LEDs light up on the transmitter) and then press the top button (the pairing sequence is sent and LEDs flash a couple of times).
- If the pairing is successful, the **`ID`** parameter will again be filled by the receiver ID, see the following figure.

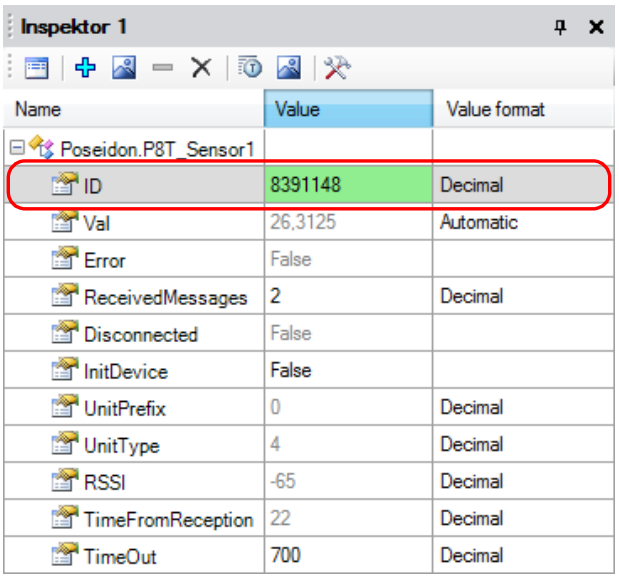

Fig. 16 – Filled transmitter ID after a successful pairing procedure

Now it is possible to communicate with the given transmitter in the Poseidon system, e.g. monitor status of individual buttons directly in the "Inspector 1" window.

When it is necessary to pair a peripheral that provides the Poseidon network with more than one type of information, it is necessary to insert that many communication objects into the project as many types of information the peripheral provides the network with. Each other object that

represents further quantities of the same peripheral needs their address to be set to a value 1 higher than the previous one. As an example, when there is a P8 T TempRh peripheral that provides information about temperature and humidity, there needs to be an extra **`P8T_Sensor`** object for humidity and its address need to be 1 higher (in this case 8191149) than the one for temperature.

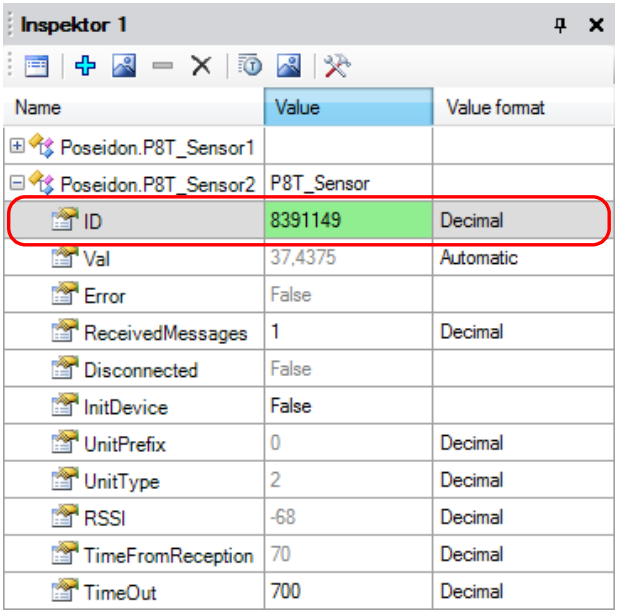

Fig. 17 – Filled transmitter ID after for further quantity

Note
*When transmitter IDs are known, it is possible to enter them in to the **`ID`** parameter in the "Properties" panel already when creating the project. It is then unnecessary to perform the pairing sequence manually. The AMiT station will receive queries from peripherals with corresponding IDs automatically.*

The pairing mode can be implemented using the aforementioned parameters directly on the HMI screens. In that case, there won't be no need to use DetStudio for correct pairing of AMiT stations with Poseidon peripherals – the pairing sequence will be performed using only the HMI screens. See a sample screen – "Screen2" – in example poseidon_p1_en_02.dsox included in the application note package.

4.4 Overwriting the application

If changes in the user application are necessary and saving the existing object parameter settings is required (such as peripheral ID, number of peripheral channels, etc.) for further use after the application is overwritten, use one of the following procedures.

- Enter ID (number of channels, etc.) into the appropriate parameter box in the "Properties" window for individual objects. See the following fig. displaying the "Properties" window for the **`P8R_Relay1`** object.

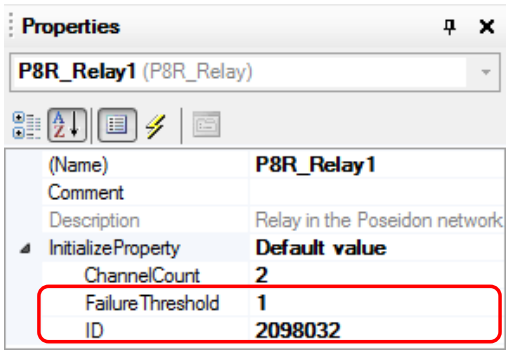

Fig. 18 The properties windows of the **`P8R_Relay1`** object

- Backup the data of the existing application from the AMiT station via the data manager and then renew the data after overwriting the application. Further information is available in the Help for **EsiDet** in chapter “Data manager”
- Select the “Perform backup and variables recovery in station” option. More information on this option is available in the Help for **EsiDet** in section “Download application” of chapter “/Contents/Loading the application/”.

5 Communication diagnostics

The AMiT station uses internal functions to automatically monitor the status of the physical communication module of the 868 MHz wireless system. If the system gets overloaded or enters another unusual status, the communication module resets automatically.

