

# smartDEN IP-PLC

## Web enabled PLC Controller

*User Manual*  
*Date: 19 Dec 2022*

*For firmware version: v1.27 / Mar 2022*



### Custom Program Editor

#### Program Code

```
} else if (VAR_I1 == 1) {  
  // State S1  
  REL1 = 1; // Barrier is opened  
  if (DIN1 == 0 && DIN2 == 0) { // Check sensors A and B  
    TIMER = 100; // Start timer (10 seconds period)  
    VAR_I1 = 2; // Transit to S2  
  }  
} else if (VAR_I1 == 2) {  
  // State S2  
  REL1 = 1; // Barrier is opened  
  if (TIMER == 0) { // Is the timer expired ?  
    if (DIN1 == 1 || DIN2 == 1) { // Another car ?  
      VAR_I1 = 1; // Return to S1  
    } else {  
      VAR_I1 = 0; // Transit to S0  
    }  
  }  
}
```

Characters left: 8552

Program state: Stopped

Browse...

Load

Upload

Run

☐ Don't reset custom variables

## Trademark Notices

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Apple, Mac, Mac OS, Mac OS logo are either trademarks or registered trademarks of Apple Computer Inc. in the United States and/or other countries.

smartDEN is registered trademark of Denkovi Assembly Electronics LTD in European Union.

Other product names and company names described in this document are trademarks or registered trademarks.

## Content

1.	Features .....	4
2.	Application examples .....	6
3.	Technical parameters.....	14
4.	Connectors, ports and led indicators.....	17
5.	Installation .....	18
6.	Default settings .....	30
7.	DAEL elements .....	33
8.	Working with I/O resources in DAEL .....	38
9.	Web access.....	40
10.	HTTP XML/JSON I/O operation .....	59
11.	HTTP XML/JSON program operation .....	65
12.	Security considerations.....	69
13.	Appendix 1. Mechanical drawing .....	70
14.	Appendix 2. DAEL language grammar.....	71
15.	Appendix 3. DAEL operators precedence.....	72
16.	Appendix 4. DAEL source code for application examples.....	73

## 1. Features

**smartDEN IP-PLC** is an Ethernet module (IP PLC controller) that can be used in a wide range of automation, remote monitoring, control, measurement and data-acquisition applications. Thank to its multi-channel design (8 digital inputs, 4 analog inputs, 4 temperature inputs, 8 SPDT relays and 2 analog outputs) this device can perform simultaneously a variety of measurement, control and automation functions. The built-in web interface allows users to configure the **smartDEN IP-PLC**, as well as to monitor/control input/output channels manually. **smartDEN IP-PLC** provides XML/JSON interface for integration with third-party developed applications.

In addition, **smartDEN IP-PLC** module can execute a custom program, written in a simple high-level language called **DAEL**. The program is composed directly from web browser and then checked and uploaded to the module. When started, the custom program is executed in parallel with the other functions and does not restrict any of them. Thus, the **smartDEN IP-PLC** module can work in standalone mode without need to be controlled by the network.

A list of **smartDEN IP-PLC** features includes:

### Communication:

- Fully compatible with 10/100/1000 Base-T networks, Auto-MDIX;
- Protocols: TCP/IP, HTTP, DHCP, DNS, SNMP, ICMP (ping);

### Inputs:

- 8 digital inputs with On/Off LED (input voltage range: 0-12V DC / 0-24V DC);
- 8 counters (32 bit) attached to digital inputs;
- 4 analog inputs (input voltage range: 0-10V DC);
- 4 temperature inputs for sensors NTC thermistors type B57500M;
- Linearization (scaling) for the analog inputs;
- Programmable temperature units: Celsius/Fahrenheit;
- Configurable filters for the digital/analog inputs;

### Outputs:

- 8 SPDT relays (with NO and NC contacts);
- 2 analog outputs: 0-10V DC (10 bit resolution);
- Single pulse feature for the relay outputs;

### Web Interface:

- Configuration of system parameters;
- Outputs control, access current measurements;
- Secure login authorization;
- Access protection (by IP and MAC address);

### TCP/IP Services:

- HTTP server:
  - Read (GET) current input/output values in XML/JSON format;
  - Set outputs along with GET request parameters;
- Encrypted login can be used to access XML/JSON values;

### Standalone Mode:

- PLC - user can program the I/O of the module to work in custom way using **DAEL** language;

- Single digital input can be configured to control several relays;
- Single or differential analog input can be configured to control several relays;
- Week schedule table for start/stop the **DAEL** program at specific time without computer (up to 30 events);
- Real time clock (can sync up with network time servers);

**Power Supply:**

- Supply voltage: 12V DC or 24V DC (selectable during purchase);
- Additional source voltage 12V DC or 24V DC provided to power input sensors;
- Power supply protection against reverse polarity;

**Physical and Environment:**

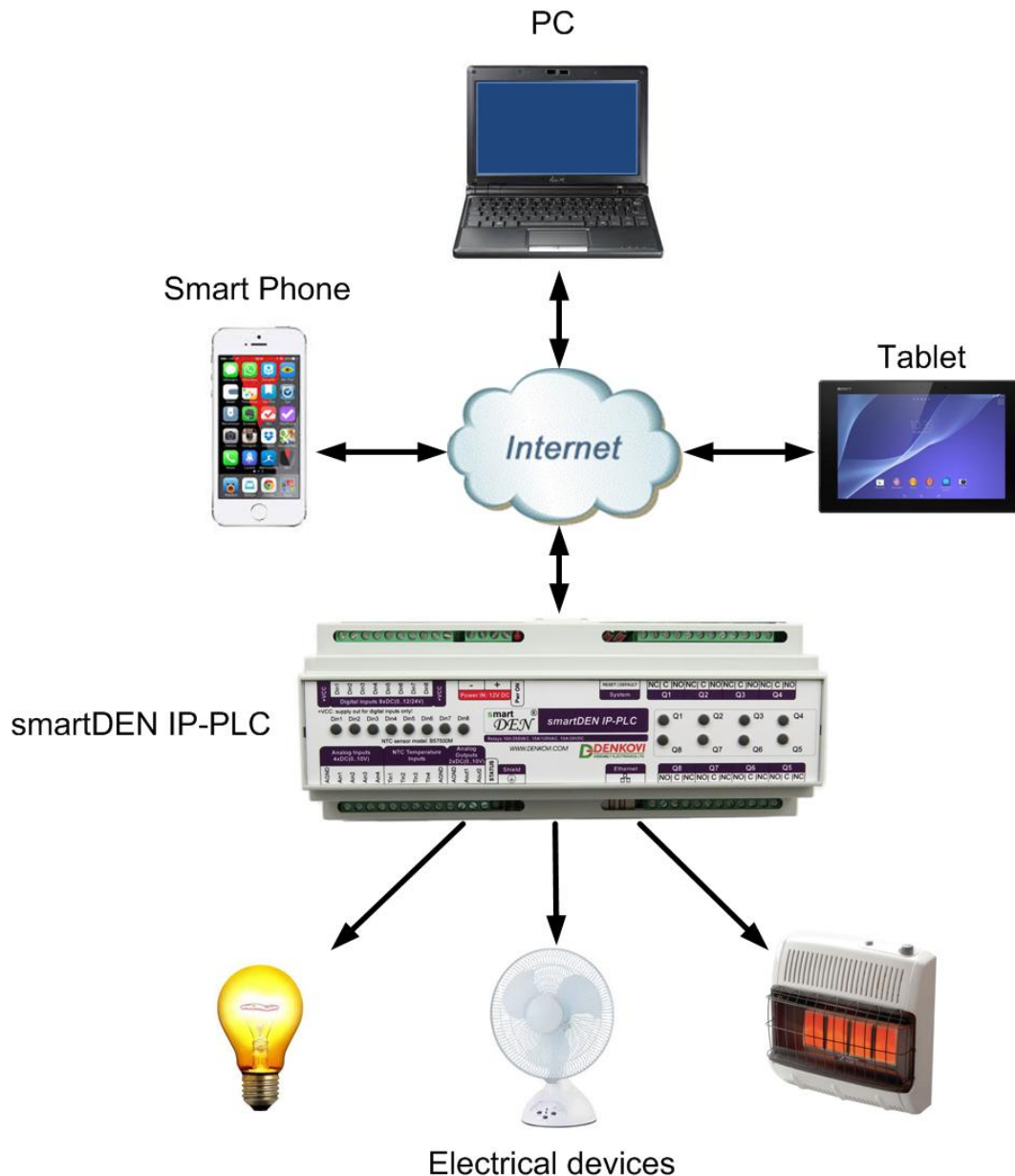
- Working temperature range: 0 to 70°C;
- DIN rail standard housing.

## 2. Application examples

The following examples show some basic applications of **smartDEN IP-PLC**. The examples are only conceptual and additional equipment/connections can be required in actual implementations.

### 2.1. Electrical appliances remote control applications

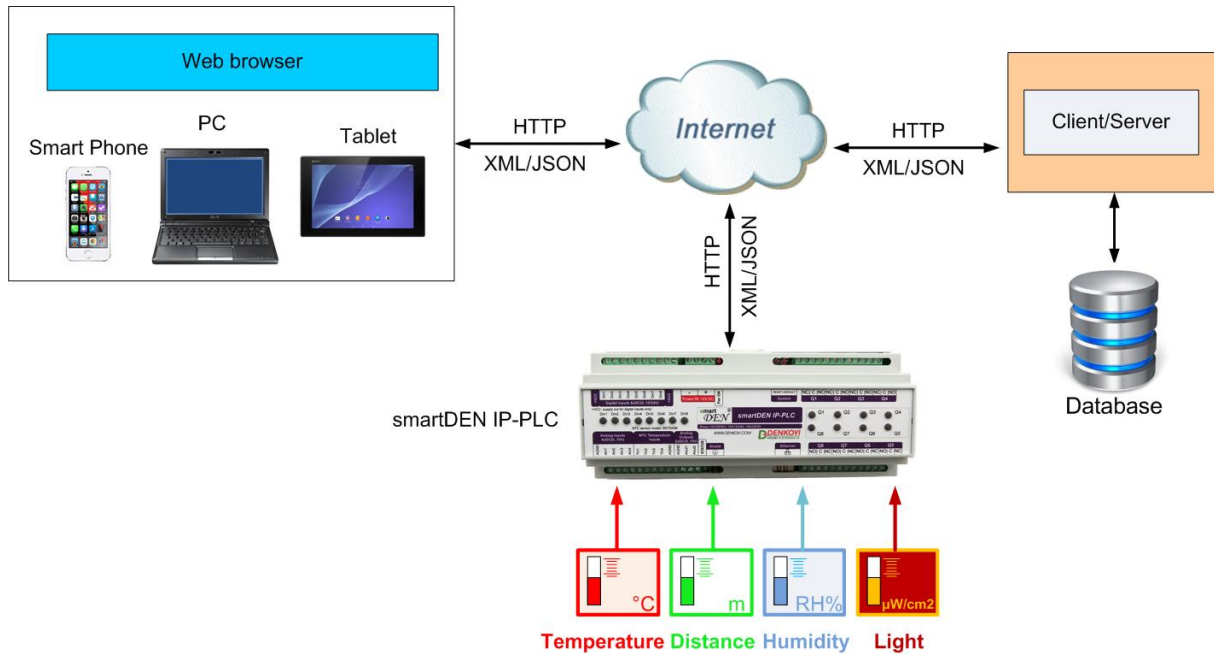
**smartDEN IP-PLC** can be used to control remotely electrical appliances (Figure 2.1). Up to 8 electrical devices connected to Relays can be controlled independently. Various integration protocols (HTTP/XML/JSON, Web-browser access) can be used by any modern device to control the appliances from all over the world.



**Figure 2.1.** Controlling electrical appliances remotely

## 2.2. Monitoring and logging applications

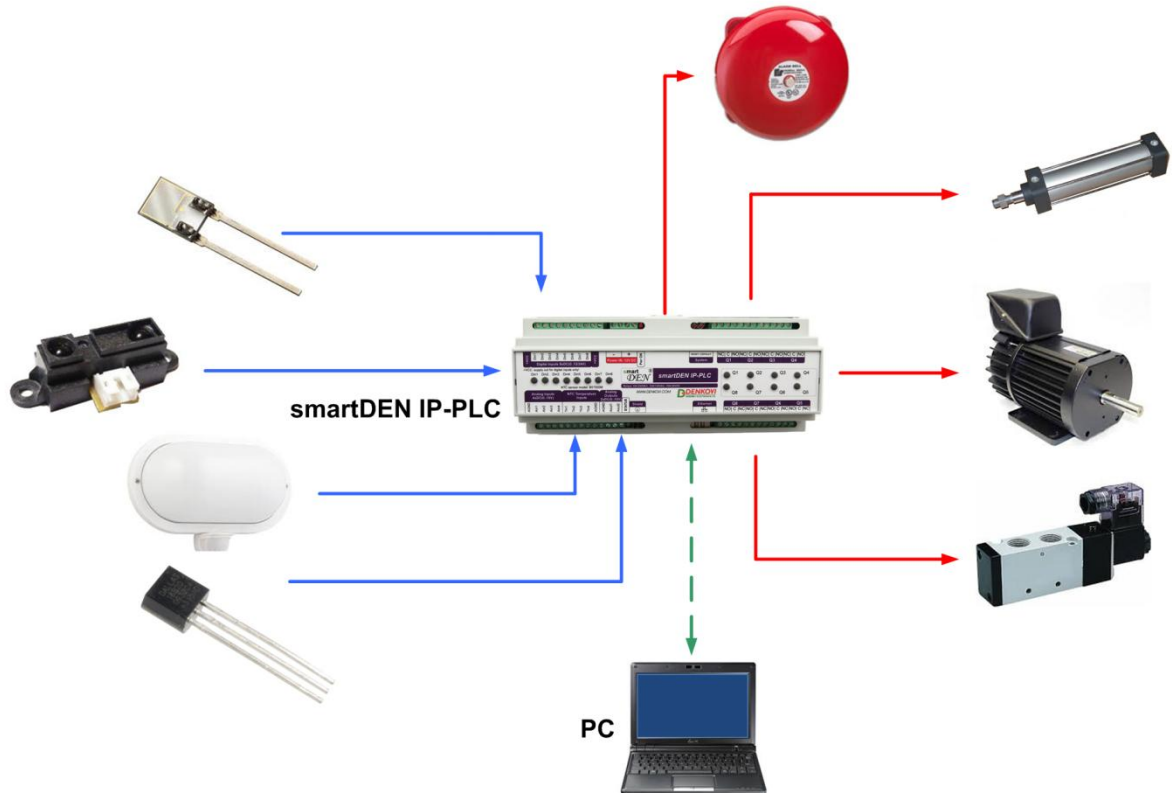
smartDEN IP-PLC can be used to build monitoring and logging systems. Up to 8 various kinds of sensors for temperature, humidity, distance, light etc. can be connected to the Analog Inputs of the device (Figure 2.2). Provided HTTP/XML/JSON APIs allow for easy integration with third-party applications.



**Figure 2.2.** Remote sensors monitoring

### 2.3. Standalone applications with sensors and electrical devices

Once configured, **smartDEN IP-PLC** can be used in standalone control applications. For example, the events from Digital Inputs or threshold conditions of single/differential Analog Inputs can be configured to control various devices connected to Relay outputs (Figure 2.3).

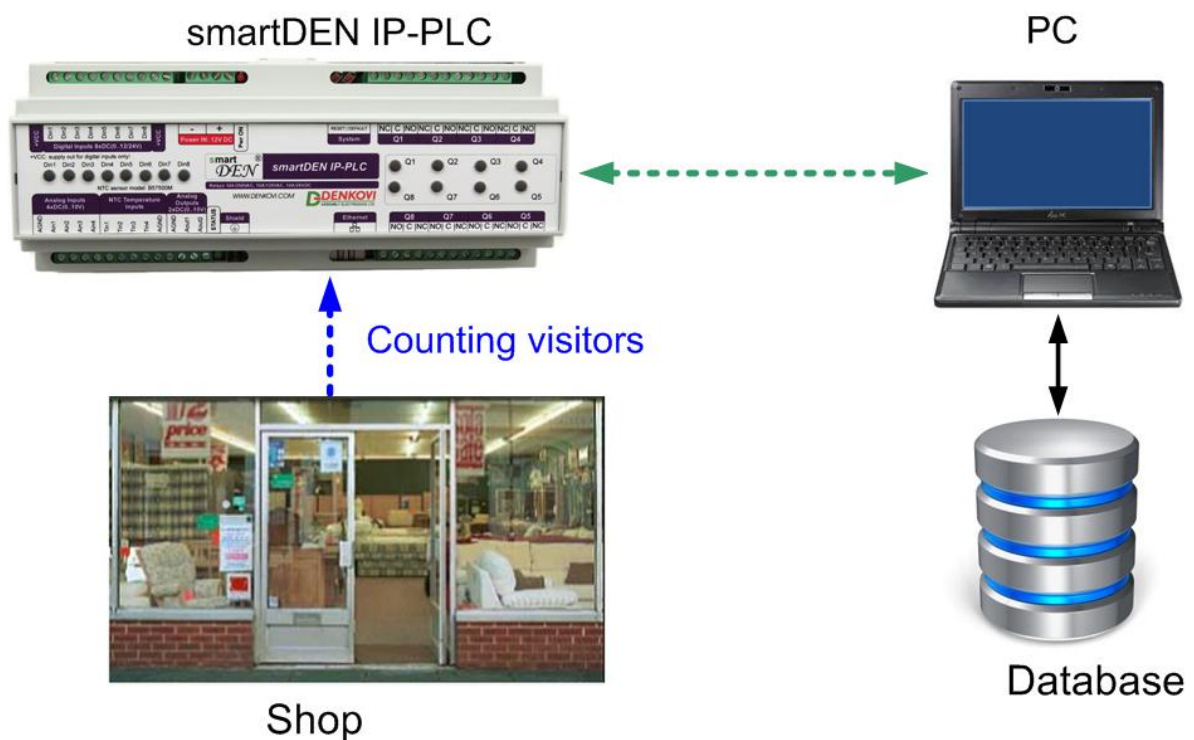


**Figure 2.3.** Standalone control of electrical devices



## 2.4. Events counter

**smartDEN IP-PLC** provides 8 x 32-bit Counters (from 0 up to  $2^{32}-1$ ) attached to the Digital Inputs that can be used to count various events - for example detect when a person enters in a shop through the door (Figure 2.4). Upon detecting the specified edge/level condition **smartDEN IP-PLC** increments the corresponding Counter. With suitable software and database one could easily organize a simple monitoring and statistic system.



**Figure 2.4.** Counting visitors in a shop

## 2.5. Web based thermo-regulator

Each Relay can be set to work in Regulator mode where it can be controlled only from an Analog Input. The controller can be configured to switch different Relays upon the value of single/differential Analog Input (one input can control many Relays). For example, one Relay can be set for heating, another for cooling and etc. (Figure 2.5). All the process may be monitored online and this makes **smartDEN IP-PLC** suitable for building standalone Web enabled thermo-regulators.

