SoftPLC IDE

User Guide

RC•Ware®
SoftPLC

Contents

1 Introduction

1.1 Prerequisites and targets

Prerequisites:

There is technical description and shop drawings of the project available (or at least I/O table with I/O peripherials assignment to the inputs and outputs of the I/O modules. The system topology is defined, inclusive management level (RcWare Vision).

For functional tests, the I/O modules are mounted in the panels, connected to peripherials and to the I/O bus, addressed, and commissioned.

Targets:

There is application software for each process station designed, parametrized, compiled, loaded into the stations, and tested in the real I/O bus, inclusive data exchange between the process stations. There is a datapoint list generated for easy datapoint import into the RcWare Vision. In case the touchscreen process stations are employed, there are panels generated and downloaded to the station so that the station can be controlled via the touchscreen.

1.2 System principles

The SoftPLC application consists at least of two basic components:

- runtime

- application program.

The runtime is an executable installed on the process station machine. It is an interpreter of the application program which has to be copied to the process station and the path to the application program has to be defined in the runtime parameters (see Runtime Configuration).

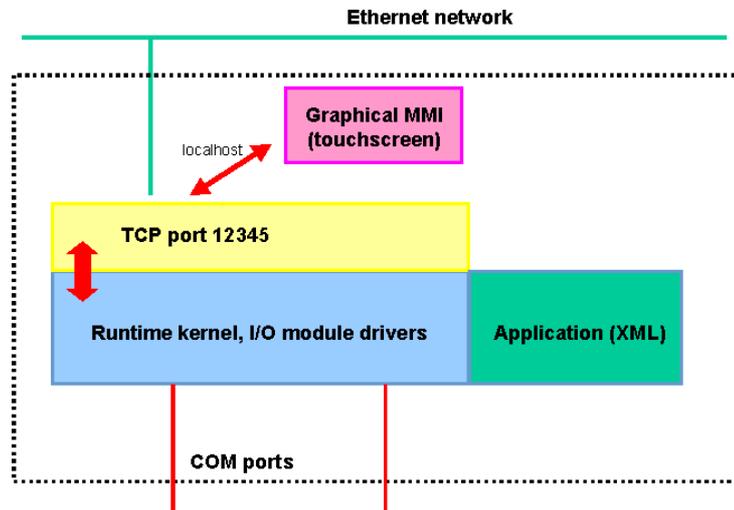**Process station with a runtime**
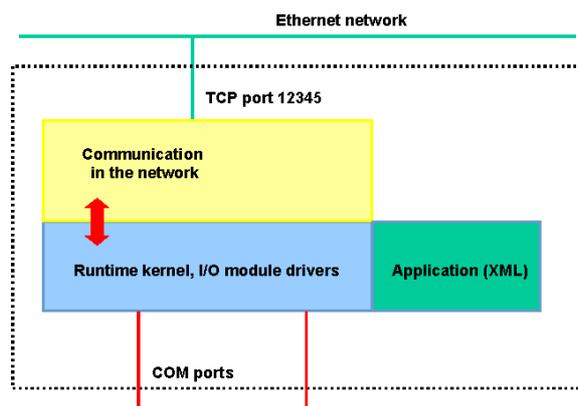
At the COM port(s), there are the I/O modules connected. When the runtime starts, it checks the licence first (see Licensing), then reads the application files (further referred to as a *project* or *application*), checks their integrity and starts to evaluate them.

For data sharing to other programs and modules, there is a communication layer. It accepts connections at the TCP port 12345 by default; it can be changed in the runtime configuration. To this port connect e.g. the touchscreen application, and/or OPC server for integration into RcWare Vision. The reason for this is that the TCP connection uses a specific protocol, is stable, and can be platform independent (unlike e.g. OPC).

The Touchscreen application connects through the TCP port 12345:

**Process station with a runtime and touchscreen (e.g. IPCT.1)**

Thanks to this network topology, it is also possible to run the Touchscreen application on a different machine, so that the process station is close to the technology and the MMI resides e.g. at the housekeeper's office.