5.1 Signal strength

Objects that operate both transmitters and receivers within the project have the **RSSI** parameter. Value of this parameter states the strength of the RF signal of a message received from the given transmitter or queried receiver.

The signal values can be classified as follows:

- very good > -50 dBm,
- satisfactory -80 dBm to -50 dBm,
- bad < -80 dBm.

5.2 Status of communication with peripherals

Status of communication with an operated peripheral of the Poseidon network can be assessed by means of parameters **Error** and **Disconnected**.

Error messages of receivers

Value	Significance
0	No error.
1	Unexpected type of command in response.
2	Upon initialisation – receiver type does not match the object type.
3	The output status does not match the requested status.
4	No response.
5	Unexpected channel number in response.
6	Change of the ID assigned from the application detected.

Error messages of transmitters

Value	Significance
0	Battery OK.
1	Low battery.

Disconnected – error messages of transmitters

Value	Significance
0	No error.
1	The transmitter has not transmitted any data for more than 10 min.

The **Disconnected** parameter is available only in objects operating transmitter peripherals, except for objects **P8T_4** and **P8T_4x8**.

6 List of supported peripherals

The chart below states types of devices from ENIKA.CZ that can be operated by the AMiT station in the Poseidon network using the aforementioned objects. The list has been updated on the date of this application note.

Supported peripherals

Peripheral	Object	Number of objects for the peripheral operation	Note
Switching receivers	P8R_Relay	1	Max. 8 channels.
P8 R 1 xxx *)			
P8 R 2 xxx			
P8 R 4 xxx			
P8 R 8 xxx			
Dimmer receivers	P8R_Dimm	1	Max. 4 channels.
P8 R D I xxx *)			
P8 R 01-10 N			
P8 R DALI xxx *)			
Roller shades receivers			
P8 R R xxx *)	P8R_Roll	1	–
P8 R 4R S	P8R_Roll4	1	–
Receivers for control of TRVs	P8R_Relay		
P8 R Valve N 24V		1	–
Lighting controllers **)	P8T_Contact	1	Presence detection.
	P8T_Percent	1	Lighting controller output.
	P8T_Sensor	1	Ambient lighting intensity.
P8 TR PSMR16 (HR)			
P8 TR PS HB			
P8 TR PS LR xxx *)			
Button transmitters *)	P8T_4	1	Max. 4 buttons.
P8 T 2 xxx			
P8 T 3 xxx			
P8 T 4 xxx			
P8 T 4a xxx			
Multiple-channel button transmitters	P8T_4x8	4	One object for the group A to D.
P8 T 4x8a xxx *)			
Contact status transmitters	P8T_Contact	2	One object per contact.
P8 T(R) 2C I			
Transmitters of state of contacts and demand-side management	P8T_Contact	2	One object per contact. Can only be used in the “Individual transmitter” mode, see the user’s guide for the peripheral.
P8 TR 2C DIN			
P8 TR 2U DIN			
Motion sensor switches	P8T_Contact	1	–
P8 T PS W			
P8 T PSMR16/A xxx *)			
Temperature transmitters	P8T_Sensor	1	–
P8 T Temp xxx			
Temperature and humidity transmitters	P8T_Sensor	2	One object per physical quantity.
P8 T Temp/RH xxx *)			
Transmitters of CO_2, temperature and humidity	P8TContact	1	Control of a built-in relay.
	P8T_Sensor	3	One object per physical quantity.
P8 T CO2 xxx *)			

Peripheral	Object	Number of objects for the peripheral operation	Note
Flood sensors	P8T_Contact	1	Detection of water leaks.
	P8T_Sensor	1	External temperature measurement
P8 T AQ			

*) *Characters xxx represent further specifications of the peripheral, e.g. design (built-in, into dropped ceiling, …) or frame design, etc.*

**) *In combined peripherals such as the lighting controller, using all listed objects (P8T_Contact, P8T_Percent, P8T_Sensor) for message processing is not necessary. If it is desirable to process e.g. only the information about lighting intensity, only the object P8T_Sensor is used.*

7 Universal applications

A communication unit application was created in cooperation with the ENIKA.CZ company – it provides processing of a pre-defined number of peripheral (transmitters or receivers). The application is available at www.amitomation.com in section "**Solutions\Application sample projects**". It is sample project called "AMR-CP24 – Poseidon wireless network".

The number of pre-defined peripherals is as follows:

- 40× relay receiver with up to 8 channels,
- 40× roller-blinds receiver,
- 40× dimmer receiver (also applicable: built-in dimmer, 1 V to 10 V or DALI),
- 25× transmitter with up to 4 buttons,
- 25× contact state transmitter,
- 40× temperature or humidity transmitter,
- 20× button press emulation of a single transmitter with 4 buttons.

The configuration of the actual number of devices in the given technology is performed using the Poseidon Asistent SW. The configurations for individual devices are saved into the communication unit using the MODBUS TCP/IP protocol. This protocol also allows reading statuses and commanding configured peripherals. The list or registers of the MODBUS TCP/IP protocol is included in the Poseidon Asistent SW. More information is available at www.enika.cz.

In order to read statuses and command device outputs using the AMiT superior system, communication via **DB-Net/IP** is prepared in the application. The list of WIDs for operation of individual peripherals is included in the documents accompanying the afore-mentioned application sample project "AMR-CP24 – Poseidon wireless network".

8 Technical support

All information on integrating AMiT stations into Poseidon networks will be provided by the technical support department of the AMiT company. The Technical support is best contacted via e-mail at **support@amit.cz**.

9 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.

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.