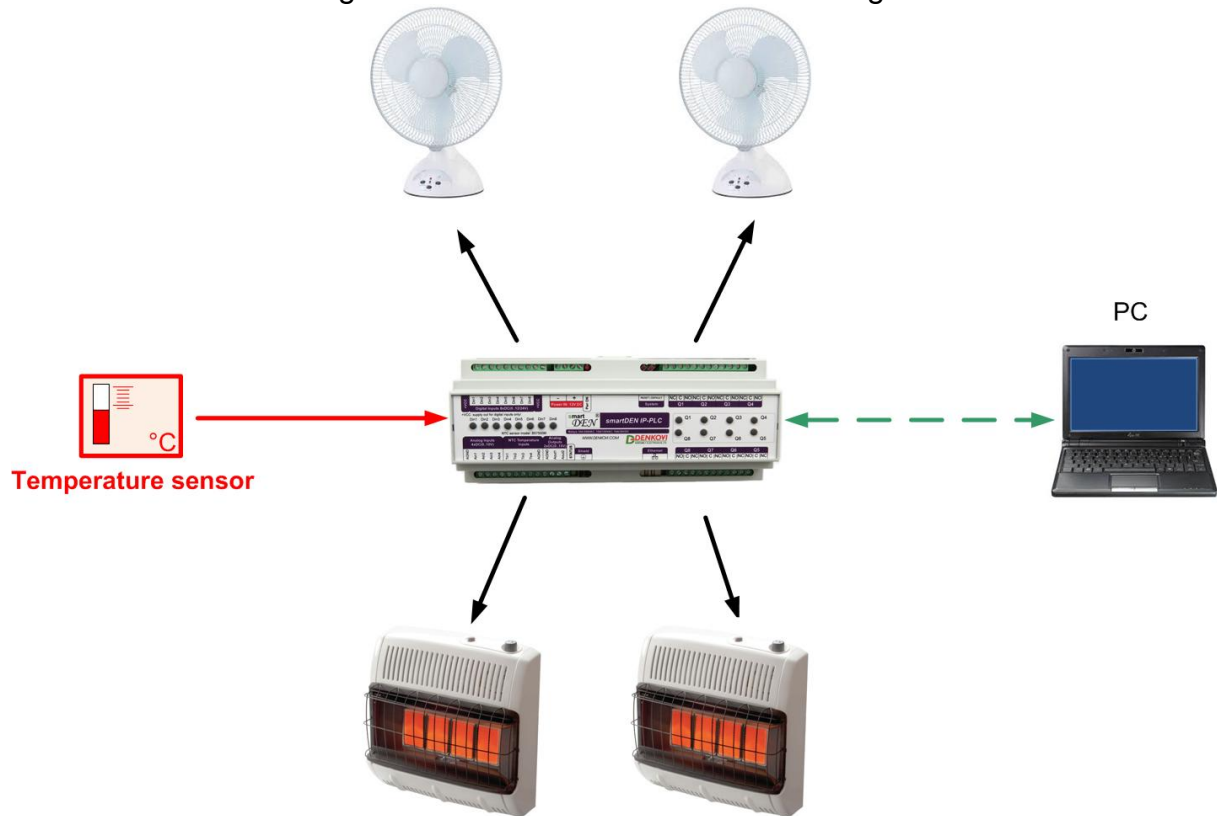
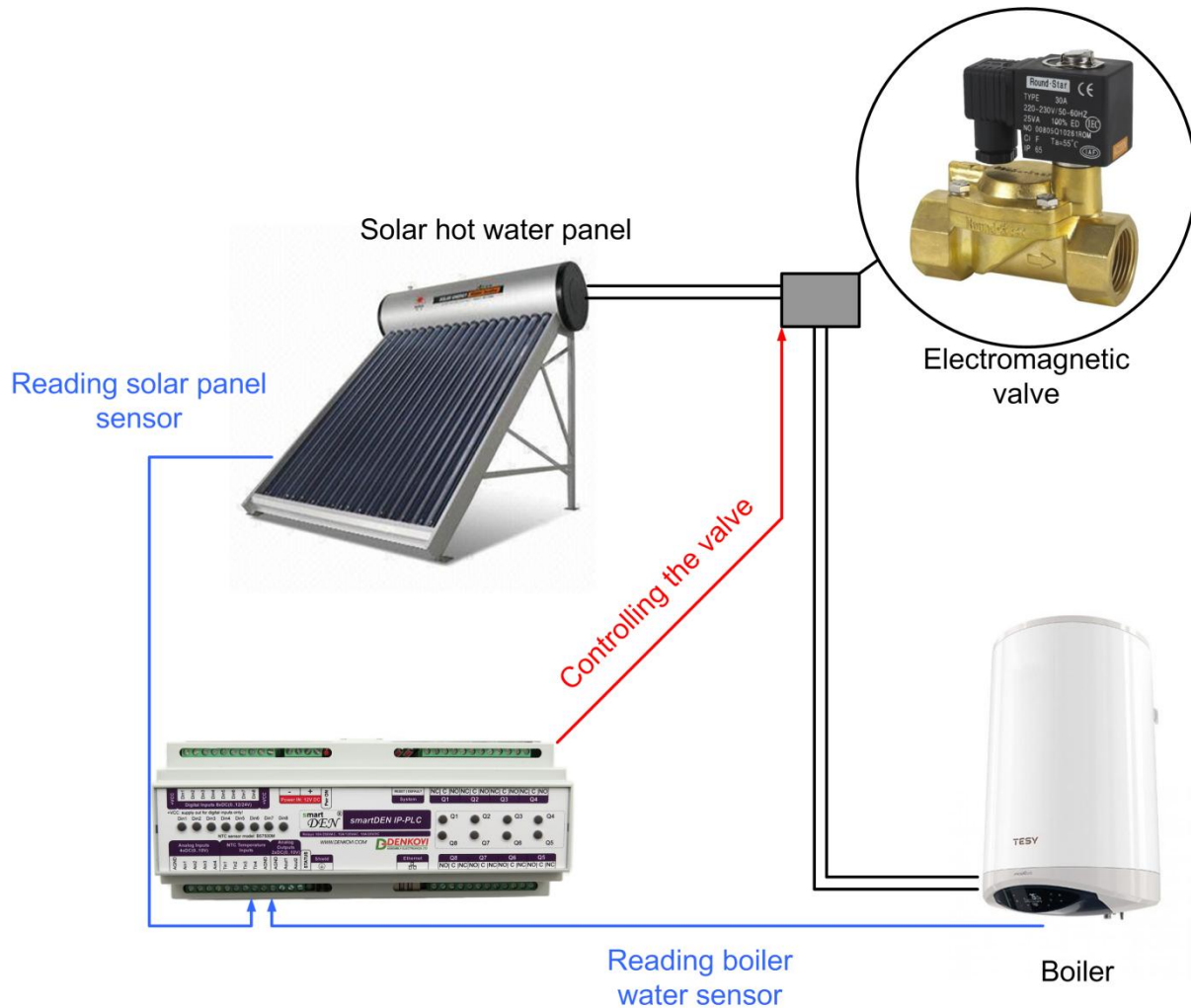


Figure 2.5. Web based thermo-regulator

Another possible application for **smartDEN IP-PLC** is differential thermo-regulator very useful in cases when solar hot water panel must be monitored and controlled based on two temperature sensors values. The first sensor monitors the temperature of the panel water and the second monitors the temperature from the boiler water. If the boiler water is cooler than the panel water, the valve is opened and the hot water from the panel comes in to the boiler.



**Figure 2.6.** Differential thermo-regulator

## 2.6. Home automation

smartDEN IP-PLC can be used in a variety of home automation tasks:

- Climate: heating, ventilation and air conditioning;
- Lighting: switching On/Off the electric lights in the house;
- Shading: opening and closing the blinds and curtains;
- Security: monitoring and control of central locking, doors and windows, etc.

A number of flexible control schemes can be set up, for example:

- Use of Regulator mode of the Relays for opening and closing the blinds and curtains depending on the level of the outside sunlight;
- Use of Pulse mode of the Relays for controlling garage doors, windows, roofs and for any other application where a certain period of time should be maintained, etc.;
- Use of “many inputs to one relay” controlling mode – each Relay can be paired simultaneously with a Digital Input (button, switch...), Analog Input (sensor for temperature, humidity, distance, light...), as well can be controlled remotely (browser, HTTP/XML/JSON);



**Figure 2.7.** smartDEN IP-PLC can be used in home automation systems

## 2.7. Irrigation systems

Other applications of **smartDEN IP-PLC** include irrigation systems. Two different modes can be combined:

- Start and stop the irrigation at specific times using the Week Schedule features;
- Start and stop the irrigation depending on humidity level by applying a control loop including a humidity or rain sensor and solenoid valve controlled by a Relay.



**Figure 2.8.** smartDEN IP-PLC can be used in home irrigation systems

## 2.8. Custom programming (PLC)

All the above applications were based on the built-in ready to be used configurable features of the **smartDEN IP-PLC**.

The other powerful feature of the module is the custom PLC like programming and in this way it is possible to be solved more complex tasks. For example:

- Custom pulse generators
- Triangular wave generators
- Advanced week/month scheduler
- Advanced thermostats
- Car parking systems
- Aquarium control systems
- Barrier control
- Garage door control
- Automation of production lines
- Advanced lighting dimming scenes
- Other custom applications....

A number of application examples along with the corresponding DAEL source code are provided in Appendix 4.

### 3. Technical parameters

**Table 3.1. Physical parameters**

Parameter	Value
Size (L / W / H), mm	210 x 85 x 58
Weight, g	420
Operating temperature, °C	0 to 70

**Table 3.2. System parameters**

Parameter	Value
Power supply voltage, V DC	12 or 24 (depends on the model) ±2
Maximum current consumption, mA	450 at 12V, 300 at 24V
Protection against reverse polarity	Yes
Hardware Real Time Clock (RTC)	Yes
Default settings restore button	Yes
Reset button	Yes

**Table 3.3. Digital inputs**

Parameter	Value
Digital inputs number	8
Digital inputs voltage range, V DC	0 up to 30
Input switching threshold from 0 to 1, V DC	> 7.6
Input switching threshold from 0 to 1, mA	> 3.2
Input switching threshold from 1 to 0, V DC	< 4.5
Input switching threshold from 1 to 0, mA	< 1.8
Supported sensor output type	PNP
Input type	Resistive with Schmitt trigger
Protection against reverse polarity	Yes

**Table 3.4. Counters**

Parameter	Value
Counters number	8
Max. pulses frequency	Digital Inputs 1, 2, 3, 8 1 ..10 Hz (See point <a href="#">7.9</a> )
	Digital Inputs 4, 5, 6, 7 9 kHz



**Table 3.5. Analog inputs**

Parameter	Value
Analog inputs number	4
Analog inputs full scale voltage range, V DC	0 up to 10
Analog inputs absolute maximum non-destructive voltage, V DC	24
Analog inputs resolution, bits	10
Value of LSB, mV	~10
Input impedance, KΩ	1330
Sample period, ms	For DINs 1,2,3 and 8: Min: 25*
	For DINs 1,2,3 and 8: Max: 300**
	For DINs 4,5,6 and 7: 0.111
Protection against reverse polarity	Yes

\*No communication (HTTP) is taking place with the module

\*\*Intensive communication (HTTP) with the module

**Table 3.6. Temperature inputs**

Parameter	Value
NTC inputs number	4
Sensor type	B57500M
Units	Celsius/Fahrenheit
Sensor working temperature range	-55°C/-67°F to +155°C/311°F
Accuracy	±0.5°C / 0.9°F
Sample period, ms	Min: 25*
	Max: 300**

\*No communication (HTTP) is taking place with the module

\*\*Intensive communication (HTTP) with the module

**Table 3.7. Relays**

Parameter		Value
Relays number		8
Max. Voltage		250V AC
Max. switchable current	Voltage=250V AC	10A
	Voltage=125V AC	15A
	Voltage=28V DC	10A

**Table 3.8. Analog outputs**

Parameter	Value
Analog outputs number	2
Output voltage	0 to 10V DC

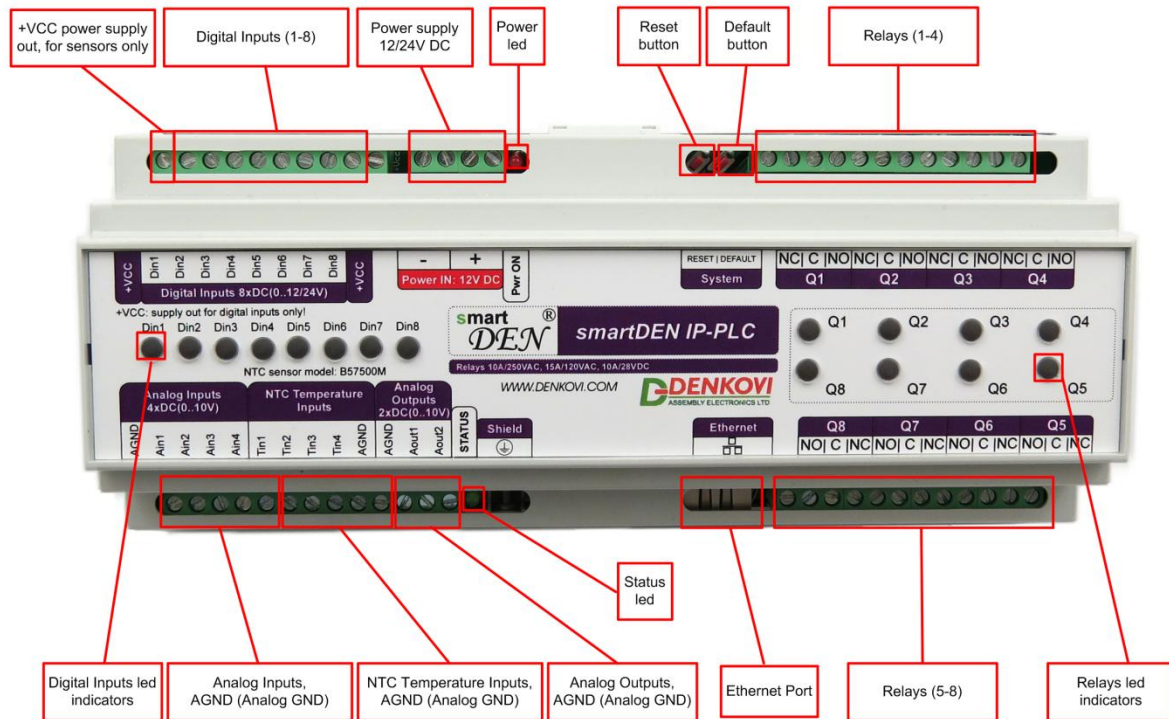
**Table 3.9.** Network/protocols

Parameter	Value
Network parameters	IP/Mask/Default gateway
MAC lock (protection)	Yes
DHCP	Yes
DNS	Yes
ICMP	Yes
Web server for configuration/access	Yes
Secure HTTP/XML/JSON access	Yes



#### 4. Connectors, ports and led indicators

A picture with the **smartDEN IP-PLC** connectors, ports and led indicators is shown in Figure 4.1.



### Figure 4.1. Device overview

## 5. Installation

- This device must be installed by qualified personnel;
- This device must not be installed directly outdoors;
- Installation consists of mounting the device, connecting to an IP network, connecting sensors, providing power and configuring via a web browser.

### 5.1. Box mounting

smartDEN IP-PLC can be mounted to a standard (35 mm by 7.55 mm) DIN rail (Figure 5.1). Attach the module to the DIN rail by hooking the hook on the back of the enclosure to the DIN rail and then snap the bottom hook into place.



**Figure 5.1.** Mounting the device to a DIN rail

## 5.2. Power supply

**smartDEN IP-PLC** must be powered with 12V DC (24V DC) stabilized and filtered voltage. After power on, the power led must be on and STATUS indicator must start blinking in 5 seconds which means the module is running normally (Figure 5.2).

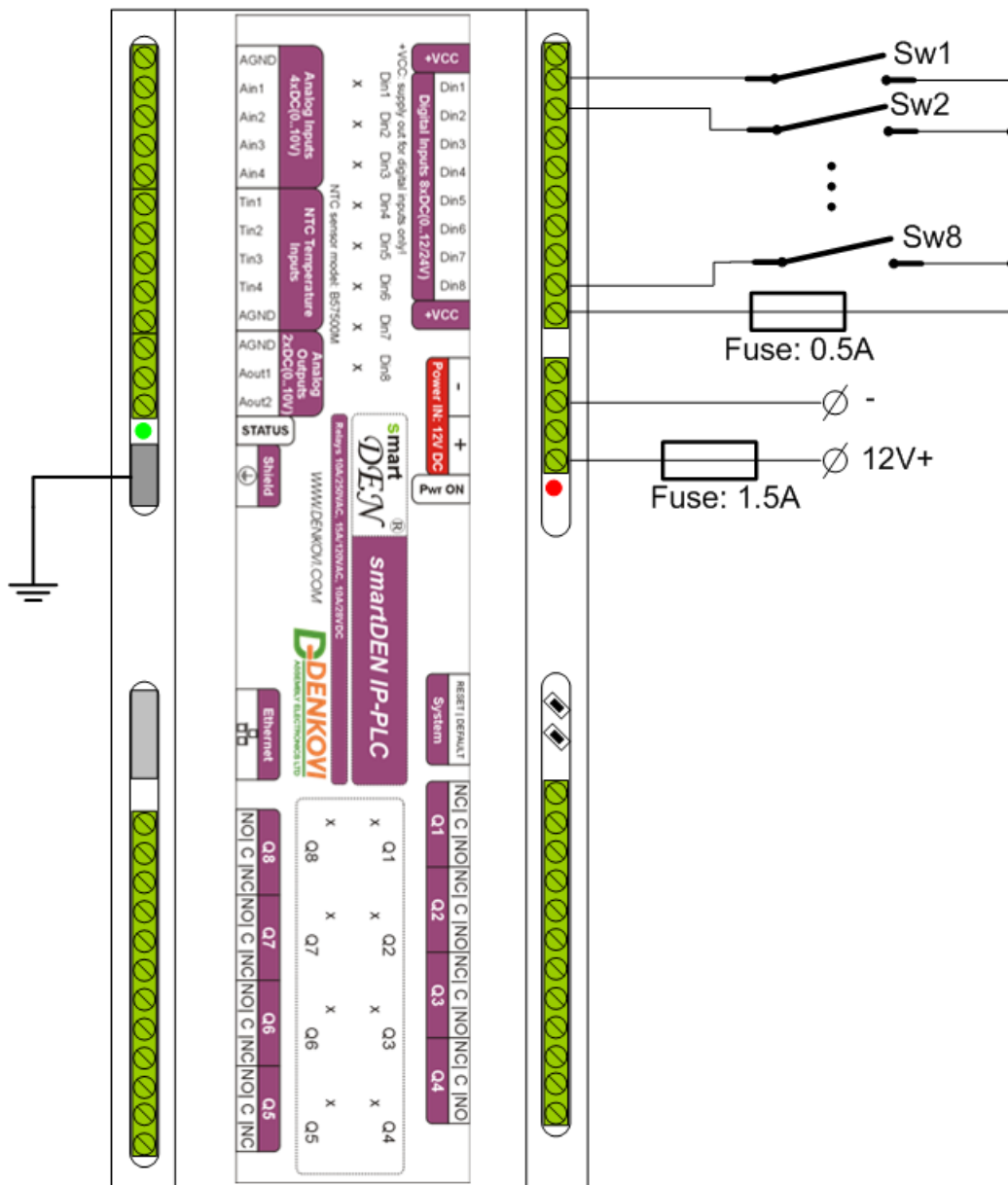


**Figure 5.2. smartDEN IP-PLC power supply**

- ❗ Please keep the polarity and the power supply voltage range!
- ❗ **smartDEN IP-PLC** does not accept AC power supply voltage. It is highly recommended to check the power supply source parameters before the module is powered on.
- ❗ The power supply equipment shall be resistant to short circuit and overload in secondary circuit.
- ❗ When in use, do not place the equipment so that it is difficult to disconnect the device from the power supply.