Process station with a runtime, separate touchscreen application
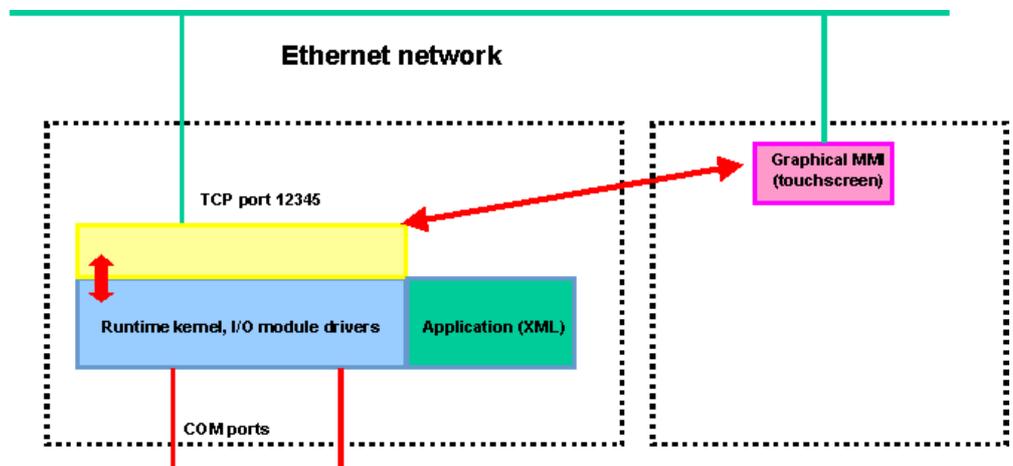
Similarly, to connect more process stations to one OPC server, the TCP communication is used:
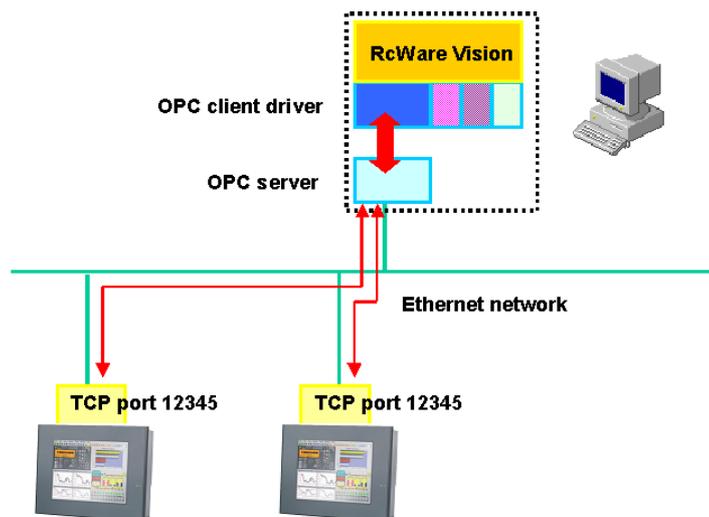
Details see below (Integration into RcWare Vision).

2   IDE

The IDE (Integrated Development Environment) is a comprehensive tool for design, parametring, commissioning and testing of the SoftPLC applications.

2.1 System requirements

The IDE requires a PC or notebook with:
- Pentium 1 GHz and above
- 256 MB RAM minimum, 512 MB for regular work, 1 GB for the largest applications
- at least 30MB HDD free space (plus projects)
- Ethernet connection
- serial port or USB to serial adapter for testing of the I/O modules
- XGA color display (1024 x 768 and above)
- Windows 2000 SP2 or Windows XP
- Microsoft.NET V2.0 (available at **http://msdn.microsoft.com/netframework** free of charge, search for „.NET Framework Version 2.0 Redistributable Package").

2.2 Installation

The complete installation package is distributed either on a CD, or as a zipped file. When unzipped, a directory structure is created which contains

- installers (.msi) for different components (runtime, IDE, touchscreen, OPC server...)
- licensing tool (ESG.SoftPLC.Licenses.HWInfo.exe)
- small program for easy launching of the installers (SoftPLC.InstallationHub.exe).

1. Run **SoftPLC.InstallationHub.exe**.
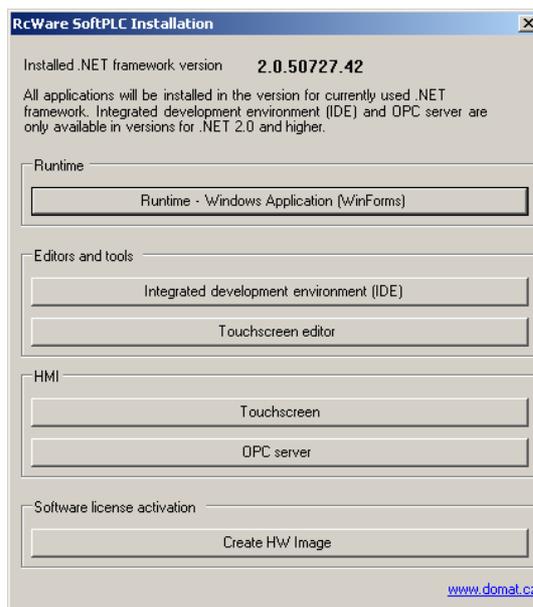
2. In the upper part of the window, the installed .NET Framework version is displayed. Make sure that it is 2.0 and above.

3. Install „**Integrated development Environment (IDE)**". In the course of installation, leave the program defaults.

4. If you plan to use the **Touchscreen editor** you can install it now. Again, leave the program defaults.

Demo version, full version

5. The IDE is now installed. It will run as a **demo** version, with the limitation that the communication with the modules *stops after 10 minutes*. All the other functions are same as in **full** version. If you want to use the full version, you have to licence the software (see Licencing).