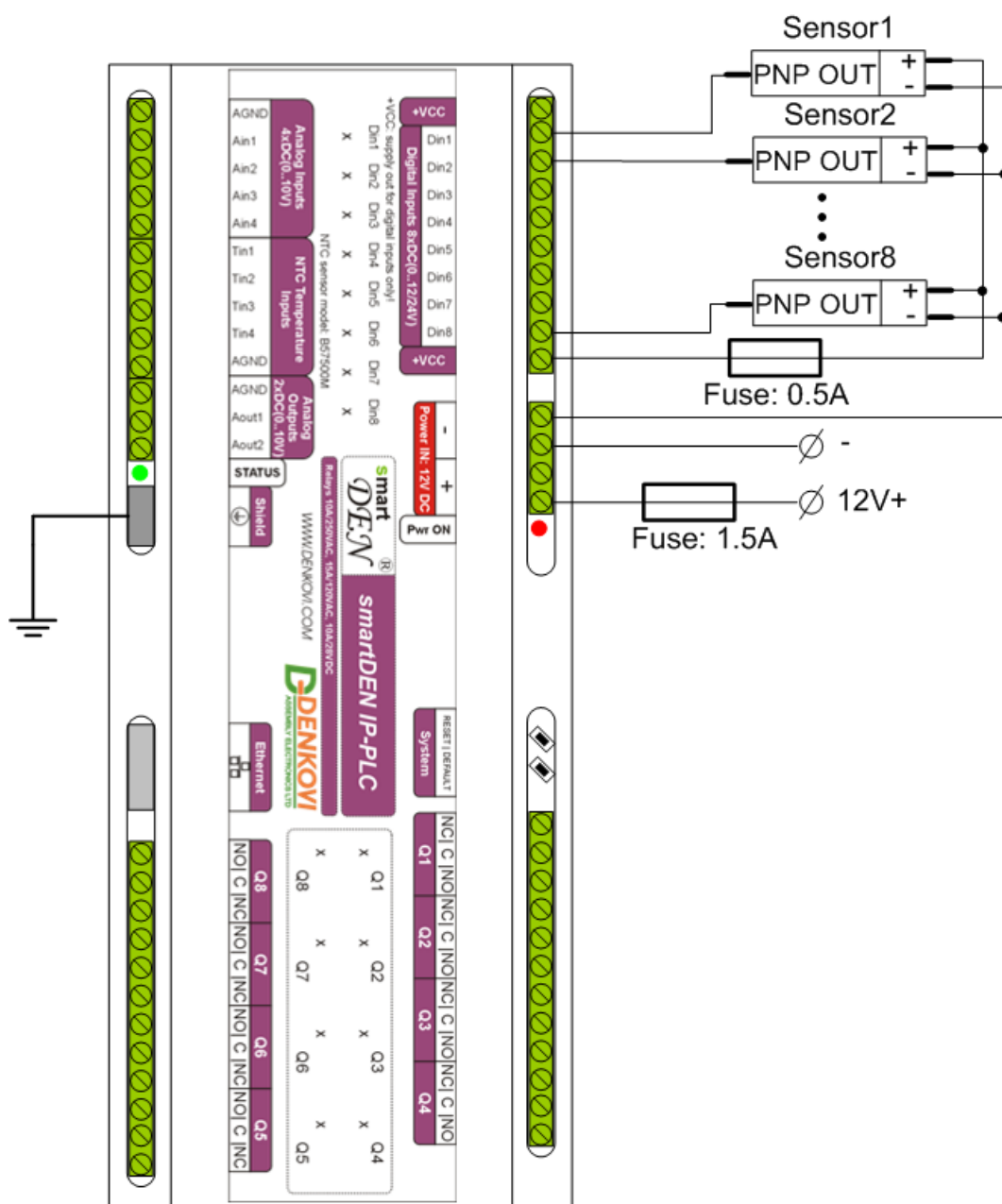
### 5.3. Connecting inputs

Figure 5.3 shows the typical connection of dry contact sensors, switches, buttons, door sensors etc. to the Digital Inputs.



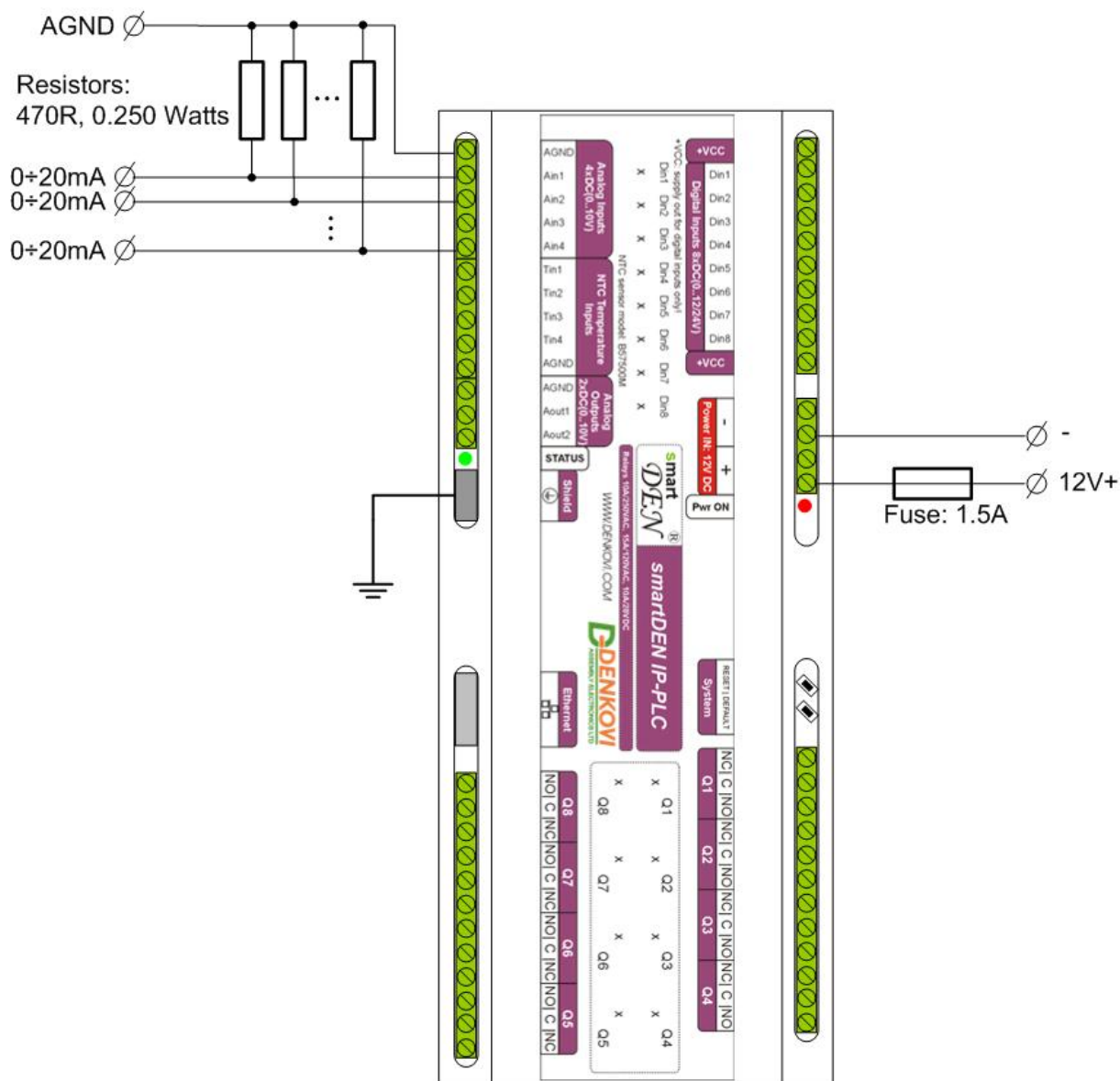
**Figure 5.3.** Connecting SPST NO output (dry contact) sensors, switches, buttons, door sensors etc. to the Digital Inputs

When connecting sensors with PNP output they can be powered from the additional internal 12V DC power source (Figure 5.4).



**Figure 5.4.** Connecting PNP output digital sensors requiring power supply voltage 12V DC to the Digital Inputs

Sensors with 0-20 mA output can be connected to the Analog Inputs as shown in Figure 5.5.



**Figure 5.5.** Connecting 0-20 mA output sensors to the Analog Inputs



The diagram illustrates the connection of two sensors to a SmartDEN IP-PLC. The PLC is a SmartDEN IP-PLC with various modules including Analog Inputs, NTC Temperature Inputs, Analog Outputs, and Digital Inputs. The PLC is connected to a 12V+ power source through a 1.5A fuse. The PLC also has a status LED and a shield connection.

**Sensor Connections:**

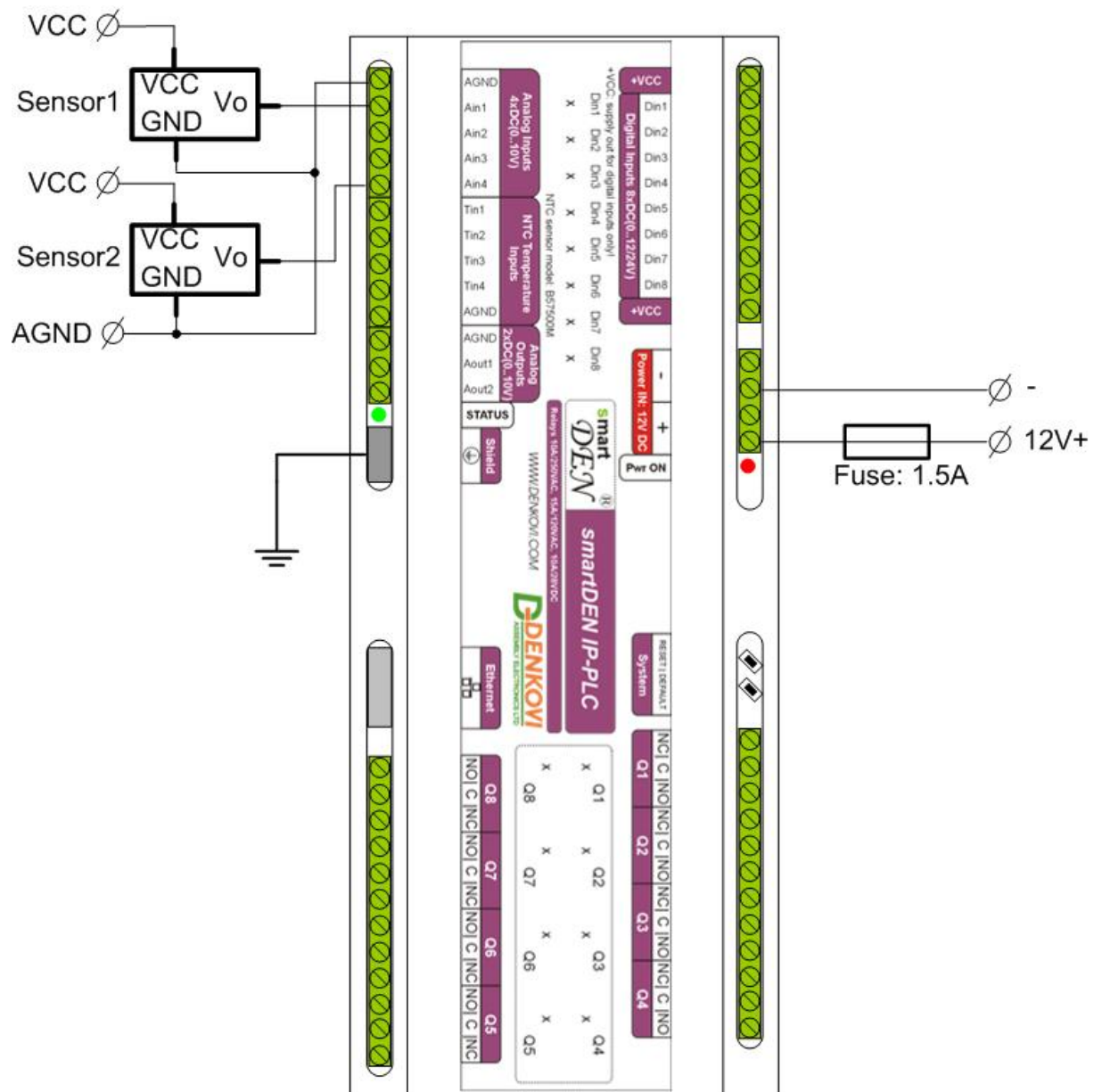
- Sensor1:** VCC, GND, Vo
- Sensor2:** VCC, GND, Vo

**PLC Modules and Connections:**

- Analog Inputs:** Ain1, Ain2, Ain3, Ain4, AGND
- NTC Temperature Inputs:** Tin1, Tin2, Tin3, Tin4, AGND
- Analog Outputs:** Aout1, Aout2, AGND
- Digital Inputs:** Din1, Din2, Din3, Din4, Din5, Din6, Din7, Din8, +VCC
- Power:** 12V DC, Pwr ON
- Status:** STATUS, Shield
- Reset:** RESET / MANUAL
- System:** System
- Outputs:** Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8
- Inputs:** I1, I2, I3, I4, I5, I6, I7, I8

**Figure 5.6.** Connecting analog sensors with output from 0 up to 10V DC to the Analog Inputs

NTC thermistors type B57500M can be connected to the Temperature Inputs (Figure 5.7).



**Figure 5.7.** Connecting NTC sensors type B57500M to the Temperature Inputs



Below is shown example connection how to control load with relays (lamp).

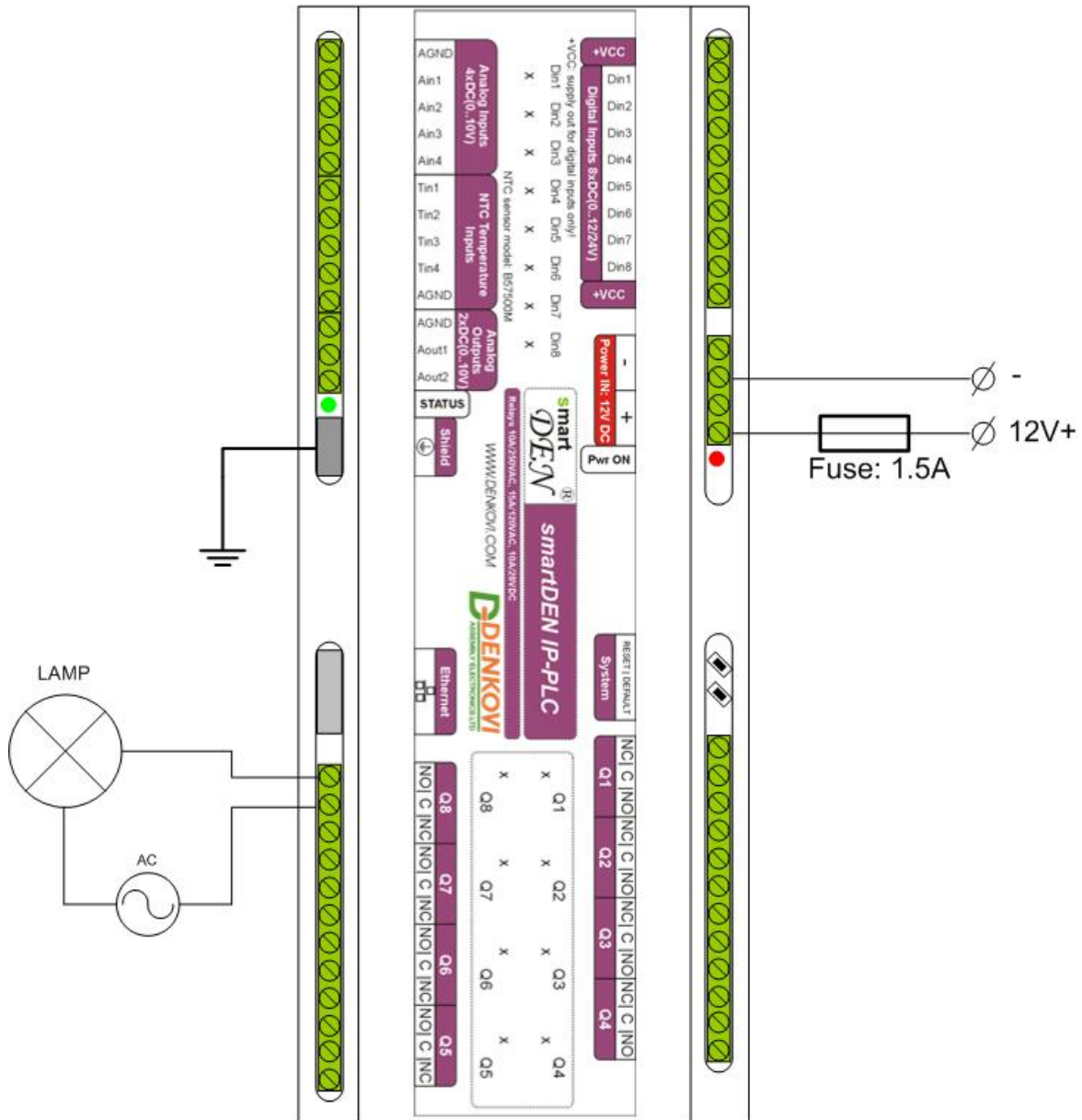
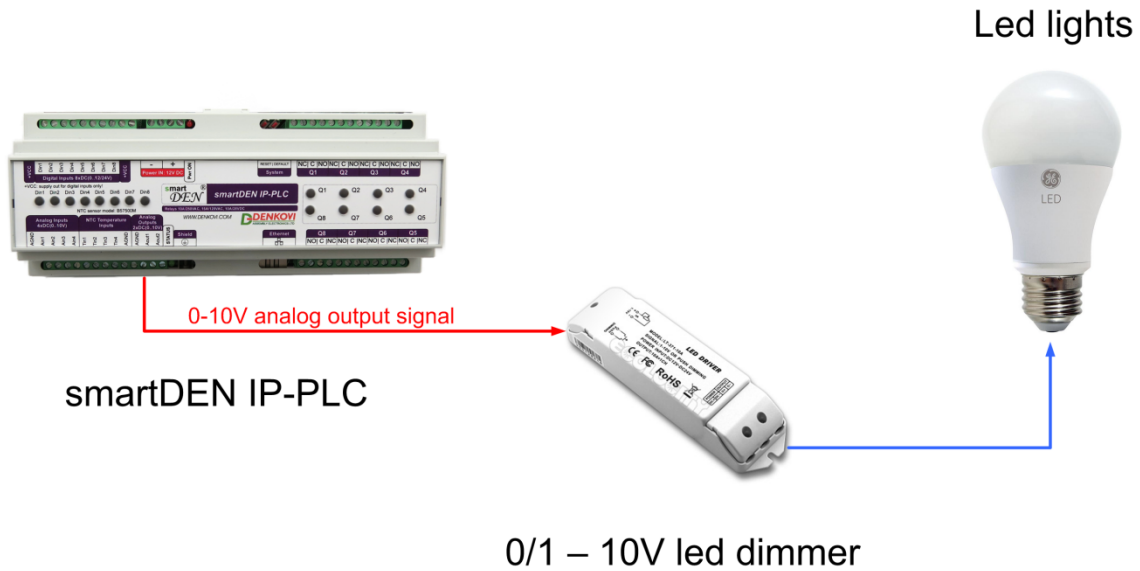


Figure 5.8. Connecting lamp to relay

The analog outputs (0-10V) can be used for dimming led lights as well. Please note that a separate dimmer could be necessary.



**Figure 5.9.** Controlling led lights with analog outputs

#### 5.4. Network connection

smartDEN IP-PLC supports AUTO-MDIX so either "crossover" or "straight-through" network cable can be used (Figure 5.10, Figure 5.11).



**Figure 5.10.** Direct connection of smartDEN IP-PLC to a computer



Figure 5.11. Connecting smartDEN IP-PLC to a wireless router

## 5.5. Communication setup

smartDEN IP-PLC ships with the following default parameters:

- IP address: 192.168.1.100
- Subnet mask: 255.255.255.0
- Gateway: 192.168.1.1
- Web password: admin

Initially it is recommended to connect the module directly to the computer.

Next you have to change your PC's IP address.



You can google how to change you computer IP settings or just visit this web page: <http://www.howtochangeipaddress.com/changeip.php>

For Windows 7 OS for example you can do that in the following way:

Navigate to *Control Panel -> Network and Internet -> View network and status tasks -> Change adapter settings*

Then just select the local area connection with right click and select *Properties* (Figure 5.12):

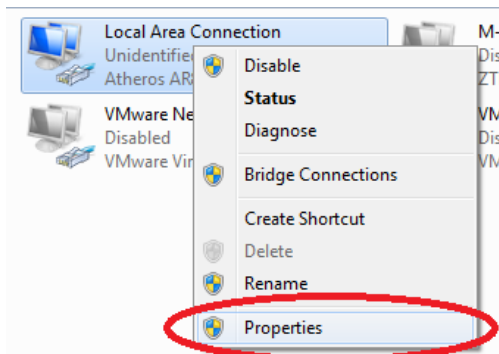
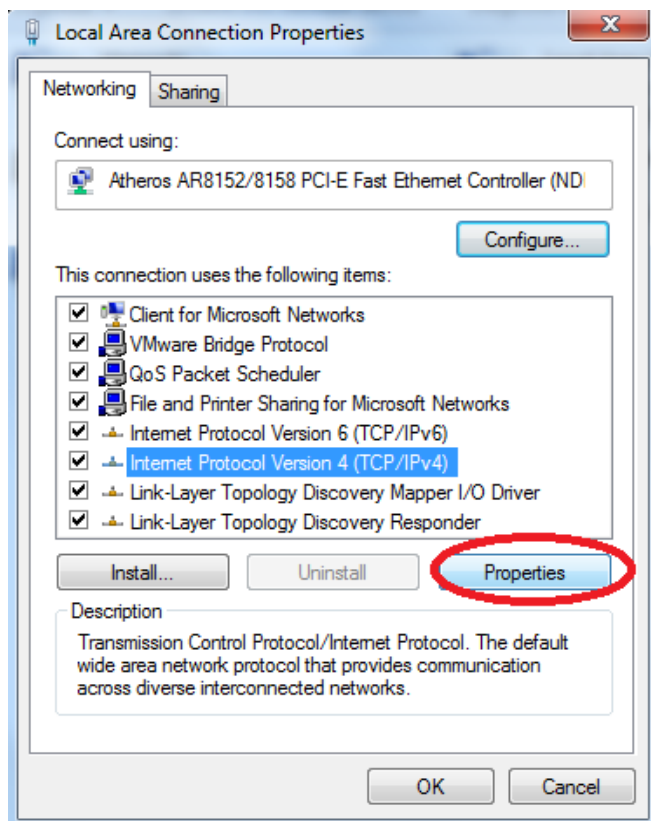


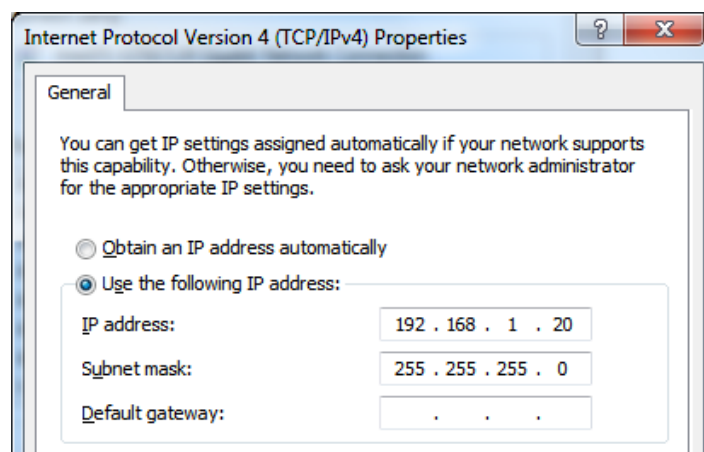
Figure 5.12. LAN card properties

The next step is to modify the IPv4 properties (Figure 5.13).



**Figure 5.13.** IPv4 properties section

Set the IP address of your PC to be in the same network as **smartDEN IP-PLC** (Figure 5.14).



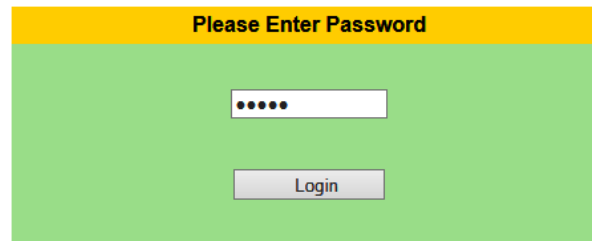
**Figure 5.14.** Set the IP address

Finally, in the address bar of your Web-browser enter the IP address of the **smartDEN IP-PLC** and press Enter, or select 'Go' (Figure 5.15).



**Figure 5.15.** Open the device in a browser

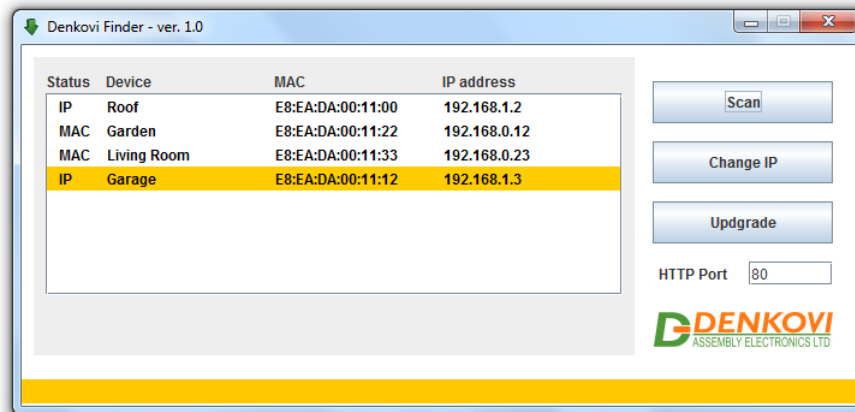
If the network settings are O'K, the login page should appear (Figure 5.16).



**Figure 5.16.** Login page



**smartDEN IP-PLC** modules connected locally can be easily scanned and found via the *Denkovi Finder* tool as well (Figure 5.17).



**Figure 5.17.** Denkovi Finder utility

## 6. Default settings

### 6.1. Table with default settings

The **smartDEN IP-PLC** module is shipped with default (factory) settings shown in Table 6.1. The default settings can be reloaded, if necessary (see point [6.2](#)).

**Table 6.1.** Default settings

Settings group	Parameter (according to web pages)	Value
General Settings	Device Name	SMARTDEN_IP-PLC
	Password	admin
	Temperature Scale	Celsius
	Analog Inputs Filter, sec	0
	Relays Filter, ms	0
	Save Outputs	Disabled
	Monitoring Timeout, sec	3
	Max. Monitoring Errors	5
Network Settings	DHCP	Disabled
	IP Address	192.168.1.100
	Gateway	192.168.1.1
	Subnet Mask	255.255.255.0
	Primary DNS	192.168.1.1
	Secondary DNS	0.0.0.0
Date/Time Settings	Date (dd/mm/yyyy)	Current
	Day of Week	Current
	Time (hh:mm)	Current
	Time Zone	GMT
	Auto Synchronization	Enabled
	Time Server	pool.ntp.org
	Server Port	123
	Synchronization Period, min	30
HTTP/XML/JSON Settings	HTTP Port	80
	Access IP Address	192.168.1.0
	Access Mask	0.0.0.0
	Access MAC Address	00:00:00:00:00:00
	Session Timeout, min	3
	Enable Access	Enabled
	Encrypt Password	Disabled
	Multiple Access	Enabled

Digital Inputs	Description	DIN<n>, where <n> = 1..8
	Counter	0
	Filter (ms)	0
	Edge	(0) Off
Analog Inputs	Description (In1 to In4)	AIN<n>, where <n> = 1..4
	Description (In5 to In8)	TIN<n>, where <n> = 1..4
	Min (In1 to In4)	0.0
	Max (In1 to In4)	10.0
	Min (In5 to In8)	-55
	Max (In5 to In8)	155
	Label (In1 to In4)	Volt
	Label (In5 to In8)	degC
Relays	Description	REL<n>, where <n> = 1..8
	Working Mode	Multiple
	DI No	0
	DI Mode	Normal
	AI+ No	0
	AI- No	0
	AI Threshold 1	0
	AI Threshold 2	0
	Pulse, ms (x100)	0
	Use Filter	No
Analog Outputs	Description	AOUT<n>, where <n> = 1..2
	Units	0

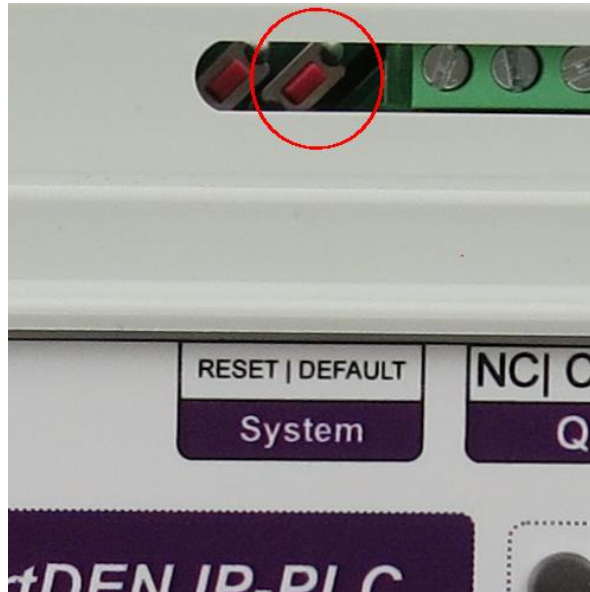
## 6.2. Steps for loading the default settings

When necessary, the factory (default settings) may be applied so the module parameters will be returned back to those pointed out in point [6.1](#) of the current document.

To return the settings to their factory default values next steps take place:

1. Turn Off the power supply of the device;
2. Press and hold the DEFAULT button (Figure 6.1);
3. Turn On the power supply of the device;
4. Wait until STATUS LED indicator become On (approximately after 10 sec);
5. Release the DEFAULT button;
6. The device is restarted and configured with the default settings.





**Figure 6.1.** Loading the default settings



## 7. DAEL elements

**DAEL** is a scripting language with a C-like syntax used for programming the **smartDEN IP-PLC**. **DAEL** programs consist of one or more statements. **DAEL** is a case sensitive language. There are only 4 keywords in **DAEL** – **if**, **else**, **while**, **delay** (always written in lowercase). Variable names must be typed in uppercase. Blanks and end-of-lines may be used freely between lexical elements.

### 7.1. Data types

**DAEL** uses only four data types, shown in Table 7.1.

Table 7.1. Data types

Data type	Format	Range
Integer	Signed 32-bit	$-2^{31}$ to $2^{31}-1$
Float	Single-precision 32-bit (IEEE-754)	$\approx \pm 3.4 \times 10^{38}$
Date	"DD/MM/YYYY"	from "01/01/2000" onwards
Time	"HH:MM"	from "00:00" to "23:59"

### 7.2. Constants

Constants are used as immediate operands in expressions. Integer constants are presented in decimal or hexadecimal. Hexadecimal constants are prefixed by **0x**. A constant that contains a decimal point is treated as floating-point number. Date and time constants should be enclosed in double quotes.

Examples:

123	// Integer constant 123
-123	// Integer constant -123
0x7E3D	// Integer constant 32317
3.14	// Floating-point constant
"20/01/2018"	// Date constant
"17:59"	// Time constant

### 7.3. Variables

**DAEL** uses only predefined variables. Part of them are used for accessing the **smartDEN IP-PLC** resources (inputs, outputs, counters, system date/time etc.). Sixteen custom variables for keeping intermediate values are predefined: eight for integer values (prefixed with **VAR\_I**) and eight for floating-point values (prefixed with **VAR\_F**). The predefined variables are listed in Table 7.2.

Table 7.2. Predefined variables

Name	Description	Type	Range	Access
DIN <sub>i</sub> (i=1..8)	Digital input	Integer	0..1	Read
DINS	Digital inputs (all)	Integer	0..255	Read
REL <sub>i</sub> (i=1..8)	Relay	Integer	0..1	Read/Write

RELS	Relays (all)	Integer	0..255	Read/Write
AIN <sub>i</sub> (i=1..8)*	Analog input	Float	-9999.9 .. 9999.9	Read
AOUT <sub>i</sub> (i=1..2)	Analog output	Integer	0..1023	Read/Write
CNT <sub>i</sub> (i=1..8)	Digital input counter	Integer	0.. 2 <sup>32</sup> -1	Read/Write
DATE	Current date	Date	DD/MM/YYYY	Read
YEAR	Current year	Integer	from 2000 onwards	Read
MONTH	Current month	Integer	1..12	Read
DAY	Current day	Integer	1..31	Read
DAYOFWEEK	Current day of week	Integer	1..7 (1 is Sunday)	Read
TIME	Current time	Time	HH:MM	Read
HOURL	Current hour	Integer	0..23	Read
MIN	Current minute	Integer	0..59	Read
TIMER	Software timer	Integer	0 to 2 <sup>32</sup> -1	Read/Write
VAR_I <sub>i</sub> (i=1..8)	Custom variables	Integer	-2 <sup>31</sup> to 2 <sup>31</sup> -1	Read/Write
VAR_F <sub>i</sub> (i=1..8)	Custom variables	Float	≈ ± 3.4 × 10 <sup>38</sup>	Read/Write

\*AIN1 to AIN4 are voltage analog inputs with range 0-10V DC, AIN5 to AIN8 are temperature inputs for NTC sensors B57500M.

## 7.4. Statements

By default, statements are executed in sequence, one after another. The sequence can be modified by using control flow statements. Two control flow statements are provided: **if-else** and **while**.

If the program is not executed in a **while** loop it will terminate after execution of the last statement.

Each statement is followed by a semicolon. Statements can be grouped in blocks delimited by curly braces ("{" and "}").

Statements cannot be a part of an expression. For example the assignment statement cannot form part of the conditional expression of **if-else** and **while** statements.

### 7.4.1. Assignment statement

The assignment statement (token '=', the equals sign) assigns a value to a variable.

Syntax:

```
variable = <expression>;
```

Examples:

```
REL1 = 1;           // Switch Relay 1 ON
RELS = 5;           // Switch Relays 1 and 3 ON, all others OFF
AOUT1 = 835;        // Set Analog Output 1 to 835 units
VAR_I1 = CNT5 * 2;  // Save Counter 5 multiplied by 2 in VAR_I1
VAR_F1 = AIN3 / 2.0; // Save Analog Input 3 value divided by 2 in VAR_F1
```

In assignment, the type of the right-hand value is converted automatically to the type of the left-hand variable.

Note that only variables with **Write** access can be used in the left-hand side of the assignment statement (see the last column of Table 7.2.). Using a **Read** only variable in the left-hand side does not cause an error, but no assignment takes place.

#### 7.4.2. "If-else" statement

The **if-else** statement conditionally executes a block of code. It can be used with or without **else** clause.

Syntax (**if**):

```
if ( <expression> )  
    <statement>
```

where **<expression>** is a conditional expression and **<statement>** is a single statement or block of statements.

Syntax (**if-else**):

```
if ( <expression> )  
    <statement1>  
else  
    <statement2>
```

where **<expression>** is conditional expression, **<statement1>** is a single or compound statement executed if the condition is true, and **<statement2>** is a single or compound statement executed if the condition is not met.

#### 7.4.3. "While" statement

The **while** statement executes a block of code as long as its condition is true.

Syntax:

```
while ( <expression> )  
    <statement>
```

where **<expression>** is a conditional expression and **<statement>** is a single statement or block of statements executed in a loop until the **<expression>** evaluates to false.

#### 7.4.4. "Delay" statement

The **delay** statement is used to introduce a delay in the program execution for a specified period.

Syntax:

```
delay(<period>);
```

where **<period>** specifies the time of the delay in tenths of a second (for example value 20 introduces a delay of 2 seconds).

## 7.5. Expressions

Variables and constants can be combined with operators to form expressions.

### 7.5.1. Arithmetic operators

Five basic arithmetic operators are provided:

<code>\+'</code>	addition
<code>\-</code>	subtraction
<code>\*</code>	multiplication
<code>\/'</code>	division
<code>\%</code>	modulus (remainder)

The `\-` operator can be used to subtract two numbers or to negate one number.

All five operators are left-associative. The multiplication, division and modulus have higher precedence than addition and subtraction.

The modulus `\%` operator can only be applied to integers.

### 7.5.2. Relational operators

The relational operators are used to form the conditional expressions for [if-else](#) and [while](#) statements.

The set of relational operators is:

<code>\&lt;'</code>	less than
<code>\&lt;='</code>	less than or equal
<code>\&gt;'</code>	greater than
<code>\&gt;='</code>	greater than or equal
<code>\=='</code>	equal
<code>\!=='</code>	not equal

The result of relational operators is an integer value of 1 or 0 depending on whether the relation is true or false.

### 7.5.3. Boolean operators

The results of relational operators can be combined to form complex conditional expressions by using boolean operators:

<code>\&amp;&amp;'</code>	logical AND
<code>\  '</code>	logical OR

The `\&&'` operator takes two true/false values and produces a true result if both operands are true, while the `\||'` operator produces a true result if either of operands is true. The logical AND has higher precedence than logical OR.

Example: Switch the [Relays 1](#) and [2](#) alternatively ON/OFF depending on the time of the day.

<pre>while(1)                                // Forever loop {     if (TIME &gt;= "07:00" &amp;&amp; TIME &lt; "19:00")    // Check the time     {</pre>
--

```
REL1 = 1;           // Switch Relay 1 ON
REL2 = 0;           // Switch Relay 2 OFF
}
else
{
    REL1 = 0;        // Switch Relay 1 OFF
    REL2 = 1;        // Switch Relay 2 ON
}
}
```

#### 7.5.4. Bitwise Operators

Bitwise operators work at bit-level:

'&'	bitwise AND
' '	bitwise OR
'^'	bitwise EXCLUSIVE OR

Example: Set the Relays with inverted value of Digital Inputs.

```
while(1)           // Forever loop
{
    RELS = DINS ^ 0xFF;
}
```

Note that both of the operands in bitwise operation must be of integer type.

#### 7.6. Comments

The comments in **DAEL** begin with `//` and continue up to the next line break.

#### 7.7. Type casting

When operands of an expression or assignment are of different type, automatic (implicit) type conversion is performed to match them.

For assignment statements, the type of the right-hand value is converted to the type of the left-hand variable. For expressions, if either operand has type float, the resulting type of the expression is float.

When converting a floating-point type to integer type, the fractional part is truncated. **Date** and **Time** type conversion to other type is not considered.

Examples:

<code>(1.5 + 2)</code>	<code>// result 3.5 (Float)</code>
<code>(2 + 1.5)</code>	<code>// result 3.5 (Float)</code>
<code>VAR_I1 = 3.14;</code>	<code>// VAR_I1 = 3 (Integer)</code>
<code>REL1 = 1.5;</code>	<code>// REL1 = 1 (ON)</code>
<code>VAR_F1 = 2;</code>	<code>// VAR_F1 = 2.0 (Float)</code>

#### 7.8. Limitations

The maximum size of the custom program is 10 000 chars.

While the typical response time of **DAEL** applications is  $\approx 100$  ms, they are not intended for special time-critical applications.

## 8. Working with I/O resources in DAEL

### 8.1. Digital Inputs

The Digital Inputs state is associated with  $DIN_i$  ( $i = 1..8$ ) and **DINS** variables. The **DINS** variable represents the state of all Digital Inputs, grouped at byte-level. The most significant bit of **DINS** is mapped to Digital Input 8, and the least significant – to Digital Input 1. For example if Digital Inputs 8, 7 and 3 are ON and all others OFF, the value of **DINS** is  $196_{10}$  ( $C4_{16}$ ,  $11000100_2$ ).

Note that the input variables have read-only access.

### 8.2. Digital Input Counters

The Digital Input Counters are linked to  $CNT_i$  ( $i = 1..8$ ) variables. These variables have read/write access. The range of the counters is from 0 to 4294967295.

### 8.3. Relays

The Relays state is associated with  $REL_i$  ( $i = 1..8$ ) and **RELS** variables. Read/write operations with **RELS** work at byte-level with Relay 8 mapped to the most significant bit and Relay 1 to the least significant bit.

Write operation to **RELS** changes the states of all Relays according to the value of individual bits in the operand ('1' – switch ON, '0' – switch OFF).

Write operation to  $REL_i$  ( $i = 1..8$ ) changes the state of a single Relay. In addition to values 1 (switch ON) and 0 (switch OFF), a value 2 can be used to toggle the Relay state between ON and OFF. Any other value written to  $REL_i$  won't change the state of the output.