6. Close the Installation Hub.

2.3 IDE windows and panels

2.3.1 Program window

Run **Start – Programs – RcWare – SoftPLC – SoftPLC**. The splash screen displays the SoftPLC logo and progress of loading the components. After the startup, the working environment opens:
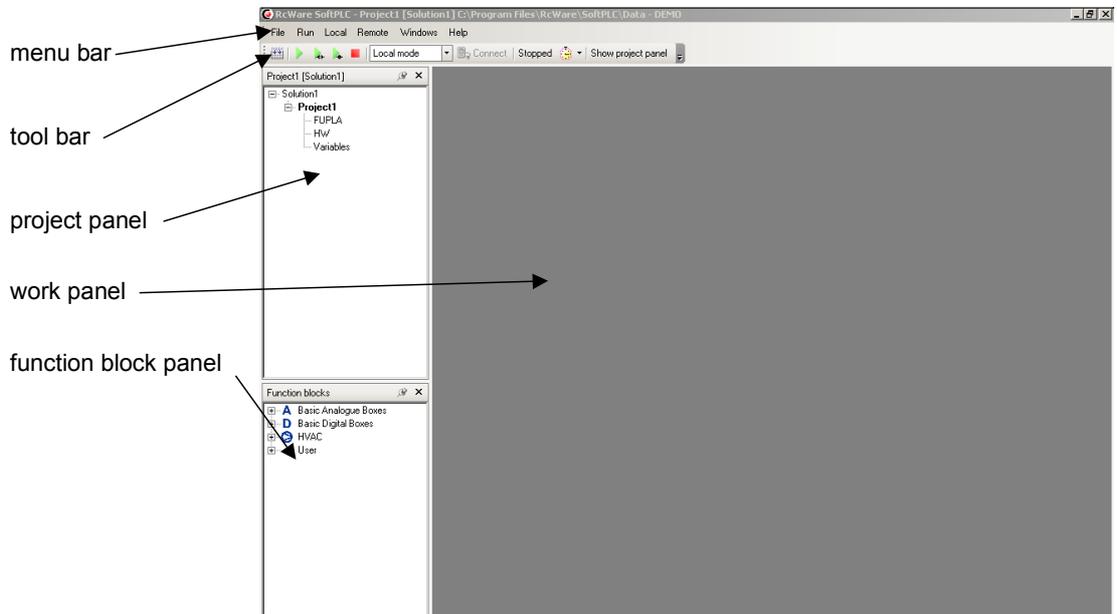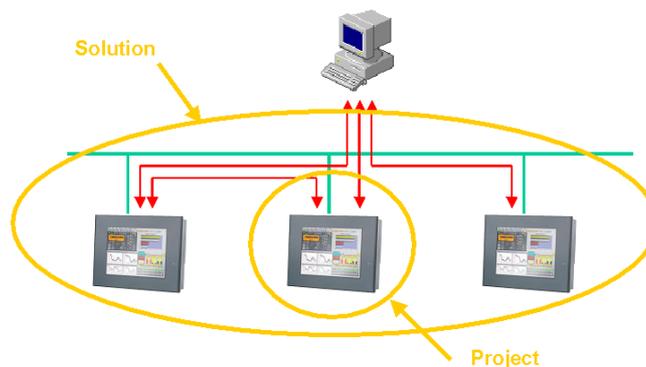

It consists of following parts:

- menu bar
– toolbar
– project panel
– function blocks panel
– work panel
– compiler output panel (not visible now).

2.3.2 Project panel

Project, solution   In the project panel, solutions and projects are defined and selected for configuration. What is a **project** and a **solution**?

A **project** refers to a program for a single process station. A **solution** contains one or more projects which are in the same network and may share data and/or be linked to one (or more) common management station(s). Each solution contains at least one project. Even in case of a single autonomous process station, a solution must be defined (which contains solely this one process station).

SoftPLC V1 supports one project per solution only. If your network contains more PLCs, a separate Solution must be created for each Project. It is possible to share data between Projects in the network – use a channel communicating over Native SoftPLC Link to read data from another process station.

Each project contains three items:

FUPLA

Short for FUnction PLAn. A graphical representation of the program's functionality, consisting of variables and functional blocks, which read the variables, process them, and write the results to other variables.

HW

Hardware configuration of the process station and the I/O modules, namely the I/O buses (serial lines or network channels to communicate with the I/O modules), their addressing, configuration, types and addresses of the I/O modules, their settings, and other parameters.

This is also where the inter-station communication is defined.

Variables

List of all variables available in the SoftPLC application. Values of the variables can be set and monitor here. For quick view, there are filtering and sorting functions available.

When selected by a double left click, the HW and Variables tabs are opened in the work panel.

2.3.3   Function block panel
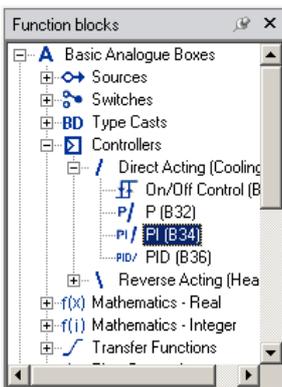
Here is the library of all available SoftPLC function blocks, which may be dragged & dropped into the schema (a graphical sheet with functional blocks, links and variables) and from which the program functionality is composed.

The blocks are sorted in a tree view according to their functionality. The main groups are:

– Basic analogue blocks
– Basic digital blocks
– HVAC blocks.

The description of all blocks and their functionality find in the **Function block documentation** (a .pdf file which installs together with the IDE). The link to the documentation is in the SoftPLC program group and is created automatically during the installation process.

2.3.4   Work panel

In this panel are opened the schema sheets, variables viewer and hardware configurator.

It is possible to have more tabs open at the same time and switch between them by clicking on the tab name.

An **asterisk** at the tab name indicates that there are unsaved changes in the tab. The active tab can be closed by clicking the „X" in the right upper corner.