If a particular Relay is configured in pulse mode, writing a value of 1 will generate a pulse with duration specified in the *Relays Settings* page.

Read operation returns the current state of the Relay(s).

### 8.4. Analog Inputs

$AIN_i$  ( $i = 1..8$ ) variables represent the measured values of Analog Inputs. Read operation of these variables returns a floating-point result. Write operation to  $AIN_i$  is not allowed.



Variables AIN1 to AIN4 represent the analog inputs 0-10V and the retrieved value is the scaled (measured) value of the input. AIN5 to AIN8 are the temperature inputs (TIN1 to TIN4) and the retrieved value is also the measured one (in degrees C or F depending on the selected option).



For variable AIN5 to AIN8 (TIN1 to TIN4) value of -273.1 means there is not connected NTC sensor to this input.

## 8.5. Analog Outputs

**AOUT<sub>i</sub>** ( $i = 1..2$ ) read/write variables represent the values of Analog Outputs, expressed in units. The range is from 0 to 1023. Writing a value  $\leq 0$  will set 0, and a value  $\geq 1023$  will set 1023.

## 8.6. Timer

**TIMER** variable represents a software timer that counts down from a specified time interval. The interval is set in tenths of a second. When a new value is written to the timer, it starts to count down every 0.1 seconds until reaches 0. For example, if the timer value is set to 50, the timer will expire in 5 seconds.

## 8.7. Sharing of output resources

By default, Relays and Analog Outputs are shared between the custom program and other functions. For example, when the program works with particular relays or analog outputs, at the same time their values can be changed from *Monitoring & Control* page or by XML/JSON request.

To control the write access to the output resources, **DAEL** provides two additional operators: **lock** and **unlock**.

Syntax:

```
lock(<output>);  
unlock(<output>);
```

where **<output>** is **REL<sub>i</sub>** ( $i = 1..8$ ) or **AOUT<sub>i</sub>** ( $i = 1..2$ ).

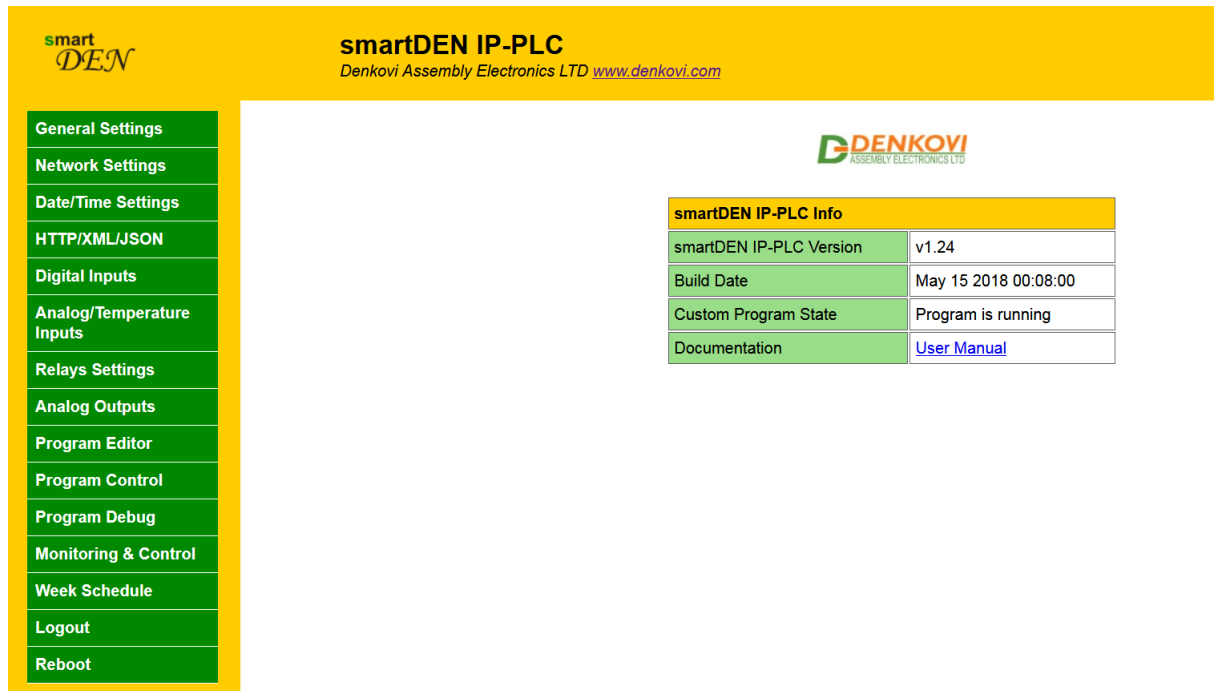
The **lock** operator locks the specified output resource for exclusive write access by the custom program (the other functions are granted only read access). The **unlock** operator returns the output resource to “shared writing” mode.

Upon stopping or terminating the program all the Relays and Analog Outputs are “unlocked” automatically.

Example:

```
lock(REL1);           // Obtain exclusive write access to Relay 1  
while(1)  
{  
    REL1 = REL1 ^ 1; // Toggle the Relay 1 state  
    delay(10);       // Delay of 1 sec  
}
```

## 9. Web access



**Figure 9.1.** Web access

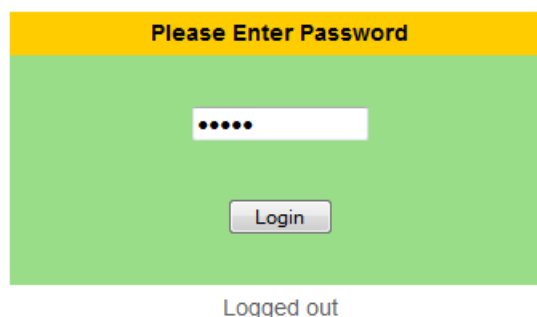
To access the setup pages, start a web browser (Internet Explorer, Chrome, Mozilla Firefox or similar), and enter the **smartDEN IP-PLC** IP address, for example: <http://192.168.1.100>.



**Figure 9.2.** Open the device in a browser

**Note:** You will need to have **JavaScript** enabled in your browser.

### 9.1. Login



**Figure 9.3.** Login page

Enter the password and click Login button or press Enter (Figure 9.3). This will bring you to the **smartDEN IP-PLC** main configuration page, which contains details



of the current firmware version/build date and provides a link to the documentation page (Figure 9.4).

**Note:** The default password is admin (passwords are case sensitive).

**Note:** When the password is entered, it is transmitted across the network in encrypted form, so eavesdropping on the data transmission will not reveal the password.

**Note:** In order to prevent setup/control conflicts, at any given moment, only one user can be logged in.



smartDEN IP-PLC Info	
smartDEN IP-PLC Version	v1.24
Build Date	May 15 2018 00:08:00
Custom Program State	Program is running
Documentation	<a href="#">User Manual</a>

**Figure 9.4.** Version/Build Date info

## 9.2. Menu

The main menu (Figure 9.5) consists of the following items, located in the left window frame:

General Settings
Network Settings
Date/Time Settings
HTTP/XML/JSON
Digital Inputs
Analog/Temperature Inputs
Relays Settings
Analog Outputs
Program Editor
Program Control
Program Debug
Monitoring & Control
Week Schedule
Logout
Reboot

**Figure 9.5.** Navigation menu

### 9.3. General Settings

General Settings page is shown in Figure 9.6.

**General Settings**

General Settings	
Device Name	SMARTDEN_IP-PLC
Password	*****
Temperature Scale	Celsius
Analog Inputs Filter, sec	0
Relays Filter, ms	0
Save Outputs	<input type="checkbox"/>
Monitoring Timeout, sec	3
Max. Monitoring Errors	5

**Figure 9.6.** General Settings page

- **Device Name** - the name of the module (max 15 symbols). Every module can have different name in your network so they can be distinguished;
- **Password** - the password used for logging into the web admin and XML/JSON operation (max. 10 chars);



When typed, the password in this screen is not hidden. Only in this case, when the password is being changed, it is transmitted across the network "in the open". Therefore, set passwords in a secure environment where you can make sure that no one is "eavesdropping". Subsequent transmissions of the password to "login" onto the device are encrypted and "safe".

- **Temperature Scale** – Celsius / Fahrenheit;
- **Analog Inputs Filter, sec** – analog inputs filter constant. The range is from 0 up to 30 sec. The filter is disabled when its value is set to 0;



This parameter sets a low pass software filter that removes the short-term fluctuations from the input signal and reduces the effect of occasional spikes. Note that the higher filter constants give a slower response to changes.

- **Relays Filter, ms** – relays filter constant. The range is from 0 up to 9999 milliseconds. Zero value disables the filter;



In real applications, switching external loads can produce spikes and noise transients that may disturb the Analog Inputs measurement. To avoid the processing of "false" Analog Inputs measurements during these transients the parameter Relays Filter can be set to appropriate value. When a Relay changes its state, during the defined period Analog Input measurements will be ignored. The filter can be enabled or disabled individually for each relay by Use **Filter** option in *Relays Settings* page.

- **Save Relays** - when checked, each time a Relay or Analog Output value is changed, it will be saved in non-volatile memory (EEPROM), so after reboot/restart it will be restored;

! This option should be used with care in dynamic systems because of restriction in maximum write cycles of the EEPROM (usually 100 000 write/erase cycles).

- **Monitoring Timeout, sec** - the connection timeout for the Web-browser;
- **Max. Monitoring Errors** – the number of successive timeouts before the connection error is reported by the Web-browser;
- **Save button** - once you have changed the settings as required, click this button.

## 9.4. Network settings

The page shown in Figure 9.7 lets you configure the network settings of smartDEN IP-PLC module.

Network Settings	
MAC Address	E8:EA:DA:00:00:20
Enable DHCP	<input type="checkbox"/>
IP Address	192.168.1.100
Gateway	192.168.1.1
Subnet Mask	255.255.255.0
Primary DNS	192.168.1.1
Secondary DNS	0.0.0.0
<input type="button" value="Save"/> <input type="button" value="Reload"/>	

Figure 9.7. Network Settings page

- **Enable DHCP** - this option allows DHCP to be enabled or disabled. If DHCP is set to Enabled, the *Network* page must be saved and **smartDEN IP-PLC** must be rebooted before obtaining an IP address;
- **IP address** - this is the IP address of the **smartDEN IP-PLC**. It has to be manually assigned only if DHCP is disabled. With DHCP enabled, this field displays the currently assigned address;
- **Gateway** - this specifies the IP address of the gateway router. It is used for accessing public SNTP servers for automatic time synchronization;
- **Subnet Mask** - this is the subnet mask for the network on which the **smartDEN IP-PLC** is installed;
- **Primary DNS** - primary DNS (Domain Name Service) address;
- **Secondary DNS** - secondary DNS address;
- **Save button** - once you have changed the settings as required, click this button.

! You have to reboot the device for these settings to apply.

## 9.5. Date/Time Settings

This page lets you adjust the date/time and set the SNTP (Simple Network Time Protocol) server for auto synchronization (Figure 9.8).

Date/Time Settings	
Date (dd/mm/yyyy)	<input type="text" value="28/02/2018"/>
Day of Week	<input type="text" value="Wed"/>
Time (hh:mm)	<input type="text" value="17:15"/>
Time Zone	<input type="text" value="(GMT)"/> ▼
Auto Synchronization	<input checked="" type="checkbox"/>
Time Server	<input type="text" value="pool.ntp.org"/>
Server Port	<input type="text" value="123"/>
Synchronization Period, min	<input type="text" value="30"/>
<input type="button" value="Save"/> <input type="button" value="Reload"/>	

**Figure 9.8.** *Date/Time Settings* page

- **Date (dd/mm/yyyy)** - the current date in specified format;
- **Time (hh:mm)** - the current time in 24-hour format;
- **Time Zone** - select the time zone for your geographic location;
- **Auto Synchronization** - this option enables or disables automatic synchronization with the SNTP server with period specified by Synchronization Period;
- **Time Server** - the SNTP server, used for synchronizing the time automatically;
- **Server Port** - the SNTP server port;
- **Synchronization Period, min** - the period in which automatic synchronization will take place, if enabled;
- **Save button** - once you have changed the settings as needed, click this button. These settings apply immediately and do not require a reboot.

## 9.6. HTTP/XML/JSON

These settings let you configure the HTTP and XML/JSON access parameters of smartDEN IP-PLC (Figure 9.9).

HTTP Access	
HTTP Port	<input type="text" value="80"/>
Access IP Address	<input type="text" value="192.168.1.0"/>
Access Mask	<input type="text" value="0.0.0.0"/>
Access MAC Address	<input type="text" value="00:00:00:00:00:00"/>
Session Timeout, min	<input type="text" value="30"/>
XML/JSON Access	
Enable Access	<input checked="" type="checkbox"/>
Encrypt Password	<input type="checkbox"/>
Multiple Access	<input checked="" type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="Reload"/>	

**Figure 9.9.** HTTP/XML/JSON Settings page







- **HTTP Port** - port on which the integrated Web server listens for HTTP requests (default port is 80). You have to reboot the device for a new port setting to apply;
- **Access IP Address/Access Mask** - these fields can be used to restrict the HTTP/XML/JSON access by specifying the IP address and subnet mask of the HTTP client;
- **Access MAC Address** - this field can be used to restrict the HTTP/XML/JSON access by specifying the MAC address of the HTTP client;
- **Session Timeout, min** - specifies the timeout period for the HTTP/XML/JSON sessions in minutes;
- **Enable Access** - this option enables or disables the XML/JSON access;
- **Encrypt Password** - when the XML/JSON access is enabled, this option adds additional security level by encrypting the login password;
- **Multiple Access** - this option enables simultaneous XML/JSON access from several HTTP clients;
- **Save button** - once you have changed the settings as required, click this button.

**Note:** If there is no traffic between the Web-browser/HTTP client and the smartDEN IP-PLC for time, specified by Session Timeout parameter, the session "times out" and a new login is required.

**Note:** When **Encrypt Password** mode is enabled, the **Multiple Access** option is not taken into account, so at any given moment, only one user can be logged-in.

**Note:** When **Multiple Access** mode is enabled, any XML/JSON request will always reset the current HTTP session.

**Note:** When **Multiple Access** mode is disabled, whether **Encrypt Password** is enabled or not, it is possible to access the module via XML/JSON only after login for the specified session timeout.

-  You have to reboot the device for these settings to apply.
-  It is highly recommended to log out from the web server after finishing the parameters setup.
-  If you don't want to restrict the HTTP/XML/JSON access by IP address, set the **Access Mask** to 0.0.0.0.
-  If you don't want to restrict the HTTP/XML/JSON access by MAC address, set the MAC Address to 00:00:00:00:00:00.
-  Setting the **Access Mask** to 255.255.255.255 allows the HTTP/XML/JSON access only from the exactly specified Access IP Address.
-  You can allow the HTTP/XML/JSON access to a range of IP addresses by setting an appropriate value for **Access Mask**. For example setting the **Access IP Address** to 192.168.1.0 and **Access Mask** to 255.255.255.0 allows the access from IP addresses in range from 192.168.1.0 to 192.168.1.255.

## 9.8. Digital Inputs

Digital Inputs web page is shown in Figure 9.10.

Digital Input	Description	Counter	Filter (ms)	Edge
Input 1	DIN1	0	0	0 (Off) ▼
Input 2	DIN2	0	0	0 (Off) ▼
Input 3	DIN3	0	0	0 (Off) ▼
Input 4	DIN4	0	0	0 (Off) ▼
Input 5	DIN5	0	0	0 (Off) ▼
Input 6	DIN6	0	0	0 (Off) ▼
Input 7	DIN7	0	0	0 (Off) ▼
Input 8	DIN8	0	0	0 (Off) ▼

**Figure 9.10.** Digital Inputs page

- **Description** - identification string of the input (max. 7 chars);



This description will appear in XML/JSON files, as well as in the *Monitoring & Control* page.

- **Counter** – the value of 32-bit counter attached to the input. The **Counter** is incremented on rising (On), falling (Off), or both edges depending on the **Edge**. The **Counter** is cyclic and its value can be set or cleared by the user. The range for this parameters is from 0 up to  $2^{32}-1$  (4294967295);
- **Filter (ms)** - the input may be adjusted to work with a digital filter. It is valid for counting and input visualization. The range for this parameter is from 0 up to 200 milliseconds. The filter is disabled when its value is set to 0;
- **Edge** - this parameter determines the condition to increment the counter:
  - 0 (Off) – the **Counter** is incremented on the falling edge (1 -> 0).
  - 1 (On) – the **Counter** is incremented on the rising edge (0 -> 1).
  - 2 (Both) – **Counter** update takes place on each Digital Input state change (both falling and rising edges).
- **Save button** - once you have changed the settings as required, click this button.



Inputs/Counters 1, 2, 3 and 8

These counters are implemented in software but not in hardware. Due to this there are some considerations which must be taken in mind when working with them:

- When the **Filter** is disabled (set to 0) and no requests are sent to the controller (no HTTP, logged out from web server) but just counting pulses, it is possible to achieve frequency about 10 Hz or this is 50 ms



On and 50 ms Off. This is the recommend state for counting pulses;

- When the **Filter** is disabled (set to 0) but there is active web session and the *Monitoring & Control* page is opened, then it is possible to count pulses with frequency about 1 Hz or this is 500 ms On and 500 ms Off;
- When the Filter is enabled the actual frequency depends also on the filter constant.
- At the moment the module is saving the configuration in the EEPROM some count pulses may be missed due to the blocking time which may be several seconds.



Inputs/Counters 4, 5, 6 and 7

- When the **Filter** is disabled (set to 0) it is possible to count pulses with frequency up to 9 KHz. Note that it is not appropriate at high frequencies to configure these inputs to send traps or to control Relays.
- When the Filter is enabled, these inputs work in the same mode as Inputs/Counters 1, 2, 3 and 8.

## 9.9. Analog/Temperature Inputs

*Analog Inputs* web page is shown in Figure 9.11. Inputs 1 through 4 are used to measure voltage in range 0-10VDC. Inputs 5 through 8 are used to measure temperature from 10K NTC sensors B57500M (with range temperature range -55°C/-67°F to +155°C/311°F).

**Analog/Temperature Inputs Settings**

Analog Inputs

Temperature Inputs

Analog Input	Description	Min	Max	Label
Input 1 (Ain1)	Ain1	0.0	1023.0	Unit
Input 2 (Ain2)	Ain2	0.0	1023.0	Unit
Input 3 (Ain3)	Ain3	0.0	1023.0	Unit
Input 4 (Ain4)	Ain4	0.0	1023.0	Unit
Input 5 (Tin1)	Tin1	-55.0	155.0	degC
Input 6 (Tin2)	Tin2	-55.0	155.0	degC
Input 7 (Tin3)	Tin3	-55.0	155.0	degC
Input 8 (Tin4)	Tin4	-55.0	155.0	degC

This is configuration page only, for monitoring please go to [Monitoring & Control page](#)

**Figure 9.11.** *Analog Inputs settings*

- **Description** - identification string of the input (max. 7 chars);  
💡 This description will appear in XML/JSON files, as well as in the *Monitoring & Control* page.
- **Min** - the measured value used for scaling corresponding to Analog Input value = 0:
  - For Input 1 to Input 4 the range is from -9999.9 to +9999.9;
  - For Input 5 to Input 8 the range is from -70.0 to +300.0 degF or -55.0 to +150.0 degC;
- **Max** - the measured value used for scaling corresponding to Analog Input value = 1023:
  - For Input 1 to Input 4 the range is from -9999.9 to +9999.9;
  - For Input 5 to Input 8 the range is from -70.0 to +300.0 degF or -55.0 to +150.0 degC;
- **Label** - the label for the measured value, for example: mA, Kg, Volt ...(max. 4 chars), available for editing only for Input 1 to Input 4. For inputs 5 to 8, it is fixed (degC or degF);

💡 The measured value for the Analog Inputs (0-10V) is calculated in the following way:

$$MeasuredValue = Min + \frac{Max - Min}{1024} \cdot ADCValue$$



## 9.10. Relays Settings

Relays Settings web page is shown in Figure 9.12.

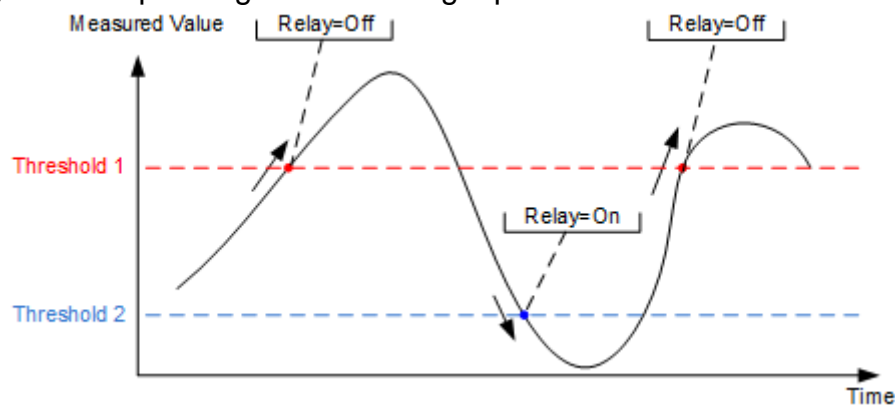
Relay	Description	Working Mode	DI No	DI Mode	AI+ No	AI- No	AI Threshold 1	AI Threshold 2	Pulse, ms (x100)	Use Filter
Relay 1	REL1	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 2	REL2	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 3	REL3	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 4	REL4	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 5	REL5	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 6	REL6	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 7	REL7	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>
Relay 8	REL8	Multiple	0	Normal	0	0	0	0	0	<input type="checkbox"/>

Save Reload

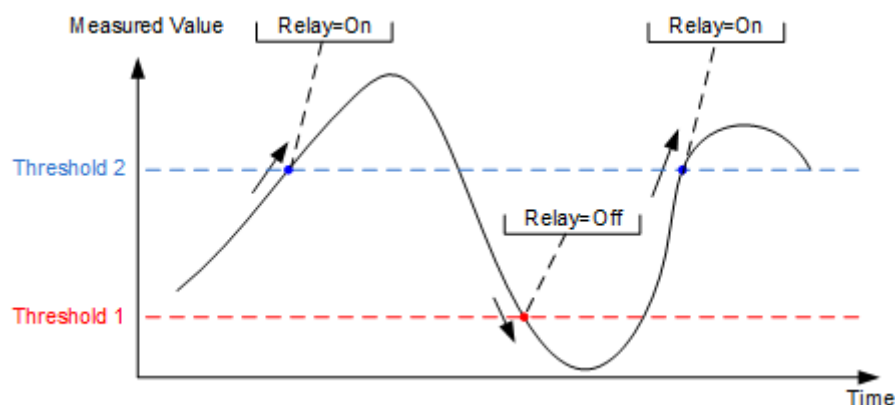
Figure 9.12. Relays Settings page

- **Description** - identification string of the relay (max 7 chars);  
 This description will appear in XML/JSON files, as well as in the *Monitoring & Control* page.
- **Working Mode** - determines the way the Relay is controlled:
  - *Regulator* - in this mode the Relay can be controlled only by an Analog Input. This mode is designed for cases where the input value depends on the output reaction (like thermo-regulator for example). In this mode the parameters DI No, DI Mode and Pulse are not taken in mind. Also in this mode it is not possible to control the Relay via Web browser or HTTP/XML/JSON protocol.
  - *Multiple* - in this mode the Relay can be controlled by Analog Inputs, Digital Inputs, Week Schedule or manually (via Web browser, HTTP/XML/JSON). This mode is designed for cases where the input event does not depend on the output reaction (for example controlling lamp depending on the sunlight);
- **DI No** – if set to non-zero (1 to 8) this is the number of the Digital Input which is "attached" to the Relay.  
 The Relay can be controlled by a Digital Input only in Multiple mode.
- **DI Mode** – one of eight level/edge control modes can be selected:
  - *Normal* - the Relay state is On when the Digital Input state is On and Off when the Digital Input state is Off;
  - *Inverse* - the Relay state is On when the Digital Input state is Off and Off when the Digital Input state is On;
  - *Toggle(DI=0)* - a falling edge (1 -> 0) of the Digital Input toggles the Relay state between On and Off;
  - *Toggle(DI=1)* - a rising edge (0 -> 1) of the Digital Input toggles the Relay state between On and Off;
  - *Normal(DI=0)* - a falling edge (1 -> 0) of the Digital Input switches the Relay Off;
  - *Normal(DI=1)* - a rising edge (0 -> 1) of the Digital Input switches the Relay On;
  - *Inverse(DI=0)* - a falling edge (1 -> 0) of the Digital Input switches the Relay On;

- *Inverse(DI=1)* - a rising edge (0 -> 1) of the Digital Input switches the Relay Off.
- **AI+ No** - the number of the Analog Input (1 to 8) which is "attached" to this Relay. If AI- No is set to zero, this is the single-ended input, otherwise – the (+) lead of the differential input.
- **AI- No** – if set to non-zero (1 to 8), this is the (-) lead of the differential input;
- 💡 If **AI+ No** is non-zero and **AI- No** is zero, the Relay is controlled by the **AI+ No** measured value.
- 💡 If both **AI+ No** and **AI- No** are non-zero, the Relay is controlled by the difference of **AI+ No** and **AI- No** measured values.
- 💡 If **AI+ No** is zero, the Relay is not controlled by the Analog Input (in this case the **AI- No** is not relevant).
- **AI Threshold 1** - the first level for controlling the Relay (range: from -9999 to +9999);
- **AI Threshold 2** - the second level for controlling the Relay (range: from -9999 to +9999);
- 💡 The Relay can be controlled in "normal" (Figure 9.13) or "inverse" (Figure 9.14) mode depending on the Analog Input threshold values.



**Figure 9.13.** Normal mode: AI Threshold 1 > AI Threshold 2



**Figure 9.14.** Inverse mode: AI Threshold 1 < AI Threshold 2

- **Pulse, ms (x100)** – if set to non-zero, the Relay operates in pulse mode. The pulse duration is set in tenths of second (for example value of 20 means 2 seconds). The range is from 0 and 65535 (0 disables the pulse mode). When enabled, each time the Relay is switched On, it will generate a single pulse with specified duration.



The Relay can generate pulses only if it's Working Mode=Multiple.



The Relay can be switched On/Off by “attached” Digital or Analog Input, HTTP/XML/JSON command, Week Schedule, and manually from Web *Monitoring & Control* page.



At any time the pulse is active the Relay can be switched back Off by “attached” Digital or Analog Input or HTTP/XML/JSON command, and manually from web *Monitoring & Control* page.

## 9.11. Analog Outputs

Analog Output	Description	Units (0..1023)
Output 1	AOUT1	392
Output 2	AOUT2	810

Save Reload

Figure 9.15. Analog Outputs page

- **Description** - identification string (max 7 chars);
- This description will appear in XML/JSON files, as well as in the *Monitoring & Control* page.
- **Units (0..1023)** – Analog Output value (0 to 1023 corresponds to a linear scale of 0V DC to 10V DC).

## 9.12. Custom Program Editor

This page is used to compose, upload and start/stop the custom program (Figure 9.16).

**Custom Program Editor**

**Program Code**

```

////////////////////////////////////
//      Thermostat      //
// Set point (SP)       : 20 degrees C //
// Hysteresis (H)       : 2 degrees C  //
// Alarm threshold     : 35 degrees C  //
// Process value (PV)  : AIN8         //
// Heater output       : Relay 1       //
// Alarm output        : Relay 2       //
////////////////////////////////////
// Initialization
VAR_F1 = 20.0;          // Set point in VAR_F1
REL2 = 0;               // No alarm at start
// Evaluate the initial heater state
if (AIN8 > VAR_F1) {
    REL1 = 0;           // Switch the heater OFF
} else {
    REL1 = 1;           // Switch the heater ON
}
////////////////////////////////////
//      Control loop      //

```

Characters left: 8779

Program is stopped

Browse... Load

Upload Run ☐ Don't reset custom variables

**Figure 9.16.** Custom Program Editor page

The following actions are available:

- **Browse:** Select a local text (.txt) file containing the program source;
- **Load:** Load the selected text file into the text area;
- **Upload:** Check the program for errors and then upload it to the device;
- **Run/Stop:** Start/Stop the program execution;
- **Don't reset custom variables:** If unchecked, the custom variables will be cleared at program start-up.

The status bar below the text area uses different colors to show the state of the program:

- The custom program is not defined yet

Program is not defined

- The program upload is in progress

Upload in progress, please wait...

- The program source is modified since the last upload

Program is modified

- The program check reports an error in the source code

Error: Parse error on line 12: ...Expecting '(', got 'b'

- The program is running

Program state: Running

- The program is stopped by the user or terminated

Program state: Stopped

Note that programs not executed in an infinite loop will terminate after execution of its last statement.

Upon stopping or terminating the program the relays are switched OFF and the analog outputs are set to 0.

If the device is switched OFF when the program is running, the next time it is switched ON the program will resume from its start point.

### 9.13. Custom Program Control

This page provides visualization of the program state and variables (Figure 9.18).

**Custom Program Control**

I/O Variables							
DIN1	0	REL1	1	AIN1	0.0	CNT1	1
DIN2	0	REL2	0	AIN2	0.0	CNT2	23
DIN3	0	REL3	0	AIN3	0.0	CNT3	45
DIN4	0	REL4	0	AIN4	0.1	CNT4	0
DIN5	1	REL5	0	AIN5	-273.1	CNT5	350
DIN6	0	REL6	0	AIN6	-273.1	CNT6	0
DIN7	1	REL7	0	AIN7	-273.1	CNT7	140
DIN8	0	REL8	0	AIN8	28.8	CNT8	0
DINS 80		RELS 1		AOUT1 0		AOUT2 0	
DATE 07/08/2018		TIME 14:28		DAYOFWEEK 3		TIMER 0	
Custom Variables							
VAR_I1	12	0	Set	VAR_F1	0.0	0.0	Set
VAR_I2	24	0	Set	VAR_F2	0.5	0.5	Set
VAR_I3	0	0	Set	VAR_F3	22.9	0	Set
VAR_I4	34	0	Set	VAR_F4	0.0	0.0	Set
VAR_I5	-12	0	Set	VAR_F5	0.0	0.0	Set
VAR_I6	2	2	Set	VAR_F6	0.0	0.0	Set
VAR_I7	0	0	Set	VAR_F7	-985.3	-985.3	Set
VAR_I8	16	0	Set	VAR_F8	16.0	0.0	Set
Program state: Running							
Stop		<input checked="" type="checkbox"/> Don't reset custom variables					

**Figure 9.18.** Custom Program Control page

The following actions are available:

- Set:** Assign a new value to a custom variable;
- Run/Stop:** Start/Stop the program execution;
- Don't reset custom variables:** If unchecked, the custom variables will be cleared at program start-up.



In addition, the program state and variables are displayed in the bottom fragment of *Monitoring & Control* page (Figure 9.19).

Program state: Stopped							
VAR_I1	VAR_I2	VAR_I3	VAR_I4	VAR_I5	VAR_I6	VAR_I7	VAR_I8
12	24	0	34	-12	0	0	20
VAR_F1	VAR_F2	VAR_F3	VAR_F4	VAR_F5	VAR_F6	VAR_F7	VAR_F8
0.0	0.5	22.9	0.0	0.0	0.0	-985.3	20.0

Figure 9.19. Fragment of *Monitoring & Control* page

## 9.14. Custom Program Debug

The *Custom Program Debug* page provides a tool for debugging **DAEL** applications (Figure 9.20). Typical debugging tasks include stepping through code, pausing at breakpoints and viewing/modifying the state of variables.

The debugger window consists of the following elements:

- **Program code:** Source code with numbered lines (current executed line marked in blue);
- **I/O Variables:** I/O variables with their current values;
- **Custom Variables:** Custom variables with option to edit and set their values;
- **Breakpoints:** Breakpoints with option to edit and set new code lines;
- **Status bar:** Indicates the current state of the program;
- **Debug commands**

**Program Code**

```

1. // Example 5.5
2. ////////////////////////////////////////////////////
3. //      Thermostat      //
4. // Set point (SP)       : 20 degrees C      //
5. // Hysteresis (H)       : 2 degrees C      //
6. // Alarm threshold      : 30 degrees C      //
7. // Process value (PV)   : AIN8             //
8. // Heater output        : Relay 1           //
9. // Alarm output         : Relay 2           //
10. ////////////////////////////////////////////////////
11. // Initialization
12. VAR_F1 = 20.0; // Set point in VAR_F1
13. REL2 = 0; // No alarm at start
14. // Evaluate the initial heater state
15. if (AIN8 > VAR_F1) {
16.     REL1 = 0; // Switch the heater OFF
17. } else {
18.     REL1 = 1; // Switch the heater ON
19. }
20. ////////////////////////////////////////////////////
21. //      Control loop      //

```

**I/O Variables**

DIN1	0	REL1	0
DIN2	0	REL2	0
DIN3	0	REL3	0
DIN4	0	REL4	0
DIN5	1	REL5	0
DIN6	0	REL6	0
DIN7	1	REL7	0
DIN8	0	REL8	0

AIN1	0.0	CNT1	1
AIN2	0.0	CNT2	23
AIN3	0.0	CNT3	45
AIN4	0.1	CNT4	0
AIN5	-273.1	CNT5	350
AIN6	-273.1	CNT6	0
AIN7	-273.1	CNT7	143
AIN8	28.9	CNT8	0

DINS	80	DATE	07/08/2018
RELS	0	TIME	14:38
AOUT1	0	DAYOFWEEK	3
AOUT2	0	TIMER	0

**Breakpoints**

BP1	25	25	Set	BP2	33	33	Set
-----	----	----	-----	-----	----	----	-----

**Program state: Paused**

☐ Don't reset custom variables

**Custom Variables**

VAR_I1	12	0	Set	VAR_F1	0.0	0.0	Set
VAR_I2	24	0	Set	VAR_F2	0.5	0.5	Set
VAR_I3	0	0	Set	VAR_F3	22.9	0	Set
VAR_I4	34	0	Set	VAR_F4	0.0	0.0	Set
VAR_I5	-12	0	Set	VAR_F5	0.0	0.0	Set
VAR_I6	0	0	Set	VAR_F6	0.0	0.0	Set
VAR_I7	0	0	Set	VAR_F7	-985.3	0	Set
VAR_I8	20	0	Set	VAR_F8	20.0	0	Set

Figure 9.20. *Custom Program Debug* page

The following actions are available:

- **Start Debug/Stop Debug:** Start/Stop the debug session;
- **Don't reset custom variables:** If unchecked, the custom variables will be cleared at debug start-up;
- **Step:** Perform a step through code;
- **Run to breakpoint:** Execute the program to a breakpoint;

Up to two breakpoints can be set on selected code lines (value 0 disables the breakpoint).

## 9.15. Monitoring & Control

This page is used to monitor the states of the Digital and Analog Inputs as well as to switch On/Off the Relays and set the values of Analog Outputs (Figure 9.21).



A relay can be switched On/Off from the *Monitoring & Control* page only if it's **Working Mode="Multiple"**.

**Monitoring & Control**

Relays							
REL1	REL2	REL3	REL4	REL5	REL6	REL7	REL8
Off ▼	Off ▼	Off ▼	Off ▼	Off ▼	Off ▼	Off ▼	Off ▼
Toggle	Toggle	Toggle	Toggle	Toggle	Toggle	Toggle	Toggle

Digital Inputs							
DIN1	DIN2	DIN3	DIN4	DIN5	DIN6	DIN7	DIN8
0 (Off)	0 (Off)	0 (Off)	0 (Off)	0 (Off)	0 (Off)	0 (Off)	0 (Off)
0	0	0	0	0	0	0	0

Analog Inputs							
AIN1	AIN2	AIN3	AIN4	TIN1	TIN2	TIN3	TIN4
0	0	0	0	839	839	839	839
0.0 Unit	0.0 Unit	0.0 Unit	0.0 Unit				

Analog Outputs	
AOUT1	AOUT2
0 <input type="text"/> 0 Set	0 <input type="text"/> 0 Set

Program state: Stopped							
VAR_I1	VAR_I2	VAR_I3	VAR_I4	VAR_I5	VAR_I6	VAR_I7	VAR_I8
0	0	0	0	0	0	0	0
VAR_F1	VAR_F2	VAR_F3	VAR_F4	VAR_F5	VAR_F6	VAR_F7	VAR_F8
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 9.21. Monitoring & Control page

## 9.16. Week Schedule

This page configures the week schedule table for starting/stopping custom program at specific times. You can add up to 30 items to the list. The top table of this page allows you to define a new item, while the bottom table shows the already defined list (Figure 9.22).

## Week Schedule

New Item (Remaining Items: 26)

Program State	Hour (hh:mm)	Days of Week							Start Date (dd/mm/yyyy)
<input type="button" value="Stop"/> <input type="button" value="Start"/>	10:00	<input type="checkbox"/> Sun	<input checked="" type="checkbox"/> Mon	<input checked="" type="checkbox"/> Tue	<input type="checkbox"/> Wed	<input type="checkbox"/> Thu	<input type="checkbox"/> Fri	<input type="checkbox"/> Sat	14/05/2018
<input type="button" value="Add"/> <input type="button" value="Reload"/>									

Existing Items (Start Date: 14/05/2018)

No	Program State	Hour	Days of Week	
1	Start	10:00	Sun,Sat	<input type="checkbox"/>
2	Stop	22:30	Sun,Sat	<input type="checkbox"/>
3	Start	06:00	Mon,Tue,Wed,Thu,Fri	<input type="checkbox"/>
4	Stop	19:00	Mon,Tue,Wed,Thu,Fri	<input type="checkbox"/>
<input type="button" value="Delete Selected"/> <input type="button" value="Update Start Date"/>				

Figure 9.22. Week Schedule page

- **Program State:** Define the state (Start/Stop) of the program;
- **Hour:** Time of the day the program will be started/stopped at;
- **Days of Week:** Select the days the defined action should take place;
- **Start Date (dd/mm/yyyy):** The start date for the **Week Schedule**.

Once you have defined a new item, click Add. This item will be added as a new row in a Week Schedule table.

- 💡 This feature allows you to turn specific Relays On/Off upon certain date and time or weekday without the need of LAN connection between the computer and the module.
- 💡 To delete an item, select it in **Existing Items table** and click the Delete Selected button.
- 💡 To set a new start date, click **Update Start Date button**.
- 💡 The module has back-up supply source for the RTC in order to keep the current date/time for several days during power off.

## 9.17. Logout

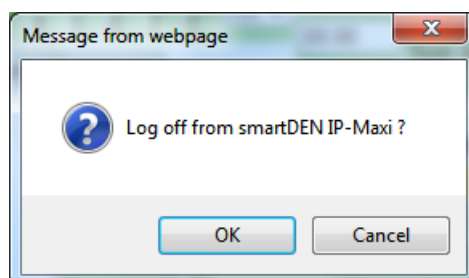
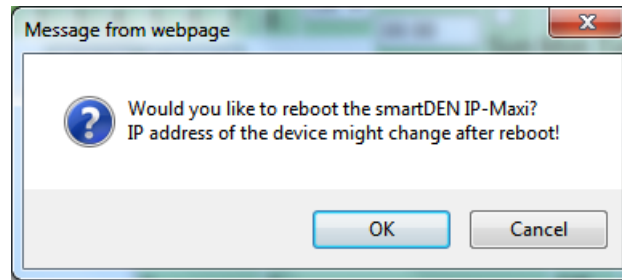


Figure 9.23. Log off

## 9.18. Reboot



**Figure 9.24.** Reboot

## 10. HTTP XML/JSON I/O operation

This operation mode allows custom applications to control the **smartDEN IP-PLC** without using a Web-browser. The custom application acts as a HTTP client, sending HTTP GET requests to the **smartDEN IP-PLC** (Figure 10.1). As a reply the **smartDEN IP-PLC** returns data as either JSON or XML file for parsing and/or processing by the custom application.