2.4   Creating a project

2.4.1   New solution

1. Select **File**, **New**, **Solution**.

2. Fill in the solution name (typically commission name, e.g. **BlueStarHotel**) and
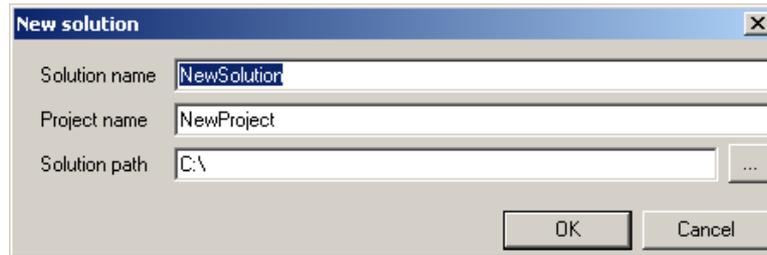
project name (typically panel in which the station resides or functionality of the process station (e.g. **BA**, **Heating**, **BuildingC**.)) The Solution path points to the folder where the project will be stored.

All those data can be changed later.

3. Right click to the project name and choose **Properties**.
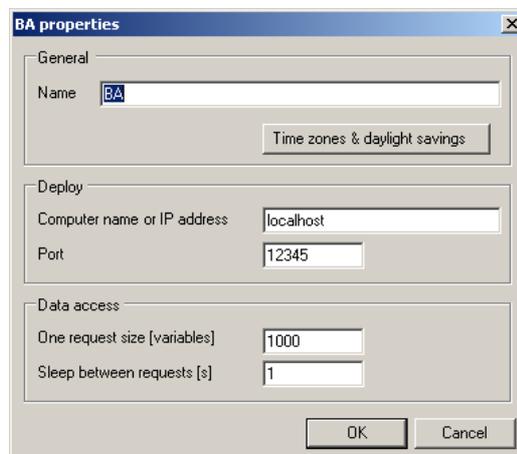
If the IP address of the process station is already known, it can be entered in the *Deploy* frame. Use *Time zones & daylight savings* only if you install the process station in a different area than Europe (UTC + 1h, daylight time offset 1 h). For other areas, daylight savings scheme can be entered manually.

Leave default values in the *Data access* frame. They should only be changed if there are special communication requirements.

2.4.2 Communication channels

1. Double-click on the **HW** tree branch. An empty tab in the work panel opens.

2. Right-click on the work panel and select **Create channel**.

For serial communication with the Domat I/O modules, choose *Serial channel* and *Modbus RTU – Serial* driver. Rename the communication bus to any suitable name (it will be part of the I/O variables names).

Communication channel types and drivers

Available channel types are:

OPC channel: the SoftPLC acts as an OPC client to an OPC server running on the same machine.

**Serial channel**: a serial RS232/RS485 line, a COM port. The most important driver for us now is

Modbus RTU TCP/serial – domat I/O modules on Modbus or any Modbus I/O modules or devices acting as Modbus servers

SoftPLC native channel: used for data exchange between the process stations

TCP channel: for Modbus over TCP etc.

UDP channel: BACnet/IP etc.

3. Right-click on the channel and choose **Edit**, **Channel properties**.
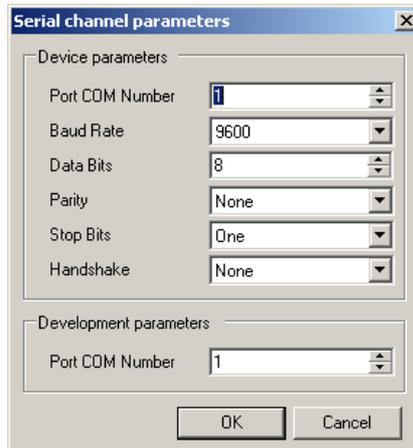
Set the COM port parameters as required. Note that the COM port number may be different for your notebook (e.g. COM1 for testing) and for the process station (e.g. COM2). Therefore it is possible to set development COM port separately. It will be used for IDE tests, while the *Device parameters* COM port specified will be used at the runtime.

Development parameters

The default settings are suitable for use with the M012 converter and the Mxxx domat I/O modules.

2.4.3   Adding modules

1. Now, the I/O modules on the bus will be added. For their types and addresses, consult the shop drawings together with addressing scheme. Right-click on the channel and choose **Add device**.
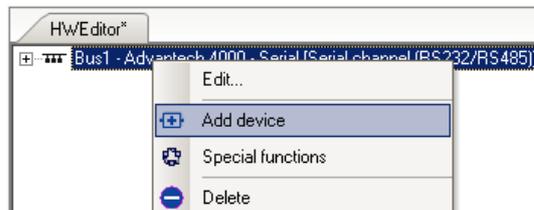
2. From the list, select the first module to be on the bus and click **OK**. The module appears in the list and double-clicking on it opens the dialogue with module settings.

3. Set the module address (decimal). Set the input and output variable names (and transformations if necessary).
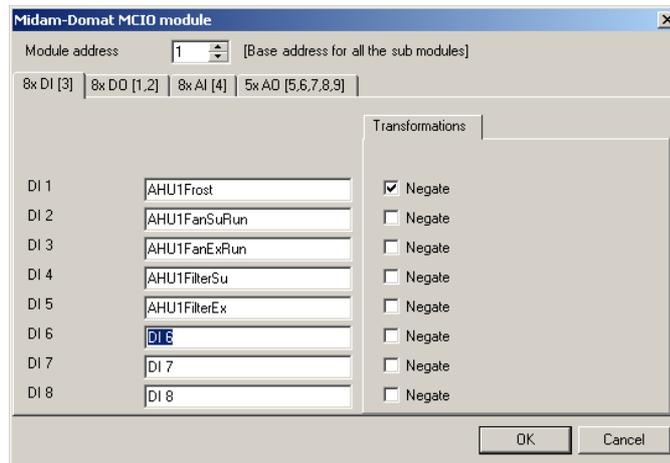
4. Continue with all the modules on the bus. Avoid duplicate addressing.

2.4.4 Variables viewer

The modules are listed in the tree view. In the Project panel, **Variables** item you can see the list of all automatically created variables relating to hardware (I/O).

The first column is used for tagging all and untagging all variables
OPC selected variables are visible in the RcWare OPC server (typically those which are important for RcWare Vision).
Name prefix is part of the variable name and is used for specifying variable nature (HW, HWBlock, SW (system)) or location (FUPLA schema name).
Name: Variable name to which the references in the FUPLA variable ladder refer. This name is also part of the OPC variable name.
Value: Actual value of the variable, changing dynamically.
Unit: Physical unit (°C, %, %rH, kPa etc.)

Setting of units at HW variables

For inputs and outputs (HW variables), the units can be set and changed by clicking on the corresponding line and pressing *Ctrl+E*. A dialogue opens where various parameters can be set.

Type: Data representation type. Available types are:

Boolean: False or True.
Double: Real values, double precision: 64 bit floating point.
Int64: 64 bit signed integer.
String: Reserved for future use.
DateTime: Used for date and time operations.

Detailed description of the data types find in the *Functional Blocks Reference*

Domat Control System **RcWare** 10

Documentation.

Update time: Date and time of last update of the variable. Here, the last communication event can be checked and see if the communication lines are optimized for the I/O module configuration.
Quality: Bad or Good, depending on the update status.

Blocks: Each hardware variable can be blocked against reading, writing, or both. The block follows after writing into the corresponding blocking variable, which is created automatically and named **<VariableName>_blk_mode**. The blocking variables accept following values:
0	No blocks
1	Read block (i.e. the variable is not read)
2	Write block (i.e. the variable is not written to)
3	Read and write block

(Only the last two bits are valid, therefore e.g. 8 = No blocks, 7 = Read and write block etc.)

Why use blocks?

Blocks are used in case that there is the complete I/O bus defined in the IDE, but only part of it is actually installed and in operation. Normally, the driver tries to contact the missing modules periodically, which results in slow communication and timeouts. To prevent this, the missing modules should be blocked.

The whole module (or rather all the HW variables in it) can be blocked by right-clicking on the module icon in the **HW** tab and choosing *Reads and writes block*. All the variables in it are blocked (see the **Variables viewer** tab.)

[001] MCIO

The module can be unblocked again by right-clicking on the module icon in the **HW** tab and choosing *Cancel all blocks.*

Origin: A hint indicating where the variable comes from. Even after it is renamed to a custom name, the Origin text serves as a help text.

2.4.5   Function plan

In the Project tree, under FUPLA, there is a default schema named *schema0*. It is an empty schema. Double-click on *schema0* and the sheet opens in the work panel.
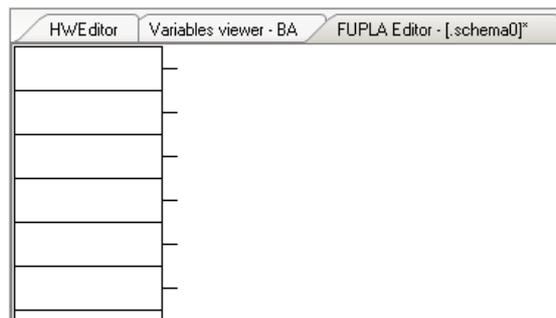
The ladders at the right and left sides provide positions for inserting variables. This is how the HW variables are linked to the function blocks.

Add new schema

There may be more schemes in a project. Add new schema by right-click on FUPLA and choose **Add schema**. You can rename the schemas so that the names describe the functionality of the schema. Note that the schema names are part of the OPC variable names.

It is also possible to compose the schemas in a tree structure (right-click on schema rather than on FUPLA, and **Add schema**). This structure can be used for convenience only, it has no influence on the evaluation of the blocks.

Each schema can incorporate any number of functional blocks.

2.4.6 Graphic functions

Change schema size

The schema can change its size by dragging the boundaries:

Zoom in & out
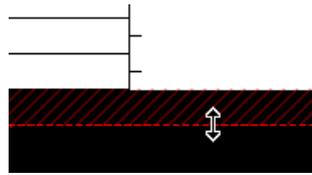
In case the boundaries are not visible, use the zooming tools at the toolbar. Zoom out to that extent that the complete schema is visible together with black background. Then locate the boundaries and change the size of the schema. You also can use Ctrl+mouse wheel to zoom.
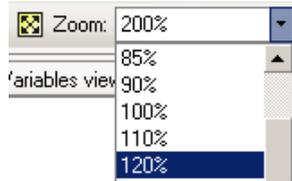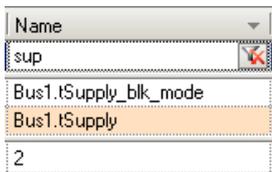

If stretched vertically, new positions appear in the ladder so that more variables can be inserted.