**Figure 10.1. smartDEN IP-PLC working as a HTTP server**

To receive the current state of the **smartDEN IP-PLC**, the application requests the page **current\_state.xml** / **current\_state.json**, for example:

[http://192.168.1.100/current\\_state.xml](http://192.168.1.100/current_state.xml)  
[http://192.168.1.100/current\\_state.json](http://192.168.1.100/current_state.json)



In order to use this mode, XML/JSON access should be enabled (see 9.6).

The XML/JSON login process differs depending on the selected Encrypt Password option.

### 10.1. XML access

#### 10.1.1. XML login (encrypted password)

This mode can be used to prevent unauthorized access to **smartDEN IP-PLC**. When selected, a two-step login sequence is performed as a protection against unauthorized access.

The first time the custom application requests the page **current\_state.xml**, a random login key is issued in the reply. Next the custom application uses this key to encrypt the password. The encrypted password is sent as a parameter with the next request to the page **current\_state.xml**.

Bellow is an example of login process:

##### Step 1:

##### Request

[http://192.168.1.100/current\\_state.xml](http://192.168.1.100/current_state.xml)

##### Reply (login required):

```
<CurrentState>
  <LoginKey>65156</LoginKey>
</CurrentState>
```

**Step 2:****Request (password is sent as a parameter)**

[http://192.168.1.100/current\\_state.xml?pw=28237099263eabfd88626124a822c64c](http://192.168.1.100/current_state.xml?pw=28237099263eabfd88626124a822c64c)

**Reply (password is O'K, login accepted):**

```
<CurrentState>
  <DigitalInput1>
    <Name>DIn1</Name>
    <Value>1</Value>
    <Count>12</Count>
  </DigitalInput1>
  ...
</CurrentState>
```



Password encryption algorithm to be implemented in custom application is available upon request.

**10.1.2. XML login (non-encrypted password)**

In this mode the password is passed as non-encrypted parameter with the request:

[http://192.168.1.100/current\\_state.xml?pw=admin](http://192.168.1.100/current_state.xml?pw=admin)

Getting the <LoginKey> in the answer in this mode means that the provided password is wrong or the login session has been expired.



If there is no data traffic between the custom application and the **smartDEN IP-PLC** for time, specified by Session Timeout parameter, the session "times out" and a new login is required.

**10.1.3. Retrieving current state in XML format**

After successful login the custom application can obtain the **smartDEN IP-PLC** current state by request to the **current\_state.xml** page. The reply contains a page in XML format (Figure 10.2).

```

-<CurrentState>
-<DigitalInput1>
  <Name>DIN1</Name>
  <Value>0</Value>
  <Count>0</Count>
</DigitalInput1>
+<DigitalInput2></DigitalInput2>
+<DigitalInput3></DigitalInput3>
+<DigitalInput4></DigitalInput4>
+<DigitalInput5></DigitalInput5>
+<DigitalInput6></DigitalInput6>
+<DigitalInput7></DigitalInput7>
+<DigitalInput8></DigitalInput8>
-<AnalogInput1>
  <Name>AIN1</Name>
  <Value>0</Value>
  <Measure>0.0 Unit</Measure>
</AnalogInput1>
+<AnalogInput2></AnalogInput2>
+<AnalogInput3></AnalogInput3>
+<AnalogInput4></AnalogInput4>
-<AnalogInput5>
  <Name>TIN1</Name>
  <Value>838</Value>
  <Measure>...</Measure>
</AnalogInput5>
+<AnalogInput6></AnalogInput6>
+<AnalogInput7></AnalogInput7>
+<AnalogInput8></AnalogInput8>
-<Relay1>
  <Name>REL1</Name>
  <Value>0</Value>
</Relay1>
+<Relay2></Relay2>
+<Relay3></Relay3>
+<Relay4></Relay4>
+<Relay5></Relay5>
+<Relay6></Relay6>
+<Relay7></Relay7>
+<Relay8></Relay8>
-<AnalogOutput1>
  <Name>AOUT1</Name>
  <Value>0</Value>
</AnalogOutput1>
+<AnalogOutput2></AnalogOutput2>
-<Device>
  <Name>SMARTDEN_IP-PLC</Name>
  <MAC>E8:EA:DA:00:00:00</MAC>
  <sysUpTime>0 days, 0 hours, 1 mins</sysUpTime>
  <ProgramState>3</ProgramState>
</Device>
</CurrentState>

```

Figure 10.2. XML file with current I/O values

## 10.2. JSON access

### 10.2.1. JSON login (encrypted password)

The encrypted login sequence is similar to those for the XML access:

Step 1:

**Request**

[http://192.168.1.100/current\\_state.json](http://192.168.1.100/current_state.json)

**Reply (login required):**

```
{
```



```
"CurrentState": { "LoginKey": "65156" }  
}
```

**Step 2:****Request (password is sent as a parameter)**[http://192.168.1.100/current\\_state.json?pw=28237099263eabfd88626124a822c64c](http://192.168.1.100/current_state.json?pw=28237099263eabfd88626124a822c64c)**Reply (password is O'K, login accepted):**

```
{  
  "CurrentState": {  
    "DigitalInput": [  
      {"Name": "DIn1", "Value": "1"},  
      ...  
    ]  
  }  
}
```



Password encryption algorithm to be implemented in custom application is available upon request.

**10.2.2. JSON login (non-encrypted password)**

The password should be passed as non-encrypted parameter with the request:

[http://192.168.1.100/current\\_state.json?pw=admin](http://192.168.1.100/current_state.json?pw=admin)

Getting the "LoginKey" in the answer means only that the provided password is wrong or the login session has been expired.



If there is no data traffic between the custom application and the **smartDEN IP-PLC** for time, specified by Session Timeout parameter, the session "times out" and a new login is required.

### 10.2.3. Retrieving current state in JSON format

When logged, the custom application can get current measurements requesting the **current\_state.json** page. The reply contains a page in JSON format (Figure 10.3).

```
{
  "CurrentState": {
    "DigitalInput": [
      {"Name": "DIN1", "Value": "0", "Count": "0"},
      {"Name": "DIN2", "Value": "0", "Count": "0"},
      {"Name": "DIN3", "Value": "0", "Count": "0"},
      {"Name": "DIN4", "Value": "0", "Count": "0"},
      {"Name": "DIN5", "Value": "0", "Count": "0"},
      {"Name": "DIN6", "Value": "0", "Count": "0"},
      {"Name": "DIN7", "Value": "0", "Count": "0"},
      {"Name": "DIN8", "Value": "0", "Count": "0"}
    ],
    "AnalogInput": [
      {"Name": "AIN1", "Value": "0", "Measure": "0.0 Unit"},
      {"Name": "AIN2", "Value": "0", "Measure": "0.0 Unit"},
      {"Name": "AIN3", "Value": "0", "Measure": "0.0 Unit"},
      {"Name": "AIN4", "Value": "0", "Measure": "0.0 Unit"},
      {"Name": "TIN1", "Value": "838", "Measure": "---"},
      {"Name": "TIN2", "Value": "838", "Measure": "---"},
      {"Name": "TIN3", "Value": "838", "Measure": "---"},
      {"Name": "TIN4", "Value": "838", "Measure": "---"}
    ],
    "Relay": [
      {"Name": "REL1", "Value": "0"},
      {"Name": "REL2", "Value": "0"},
      {"Name": "REL3", "Value": "0"},
      {"Name": "REL4", "Value": "0"},
      {"Name": "REL5", "Value": "0"},
      {"Name": "REL6", "Value": "0"},
      {"Name": "REL7", "Value": "0"},
      {"Name": "REL8", "Value": "0"}
    ],
    "AnalogOutput": [
      {"Name": "AOUT1", "Value": "0"},
      {"Name": "AOUT2", "Value": "0"}
    ],
    "Device": {
      "Name": "SMARTDEN_IP-PLC",
      "MAC": "E8:EA:DA:00:00:00",
      "sysUpTime": "0 days, 0 hours, 3 mins",
      "ProgramState": "3"
    }
  }
}
```

Figure 10.3. JSON file with current I/O values

## 10.3. Multiple XML/JSON access

With **Multiple Access** option selected (see 9.6) the password should be passed as a non-encrypted parameter with each request:

[http://192.168.1.100/current\\_state.xml?pw=admin](http://192.168.1.100/current_state.xml?pw=admin)  
[http://192.168.1.100/current\\_state.json?pw=admin](http://192.168.1.100/current_state.json?pw=admin)



**Multiple Access** is not allowed when **Encrypt Password** option is enabled.

## 10.4. Parameters

After a login the custom application can also control the **smartDEN IP-PLC** by sending parameters (name/value pairs) with the HTTP GET request.

Valid parameters and values are shown in Table 10.1.

**Table 10.1.** HTTP GET i/o parameters

Name	Value	Description
Relayi (i=1..8)	0	Switch the Relay Off
	1	Switch the Relay On
	2	Toggle the Relay state
SetAll	0.. 65535	Switch Relays On/Off
Counti (i=1..8)	0.. 4294967295	Set Counter value
Pulsei (i=1..8)	0.. 65535	Generate pulse
AnalogOutputi (i=1..2)	0..1023	Set Analog Output units
pw	password	Required at login



Multiple name/value pairs separated by the ampersand ('&') can be passed with a single request, for example:

[http://192.168.1.100/current\\_state.xml?pw=admin&Relay1=1&Pulse5=20&AnalogOutput1=10](http://192.168.1.100/current_state.xml?pw=admin&Relay1=1&Pulse5=20&AnalogOutput1=10)



The overall length of name/value pairs is limited to 100. If this limit is exceeded, the request fails with the "**414 Request-URI Too Long: Buffer overflow**" status code.

## 11. HTTP XML/JSON program operation

In addition to web pages, the custom program can be monitored / controlled by an external application using XML/JSON HTTP requests to **program.xml** / **program.json** pages.



In order to use this mode, XML/JSON access should be enabled (see [9.6](#)).

The following parameters (name/value pairs) can be sent along the HTTP requests:

**Table 11.1.** HTTP GET program parameters

Name	Value	Description
Exec	0	Stop the program execution
	1	Start the program execution (custom variables will be cleared at start-up)
	2	Start the program execution (custom variables will not be cleared at start-up)
VAR_I <sub>i</sub> (i=1..8)	Integer value	Assign a new value to <a href="#">VAR_I<sub>i</sub></a> (i=1..8)
VAR_F <sub>i</sub> (i=1..8)	Float value	Assign a new value to <a href="#">VAR_F<sub>i</sub></a> (i=1..8)
pw	password	Required at login

Examples: Set a new value to [VAR\\_I1](#) and start the program:

[http://192.168.1.100/program.xml?pw=admin&VAR\\_I1=100&Exec=2](http://192.168.1.100/program.xml?pw=admin&VAR_I1=100&Exec=2)

or

[http://192.168.1.100/program.json?pw=admin&VAR\\_I1=100&Exec=2](http://192.168.1.100/program.json?pw=admin&VAR_I1=100&Exec=2)

The replies of requests to **program.xml.xml** / **program.json** contain the current values of the variables and the state of the program.

The **ProgramState** values in the reply have the following meaning:

- “0” - the program is not defined;
- “1” - the program is checked, uploaded to the device, and ready to be started;
- “2” - the program is running;
- “3” - the program is stopped by the user or terminated;
- “4” - the program is stopped due to internal execution error;
- “5” – the program is stopped due to source code CRC error;
- “6” – the program is stopped due to machine code CRC error.

XML reply:

```
<Program>
  <ProgramState>2</ProgramState>
  <Variables>
    <DIN1>0</DIN1>
    <DIN2>0</DIN2>
    ...
    <DIN8>0</DIN8>
    <DINS>32</DINS>
    <OUT1>1</OUT1>
    <OUT2>0</OUT2>
    ...
    <OUT8>0</OUT8>
    <OUTS>73</OUTS>
    <AIN1>0.0</AIN1>
    <AIN2>430.0</AIN2>
    ...
    <AIN8>20.7</AIN8>
    <CNT1>12</CNT1>
    <CNT2>0</CNT2>
    ...
    <CNT8>18</CNT8>
    <AOUT1>110</AOUT1>
    <AOUT2>450</AOUT2>
    <DATE>20/01/2018</DATE>
    <TIME>14:59</TIME>
    <DAYOFWEEK>7</DAYOFWEEK>
    <VAR_I1>100</VAR_I1>
    ...
    <VAR_I8>3</VAR_I8>
    <VAR_F1>1.8</VAR_F1>
    ...
    <VAR_F8>0.0</VAR_F8>
  </Variables>
  <Device>
    <Name>SMARTDEN_IP-PLC</Name>
    <MAC>E8:EA:DA:00:00:20</MAC>
    <sysUpTime>0 days, 0 hours, 6 mins</sysUpTime>
  </Device>
</Program>
```

**Figure 11.1.** XML file with program state values

JSON reply:

```
{
  "Program": {
    "ProgramState": "2",
    "Variables": {
      "DIN1": "0",
      "DIN2": "0",
      ...
      "DIN8": "0",
      "DINS": "0",
      "OUT1": "0",
      "OUT2": "0",
      ...
      "OUT8": "0",
      "OUTS": "0",
      "AIN1": "0.0",
      "AIN2": "0.0",
      ...
      "AIN8": "20.7",
      "CNT1": "12",
      "CNT2": "0",
      ...
      "CNT8": "18",
      "AOUT1": "0",
      "AOUT2": "0",
      "DATE": "20/01/2018",
      "TIME": "15:13",
      "DAYOFWEEK": "7",
      "VAR_I1": "100",
      ...
      "VAR_I8": "3",
      "VAR_F1": "1.8",
      ...
      "VAR_F8": "0.0"
    },
    "Device": {
      "Name": "SMARTDEN_IP-PLC",
      "MAC": "E8:EA:DA:00:00:20",
      "sysUpTime": "0 days, 0 hours, 19 mins"
    }
  }
}
```

**Figure 11.2.** JSON file with program state values

The **ProgramState** value is also included in the replies to **current\_state.xml** and **current\_state.json** pages:

XML reply:

```
▼<CurrentState>
  ...
  ▼<Device>
    <Name>SMARTDEN_IP-PLC</Name>
    <MAC>E8:EA:DA:00:00:00</MAC>
    <sysUpTime>0 days, 0 hours, 1 mins</sysUpTime>
    <ProgramState>3</ProgramState>
  </Device>
</CurrentState>
```

**Figure 11.3.** Program state in the XML I/O reply

JSON reply:

```
{
  "CurrentState": {
    ...
    "Device": {
      "Name": "SMARTDEN_IP-PLC",
      "MAC": "E8:EA:DA:00:00:00",
      "sysUpTime": "0 days, 0 hours, 0 mins",
      "ProgramState": "2"
    }
  }
}
```

**Figure 11.4.** Program state in the JSON I/O reply



## 12. Security considerations

The **smartDEN IP-PLC** runs a special firmware and do not use a general-purpose operating system. There are no extraneous IP services found on general-purpose operating systems (e.g. Telnet, FTP, Finger, etc.) that can be particularly vulnerable.

### Web-browser access

A challenge-response authentication is used in login process. When the password is entered, it is transmitted across the network in encrypted form, so eavesdropping on the data transmission will not reveal the password. Subsequent transmissions of the password to "login" onto the device are encrypted and "safe". The only case when the password is transmitted across the network "in the open", is when it is being changed and submitted in General Setting form. Therefore, you must set passwords in a secure environment where you can make sure that no one is "eavesdropping".

### XML/JSON operation

A challenge-response authentication can be used in login process. The password can be transmitted by custom application across the network in encrypted form.



Web and XML/JSON access can be restricted by IP Address (range of IP Addresses) or by MAC Address.

### 13. Appendix 1. Mechanical drawing

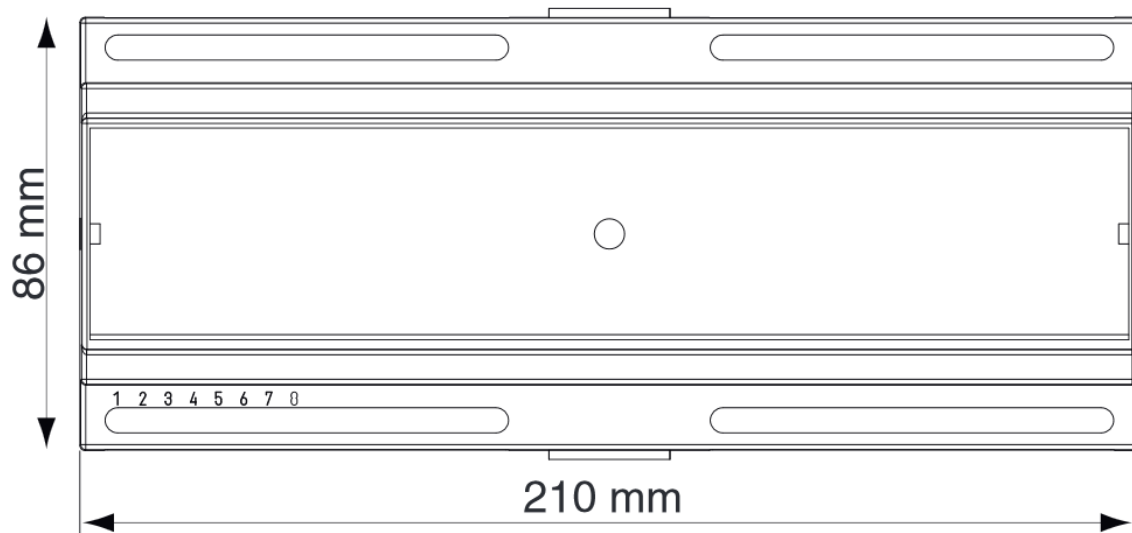


Figure 13.1. Device dimensions

## 14. Appendix 2. DAEL language grammar

Figure 14.1 shows the **DAEL** grammar with non-terminal symbols styled in *italic* and terminal symbols in **bold**.

```

<Program> → <StatementList> EOF
<StatementList> → <Statement> <StatementList> | <null>
<Statement> → VARIABLE = <Expression> ; |
               while ( <Expression> ) <Statement> |
               if ( <Expression> ) <Statement> |
               if ( <Expression> ) <Statement> else <Statement> |
               delay ( INTEGER ) ; |
               { <StatementList> } |
               ;
<Expression> → INTEGER | FLOAT | DATE | TIME | VARIABLE |
               <Expression> + <Expression> |
               <Expression> - <Expression> |
               <Expression> * <Expression> |
               <Expression> / <Expression> |
               - <Expression> |
               <Expression> < <Expression> |
               <Expression> <= <Expression> |
               <Expression> > <Expression> |
               <Expression> >= <Expression> |
               <Expression> == <Expression> |
               <Expression> != <Expression> |
               <Expression> && <Expression> |
               <Expression> || <Expression> |
               ( <Expression> )

```

Figure 14.1. **DAEL** language grammar

## 15. Appendix 3. DAEL operators precedence

Table 15.1. lists **DAEL** operators in order of precedence (lowest to highest). The associativity shows in what order operators of equal precedence in an expression are executed. The precedence can be changed using parentheses to group sub-expressions in an expression. When parenthetical expressions are nested, they are evaluated from inner to outer.

**Table 15.1. DAEL operators precedence**

Operator	Description	Associativity
	Logical OR	Left to right
&&	Logical AND	Left to right
	Bitwise OR	Left to right
^	Bitwise XOR	Left to right
&	Bitwise AND	Left to right
== !=	Equal, Not equal	Left to right
< <= > >=	Less than, Less than or equal, Greater than, Greater than or equal	Left to right
+ -	Addition, Subtraction	Left to right
* / %	Multiplication, Division, Modulus	Left to right

## 16. Appendix 4. DAEL source code for application examples

### 16.1. Pulse generator

This example generates 10 pulses on Relay 1 in a **while** loop. The code demonstrates modeling of **for** loop by **while** operator using **VAR\_I1** as a loop variable.

```
// Example 16.1.1
// This example generates 10 pulses on Relay 1
REL1 = 0;           // Initial Relay state - OFF
VAR_I1 = 0;         // Initialization of the loop variable
while (VAR_I1 < 20)  // Check the end condition
{
    REL1 = 2;        // Toggle the Relay state
    delay(10);       // Delay of 1 sec
    VAR_I1 = VAR_I1 + 1; // Increment the loop variable
}
```

Note that value 2 is used to toggle the Relay state between ON and OFF.