Fit to screen 🔳

At any time you can fit the schema to screen by clicking the „Fit to screen" icon.

2.4.7 Insert a variable into schema

To be able to link variable to any FUPLA function block, e.g. to bring the sensor signal to the PID controller input, it is necessary to insert the hardware variable in the variable ladder.

1. Right-click to a position in the ladder.

2. Choose **Place variable**.

3. Select the variable from the variables list. Use sorting and filtering functions: in the Name filter box, write a substring of the variable's name and press <Enter>.

4. The list is limited to those variables which comply with the filtering criteria.

5. Double-click to the variable name. The dialogue closes and the variable is inserted into the ladder. It can now be referred to by the function blocks.

2.4.8 Software variables

It is also possible to define *software variables*. Those are not part of any functional block nor are connected to the hardware. Example of the software variables are the *system variables*.

Let's have a look at the system variables (which are the only software variables in the project now). Filter the variables according their name prefix – it shall be **SW**.

	OPC	Name prefix	Name	Value	Unit	Type	UpdateTime	Quality	Blocks	Origin
		SW								
		SW	System.RTC	25.07 2006 10:55:32.980		DateTime	25.07 2006 10:55:32..	Good	N/A	SW
		SW	System.RTCDaylight	false		Boolean	25.07 2006 10:55:32..	Good	N/A	SW
		SW	System.RTCUTC	25.07 2006 08:55:32.980		DateTime	25.07 2006 10:55:32..	Good	N/A	SW
		SW	System.StopAllowed	false		Boolean	25.07 2006 10:55:32..	Good	N/A	SW
		SW	System.StopRequest	false		Boolean	25.07 2006 10:55:32..	Good	N/A	SW
		SW	System.WaitForApplicatio...	false		Boolean	25.07 2006 10:55:32..	Good	N/A	SW
			6							

System variables	*System.RTC*: Returns actual system time in the local format.

System.RTC: Returns actual system time in the local format.
System.RTCDaylight: If true, the system finds itself in the daylight savings time period.
System.RTCUTC: Returns actual system time as Universal Time Coordinated.
System.WaitForApplicationStop: If false (default), the system does not wait for *System.StopAllowed* and after request for stop (invoked by „Stop" button at the runtime MMI) shuts down immediately.
If true, the stop request sets the *System.StopRequest* variable to True and the system keeps running until *System.StopAllowed* is True. The *System.StopAllowed* variable must be set by the application.
This functionality makes possible to control the shutdown of the program and complete certain tasks before the application is closed.

Insert a new variable

A new software variable can be inserted in the **Variables viewer** tab:

1. Go to the **Variables viewer** tab

2. Press <Ins>
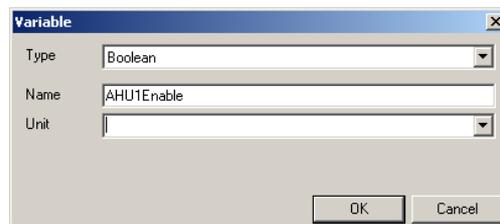
3. Select the variable type, enter name and optionally unit (you can choose from the list or enter another, free definable unit)

4. Click **OK**. The new variable appears in the list.

Why software variables?

The software variables are used especially for transferring signals from one schema to another. In the source schema, the variable is inserted in the output ladder (right), while in the target schema the same variable is inserted in the input ladder (left). This is how the signal gets through the schemas.

There may be several variable instances which read from one source variable, however, only one function block may write to a variable. Now let's see how function blocks look like.

2.4.9   Function blocks

To learn more about function blocks, see the *Functional Blocks Reference Documentation*.

The function blocks share data with the application through inputs, parameters, and outputs. Together, inputs, parameters, and outputs are called terminals.

Some terminals, according to their functionality, may be defined as *inside* or *outside* the function block (or box).

Inside the box: the value is fixed and can be changed only as a parameter (via display or SCADA).

Outside the box: the value must be connected to a variable (or output terminal of another block, which is a variable, too), and changes in the course of application runtime. It cannot be influenced from display or SCADA (unless the source block is in manual mode, see below).

Inputs

The inputs provide the function blocks with values for processing of the blocks' functionality. Binary (boolean) values may be inverted by checking the **NOTed** tickbox.

Each input has several properties which are listed in the table:

Name: A short or symbol which gives an unique variable description within the block. It is a part of variable name and can not be changed.

Outside: If checked, the terminal appears outside the function block and can be connected to another block's output, or variable.

NOTed: If checked, the boolean value is inverted before processing. The inversion is symbolized by a small circle at the terminal.

A block with inverted **enable** input.

Type: Variable type (Boolean, Double64, Integer64, String, DateTime). Can not be changed.

Context block help

Description: Short description of the terminal's functionality in the block logic. For detailed description, go outside the block, click on it once and press **<F1>**. The help file opens at the actual block's description.

Parameters

They are similar to inputs according to their functionality. However, there are some more properties in the table:

OPC: If checked, the parameter appears in the OPC variables list and will be exported to the OPC definition file, a list of variables to be integrated into RcWare Vision / SCADA. Check this if the parameter should be visible and/or controllable at user level (heating curve parameters, night setpoint shifts etc.).

Default value: If the parameter is not *Outside*, this is the value of the parameter. It can be changed any time during runtime, either from SCADA (if OPC is checked), or from IDE. The values are kept even after runtime shutdown and restart.

Each function block has at least one output.

Outputs

Results of calculations in the function block. There are some special properties of outputs:

Manual override

Manual: If checked, the output terminal does not send the computed value. It is overridden by a manual value.

Manual value: The value which is present on the output terminal if in *Manual* mode.

The manual mode is useful for testing and commissioning purposes. In the schema, manual override is marked by a yellow hand icon close to the output terminal. At the

same time, if there is at least one manual override in the application, there is a yellow hand icon at the toolbar.

2.4.10 Inserting of function blocks

Let's make a simple controller – a room thermostat.

1. In the test project, define two I/Os:
   - room temperature sensor
   - boiler enable.
   See 2.4.3, Adding modules.

2. Insert those variables into the ladder of an empty schema: room sensor to the left, boiler enable to the right.
   See 2.4.7, Insert a variable into schema.

Variables in the schema
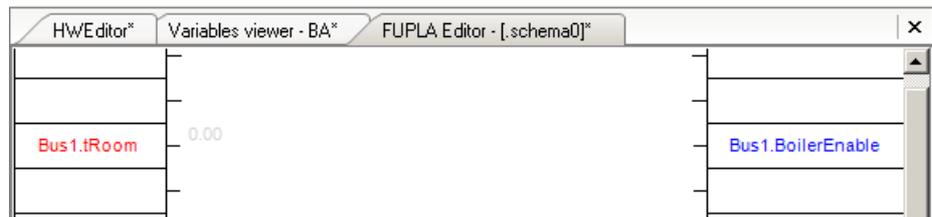
3. In the **Function block panel**, navigate to **Basic analogue boxes, Controllers, Reverse acting (Heating), On/Off control (B31)**.

4. Drag & drop this block to the schema.

5. The block can be dragged across the schema and positioned at any suitable place. To select a block in case there are more block in the schema, left-click anywhere within the boundaries of the block. Selected block is surrouded by a green frame.

Selected block

6. Right-click in the box invokes a context menu:
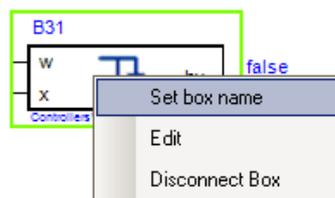
Select **Set box name** and rename the block if necessary. Remember that the block name becomes part of the variable name both in FUPLA and in OPC (SCADA).

7. For training purposes, rename the block to *Thermostat*. Confirm by <Enter> or **OK**.

8. Now have a look at the **Variables viewer** window. Set the **Name** filter e.g. to *Ther* (a substring of „Thermostat") and see the list of corresponding variables. All of those refer to the *Thermostat* function block.

2.4.11 Connecting of function blocks

Go back to the **FUPLA Editor** tab. We will now connect the block to the variables so that the variables are processed by the block.

1. Insert a setpoint variable. It may be either a SW variable (see 2.4.8, Software variables) or rather another function block (the most suitable one is Basic analogue boxes, Sources, Fixed value (B1)).

2. Set the default value of the setpoint to 20°C.

3. Rename the B1 block to *Setpoint*.

4. Locate the „short wire" of the temperature variable in the variable ladder. The cursor changes to a symbol of hand.

5. Left-click and drag the green wire as far as to the „x" terminal of the *Thermostat* block. The wire snaps automatically to the input block terminal.

6. Release the mouse button.
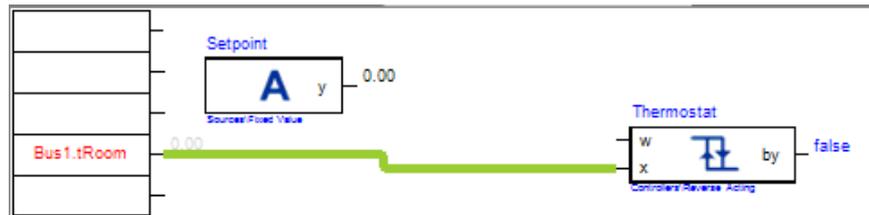
7. The variable is now bound to the „x" terminal of the controller.

Signal type mismatch

8. In case the signal types do not match (attempt to link a Boolean variable to a Double input, for example), the connection changes its colour to red and does not snap to the target terminal.

9. In the same way, link the *Setpoint* to the „w" terminal, and *Thermostat* output to the *Boiler enable* variable (which controls a physical output – a relay in the I/O module that switches the boiler on).
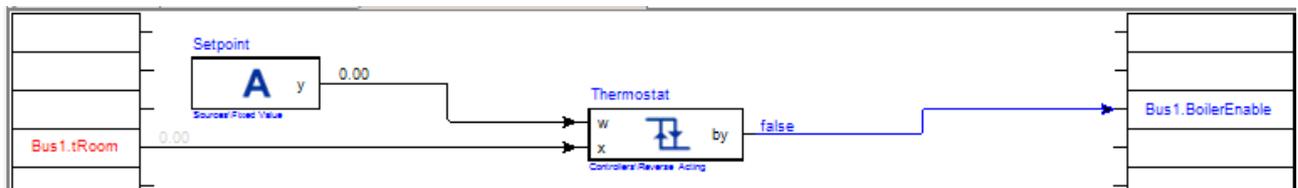
The application is now ready to be compiled.