The relay state can be toggled as well by:

```
REL1 = REL1 ^ 1;    // Toggle the Relay 1 state
```

A slightly modified version of the program uses the modulus operator to evaluate the state of the output:

```
// Example 16.1.2
// This example generates 10 pulses on Relay 1
VAR_I1 = 0;           // Initialization of the loop variable
while (VAR_I1 < 20)  // Check the end condition
{
    REL1 = VAR_I1 % 2; // Switch ON/OFF depending on LSB of VAR_I1
    delay(10);         // Delay of 1 sec
    VAR_I1 = VAR_I1 + 1; // Increment the loop variable
}
```

The above examples use **delay()** to form the pulses duration. During delay the program execution is suspended, and no other statements are executed. If the program should take some processing during the delay, the alternative is to use the **TIMER** variable. For example, the statement **delay(10)** can be replaced by:

```
TIMER = 10;          // Set the timer to 1 second
while (TIMER != 0)    // Wait for the timer to expire
{
    // Some other processing
}
```

### 16.2. Running light

This example switches in a loop the Relays, one by one, from Relay 8 down to Relay 1. With LEDs or lamps attached to the outputs, this will generate a running light effect.

```
// Example 16.2
```

```
// This example switches in a loop the Relays,  
// one by one, from Relay 8 down to Relay 1.  
while (1)                // Loop forever  
{  
    RELS = 0x80;          // Switch Relay 8 ON  
    while (RELS != 0)  
    {  
        delay(10);        // Delay of 1 sec  
        RELS = RELS / 2;   // Shift right RELS  
    }  
}
```

### 16.3. Generating a triangular wave

This example generates a triangular wave on Analog Output 1. The code produces 20 samples with a step of 100 milliseconds in each loop iteration.

```
// Example 16.3  
// This example generates a triangular wave on AOUT1.  
// The code produces 21 samples per period with  
// a step of 100 milliseconds.  
AOUT1 = 0;                // Start value - 0 units  
while (1)                 // Loop forever  
{  
    delay(1);              // Delay 100 ms  
    if (AOUT1 < 1023)  
    {  
        AOUT1 = AOUT1 + 50; // Increase by 50 units  
    }  
    else  
    {  
        AOUT1 = 0;          // Scroll to 0 units  
    }  
}
```

### 16.4. Week/month schedule

Example 16.4.1 demonstrates the implementation of week schedule for switching particular Relays ON or OFF at specific times.

```
// Example 16.4.1  
// Implement a week schedule for switching particular  
// Relays ON or OFF at specific times  
while (1)                // Loop forever  
{  
    if (DAYOFWEEK == 2) {  
        // Monday  
        if (TIME >= "12:00" && TIME < "18:00") {  
            RELS = 0x01;    // Relay is ON, others are OFF  
        } else {  
            RELS = 0x00;    // All OFF  
        }  
    } else if (DAYOFWEEK == 3) {  
        // Tuesday  
        if (TIME >= "10:00" && TIME < "14:00") {  
            RELS = 0x02;    // Relay 2 is ON, others are OFF  
        } else {
```

```

        RELS = 0x00;      // All OFF
    }
    // etc. ...
} else if (DAYOFWEEK == 1) {
    // Sunday
    if (TIME >= "20:00" && TIME < "23:59") {
        RELS = 0x80;      // Relay 8 is ON, others are OFF
    } else {
        RELS = 0x00;      // All OFF
    }
}
}
}

```

Example 16.4.2 implements a month/year schedule using **DATE** variable.

```

// Example 16.4.2
// Implement a month schedule for switching particular
// Relays ON or OFF on specific days
while (1)          // Loop forever
{
    if (DATE >= "01/06/2018" && DATE <= "10/06/2018") {
        // From 1 to 10 June 2018
        RELS = 0x01;      // Relay 1 ON, others OFF
    } else if (DATE >= "11/06/2018" && DATE <= "20/06/2018") {
        // From 11 to 20 June 2018
        RELS = 0x02;      // Relay 2 ON, others OFF
    } else if (DATE >= "21/06/2018" && DATE <= "30/06/2018") {
        // From 21 to 30 June 2018
        RELS = 0x04;      // Relay 3 ON, others OFF
    } else {
        // Otherwise
        RELS = 0x00;      // All OFF
    }
}
}

```

## 16.5. Simple ON/OFF thermostat

This example implements a simple ON/OFF thermostat. The temperature sensor connected to Analog Input 8 measures the temperature in the room. The heater element is controlled by Relay 1. The set point is fixed to 20°C with hysteresis band of 2°C. If the temperature exceeds 30°C, an alarm is activated by Relay 2.

```

// Example 16.5
// =====
//          Thermostat
// Set point (SP)      : 20 degrees C
// Hysteresis (H)      : 2 degrees C
// Alarm threshold     : 30 degrees C
// Process value (PV)  : AIN8
// Heater output       : Relay 1
// Alarm output        : Relay 2
// =====
// Initialization
VAR_F1 = 20.0;        // Set point in VAR_F1
REL2 = 0;             // No alarm at start
// Evaluate the initial heater state
if (AIN8 > VAR_F1) {

```



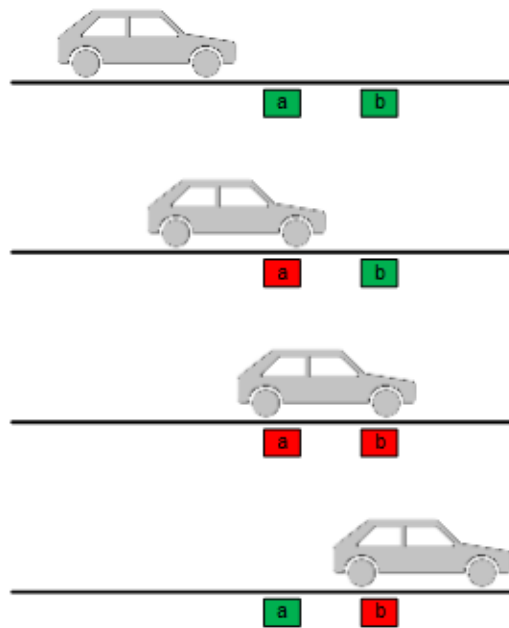
```
REL1 = 0;          // Switch the heater OFF
} else {
    REL1 = 1;       // Switch the heater ON
}
//////////////////////////
//      Control loop      //
//////////////////////////
while(1) {          // Loop forever
    // Check process value
    if (AIN8 > VAR_F1 + 1.0) {
        // PV > SP + H/2
        REL1 = 0;    // Switch the heater OFF
    } else if (AIN8 < VAR_F1 - 1.0) {
        // PV < SP - H/2
        REL1 = 1;    // Switch the heater ON
    }
    // Check for alarm
    if (AIN8 > 30.0) {
        REL2 = 1;    // Switch alarm ON
    } else {
        REL2 = 0;    // Switch alarm OFF
    }
}
```

A more practical version of the thermostat supposes that the set point is determined by the value of another Analog Input.

## 16.6. Car park counting

This example demonstrates a simple car park counting system. The parking lot has only one entrance/exit lane equipped with two sensors placed close together along the path of the vehicle. The system detects the direction in which cars are travelling and maintains a count of cars found in the parking lot. Two signal lamps (“full” and “spaces”) inform the visitors of the availability of spaces.

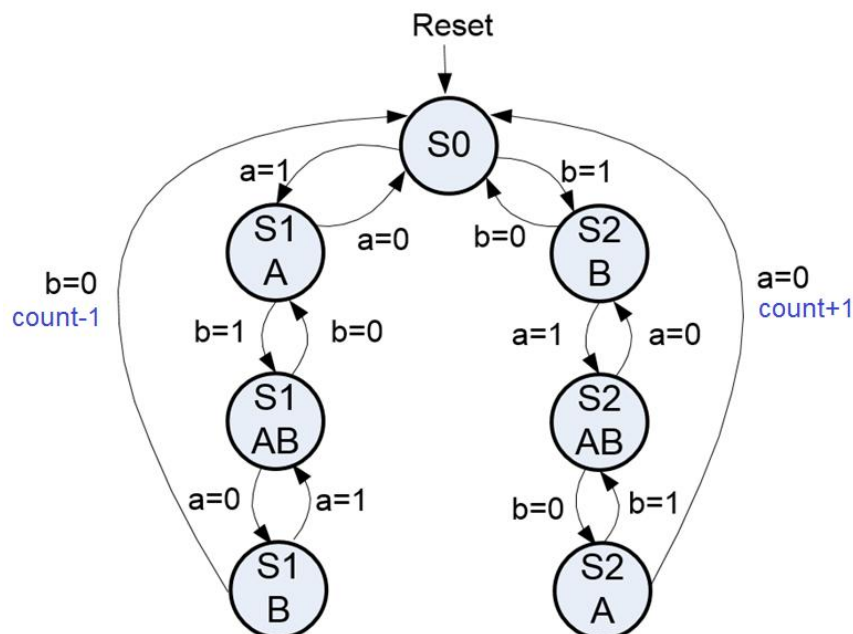
To detect the direction of cars the program considers the successive states of two sensors at the park entrance - **a** and **b** (Figure 16.1). For example the sequence ‘ab’, ‘ab’, ‘ab’, ‘ab’ detects a car entering the car park, and ‘ab’, ‘ab’, ‘ab’, ‘ab’ – a car leaving out.



**Figure 16.1.** Successive states of the sensors in detection of a car

Initially, when the car park is empty the count of cars is set to the total number of parking spaces, and the “spaces” lamp is on. As a car enters the car park the count is decremented, and when a car leaves the count is incremented. When the count equals zero, the “full” lamp is on.

The corresponding state diagram is shown in Figure 16.2. The left subsequence of states (prefixed with S1) detects entering cars, and the right (prefixed with S2) – cars leaving out. Suffix denotes the sensors active in that state.



**Figure 16.2.** State diagram

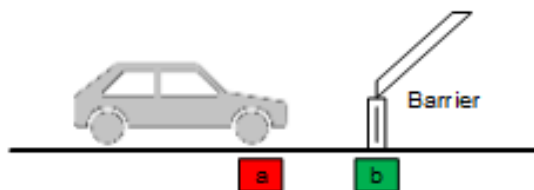
The following code shows an example of a program for a parking lot containing 20 spaces.

```
// Example 16.6
// Car park counting
// Sensor A      : DIN1
// Sensor B      : DIN2
// Lamp "spaces"  : REL1
// Lamp "full"    : REL2
// Current state  : VAR_I1
// Car count      : VAR_I2
// Initial state (S0) in VAR_I1
// Total number of spaces in VAR_I2
// State machine loop
while(1) { // Loop forever
  REL1 = (VAR_I2 != 0); // Set lamp "spaces"
  REL2 = (VAR_I2 == 0); // Set lamp "full"
  if (VAR_I1 == 0) {
    // State S0
    if (DIN1 == 1) {
      VAR_I1 = 1; // Transit to S1A
    } else if (DIN2 == 1) {
      VAR_I1 = 4; // Transit to S2B
    }
  } else if (VAR_I1 == 1) {
    // State S1A
    if (DIN1 == 0) {
      VAR_I1 = 0; // Return to S0
    } else if (DIN2 == 1) {
      VAR_I1 = 2; // Transit to S1AB
    }
  } else if (VAR_I1 == 2) {
    // State S1AB
    if (DIN2 == 0) {
      VAR_I1 = 1; // Return to S1A
    } else if (DIN1 == 0) {
      VAR_I1 = 3; // Transit to S1B
    }
  } else if (VAR_I1 == 3) {
    // State S1B
    if (DIN1 == 1) {
      VAR_I1 = 2; // Return to S1AB
    } else if (DIN2 == 0) {
      // Car entered
      if (VAR_I2 > 0) {
        VAR_I2 = VAR_I2 - 1; // Decrement count
      }
      VAR_I1 = 0; // Transit to S0
    }
  } else if (VAR_I1 == 4) {
    // State S2B
  }
}
```

```
// State S2B
if (DIN2 == 0) {
    VAR_I1 = 0;          // Return to S0
} else if (DIN1 == 1) {
    VAR_I1 = 5;          // Transit to S2AB
}
} else if (VAR_I1 == 5) {
    ///////////////////////////////////////////////////
    // State S2AB
    if (DIN1 == 0) {
        VAR_I1 = 4;      // Return to S2B
    } else if (DIN2 == 0) {
        VAR_I1 = 6;      // Transit to S2A
    }
} else if (VAR_I1 == 6) {
    ///////////////////////////////////////////////////
    // State S2A
    if (DIN2 == 1) {
        VAR_I1 = 5;      // Return to S2AB
    } else if (DIN1 == 0) {
        // Car left out
        if (VAR_I2 < 20) {
            VAR_I2 = VAR_I2 + 1; // Increment count
        }
        VAR_I1 = 0;      // Transit to S0
    }
}
}
```

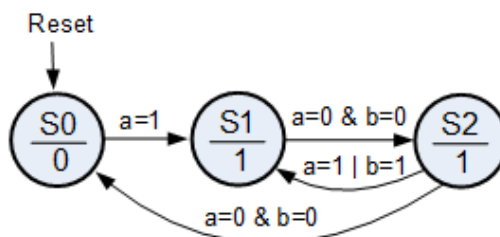
## 16.7. Barrier control

This example demonstrates how the **TIMER** variable can be used in transitions of a state machine. The state machine detects the presence of a car in front of a barrier using sensor **a** (Figure 16.3). Sensor **b** checks for a car under the barrier. An output named “Barrier” is used to control the barrier (1 – open, 0 – close).



**Figure 16.3.** Barrier control

The corresponding state diagram is shown in Figure 16.4. The output value associated with the state is depicted below the state's name.



**Figure 16.4.** State diagram

In the initial state S0 the barrier is closed. Upon detecting a car ( $a=1$ ) the barrier is opened and the state machine transits to S1. While in S1, the presence of a car ( $a=1$  or  $b=1$ ) is checked. If neither of sensors is active, the state machine moves to S2, where a delay (using **TIMER** variable) is introduced before closing the barrier. When the timer expires (**TIMER**=0), the state machine transits to S0 or S1 depending on the presence of a new car.

```
// Example 16.7
// Barrier control
// Sensor A : DIN1
// Sensor B : DIN2
// Output Barrier : REL1
// Current state : VAR_I1
// Exclusive use of REL1
VAR_I1 = 0; // Initial state (S0) in VAR_I1
// State machine loop
while(1) { // Loop forever
    if (VAR_I1 == 0) {
        // State S0
        REL1 = 0; // Barrier is closed
        if (DIN1 == 1) { // Check sensor A
            VAR_I1 = 1; // Transit to S1
        }
    } else if (VAR_I1 == 1) {
        // State S1
        REL1 = 1; // Barrier is opened
        if (DIN1 == 0 && DIN2 == 0) { // Check sensors A and B
            TIMER = 100; // Start timer (10 seconds period)
            VAR_I1 = 2; // Transit to S2
        }
    } else if (VAR_I1 == 2) {
        // State S2
        REL1 = 1; // Barrier is opened
        if (TIMER == 0) { // Is the timer expired ?
            if (DIN1 == 1 || DIN2 == 1) { // Another car ?
                VAR_I1 = 1; // Return to S1
            } else {
                VAR_I1 = 0; // Transit to S0
            }
        }
    }
}
```

## 16.8. Traffic lights

smartDEN IP-PLC can be used to control simple traffic lights system (red, yellow, green) with single cars flow and a zebra crossing (red, green). The system is in working mode between 06:00 AM and 22:00 PM. Out of this time interval, the yellow

color is blinking. Currently no feedback is provided but it is possible to be added simply current sensors in order to provide information if some lamp is defected.

```
// Example 16.8
////////////////////////////////////
//          Traffic lights          //
// Cars red       : REL1            //
// Cars yellow    : REL2            //
// Cars green     : REL3            //
// Pedestrian red  : REL4            //
// Pedestrian green: REL5            //
////////////////////////////////////

lock(REL1); //cars red
lock(REL2); //cars yellow
lock(REL3); //cars green
lock(REL4); //pedestrian red
lock(REL5); //pedestrian green
RELS = RELS & 0xE0; //all lights are off
VAR_I1 = 0; //0=time between 06:00 and 22:00, 1=time between 22:00 and
//06:00
while(1)
{
  if (TIME >= "22:00" && TIME < "06:00")
  {
    if(VAR_I1 == 0)
    {
      RELS = RELS & 0xE0; //all off
    }
    REL2 = 1; //REL2 is on, others off
    delay(5); //delay 500 millisecond
    REL2 = 0; //all lights off
    delay(5); //delay 500 millisecond
    VAR_I1 = 1;
  } else {
    if(VAR_I1 == 1)
    {
      RELS = RELS & 0xE0; //all off
      REL2 = 1;
      REL4 = 1; //REL2 and REL4 are on, others off
      delay(50); //delay 5 seconds
      REL2 = 0;
      REL4 = 0;
    }
    REL1 = 1;
    REL4 = 1; //REL1 and REL4 are on, others off
    delay(20); //delay 2 seconds
    REL2 = 1; //REL1,REL2 and REL4 are on,others off
    delay(20); //delay 2 seconds
    REL1 = 0;
    REL2 = 0;
    REL3 = 1; //REL3 and REL4 are on, others off
    delay(150); //delay 15 seconds
    REL3 = 0;
    REL2 = 1; //REL2 and REL4 are on, others off
    delay(30); //delay 3 seconds
    REL2 = 0;
    REL1 = 1; //REL1 and REL4 are on, others off
    delay(20); //delay 2 seconds
  }
}
```

```
REL4 = 0;
REL5 = 1; //REL1 and REL5 are on, others off
delay(100); //delay 10 seconds
REL5 = 0;
VAR_I1 = 0;
}
}
```

## 16.9. PID controller

The below example demonstrates how the **smartDEN IP-PLC** module can be used as PID controller for regulating temperature by given set point. It is used one temperature sensor input (**AIN5**) and for output is used **AOUT1**. For example for boiler heating it can be used some 0-10V heating controller. The proportional ( $K_p=100$ ), Integral ( $K_i=50$ ) and Differential ( $K_d=20$ ) coefficients are adjusted on paper but for real system most probably they will be different and that is one of the most important adjustment of the system.

```
// Example 16.9
////////////////////////////////////
//          PID controller          //
// Input value      : AIN5          //
// Output value     : AOUT1         //
////////////////////////////////////

lock(AOUT1); //Output = heater control, Setpoint = 34 degC +/-1degC
//Kp = 100, Ki = 50, Kd = 20
VAR_F1 = AIN5; //Input
VAR_F2 = 0; //Error
VAR_F3 = 0; //Derivative Error
VAR_F4 = 0; //Error Sum
VAR_F5 = 0; //Last Error
VAR_F6 = 0; //Output
AOUT1 = VAR_F6;
VAR_F2 = 34.0 - AIN5; //Error = Setpoint - Input
VAR_F5 = VAR_F2; //Last Error = Error
delay(10); //1 second delay
while(1){
  VAR_F2 = 34.0 - AIN5; //Error = Setpoint - Input
  if(AOUT1 > 0 && AOUT1 < 1023) {
    VAR_F4 = VAR_F4 + 1*VAR_F2; //Error Sum = TimeChange * Error
  }
  VAR_F3 = (VAR_F2 - VAR_F5)/1; //(Error - Last Error) / TimeChange
  //Output = Kp*Error+Ki*ErrorSum+Kd*Derivative Error
  VAR_F6 = 100*VAR_F2 + 50*VAR_F4 + 20*VAR_F3;
  if(VAR_F6 < 0)
    VAR_F6 = 0;
  else if(VAR_F6 > 1023)
    VAR_F6 = 1023;
  AOUT1= VAR_F6; //Setting the output
  VAR_F5 = VAR_F2; //Last Error = Error
  delay(10); //1 second delay
}
```