2.4.12 Compiling

Compile icon

1. Click on the **Compile** icon at the toolbar
   or
   press <Ctrl>+<Shift>+<B>
   or
   select **Local, Compile** from the menu.

2. The IDE prompts if the project shall be saved. Confirm with **Yes**. An unsaved project can not be compiled. Note that after saving, the asterisks at the work panel tabs disappear (= no unsaved changes).

3. In the lower part of the work panel, a *Compiler output panel* opens and displays the result of the compilation.

Locating a compile error

4. If there are errors in the compilation, they list in the *Compiler output panel* and have to be corrected before the program can be run. Below is an example of the most

common error. The **Node** column shows the block name. A left double-click on the error line navigates to the block concerned and makes localization of the problem easy.
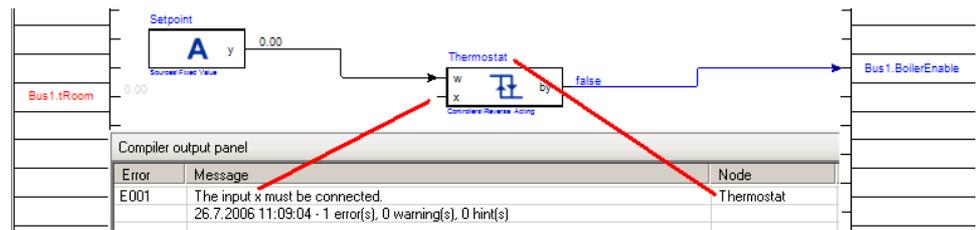
5. After a successful compilation, the program can be run.

2.5 Running a program

The program can be run either **locally**, or at a **remote** process station.

With remote connection, it is possible to:

- connect to a process station via TCP/IP

- configure the runtime

- upload a new version of the application into the station

- run and stop the application

- monitor and change variables during runtime

- download the application and its parameters to IDE for a backup.

This is the typical usage of the IDE. However, now we will run the application directly in the IDE (local connection) – the I/O modules shall be connected to the COM port of the PC where IDE is running.

There are three run modes:

Full run

Both communication and evaluation of the blocks is active. This is the normal run mode. NB. in the unlicensed version, the communication **stops after 10 minutes**.

Evaluation only

The communication driver is not active („unplugged"). The values of the inputs can be set manually by clicking on the variable box in the ladder.

A dialog opens where it is possible to set the value of the input variable. It will not be overriden by the field values as the communication is not active.

Why evaluation run? This evaluation run can be used for program testing where the field components are not commissioned yet.

Communication only

To check the peripherials without their influencing by the application. Only the communication drivers are active („plugged"), the function blocks are not processed and do not write their values to the HW variables.

The input variables' actual values display in the **Variables viewer** as well as in the ladder in the schemas.

The output variables can be set manually by clicking on the variable box in the ladder and the set values are communicated to the I/O modules.

Let's run our application in an evaluation run first. Later, if available, I/O modules and sensor can be connected.

1. Click the Evaluation only icon
   or
   select **Run – Run evaluation only** in the menu.

Show links' values [1]

2. Analog and digital values will be displayed close to the blocks' output terminals. They can be switched off by toggling the **Show links' values** toolbar button.

3. Set the room temperature setpoint to 20°C in the *Setpoint* block.

4. Set the room temperature in the variable ladder to 17°C (see Evaluation only). The output is *True*.

5. Set the room temperature in the variable ladder to 23°C and watch the controller's output. It will change to *False*. The boiler was disabled due to high room temperature.

   Further suggested steps:

6. Play a bit with settings of the Thermostat parameters.

7. Insert a PI controller to control an analogue valve.

8. Add a time scheduler which performs a night depression functionality. Use either binary scheduler (HVAC – TPG - ... in the block library) and a switch, or analog scheduler.

9. Add an Enable terminal to the controller (based e.g. on a digital input from the fire security system).

10. Employ various blocks (e.g. heating curve) and design a complete heating circuit with outside temperature and room compensation.

To upload a program to a MiniPLC (IPLC201, 301, 5xx), follow separate document **RcWare MiniPLC IDE**.

3 Licencing

3.1 Principles

As the SoftPLC can run on any PC platform (not exclusively IPCT.1 or other Domat Control System hardware), the IDE as well as runtime are licenced.

The licences can be obtained as:

- hardware (COM port) key *or*
- software (file).

Hardware key

The hardware key (ordering code **HWS01**) plugs into a serial port of the runtime machine. It is transparent for any RS232 communication and can (but need not to) be installed on the same port which serves the I/O modules. It enables unlimited number of data points at both runtime and IDE. It is **freely transferable** (not bound to a particular process station or PC) and does not require any registration nor activation.

This way of licensing is very flexible and useful especially for tests, engineering, and emergency.

3.2   Hardware key licencing

Configuration of the runtime for hardware licencing:

1.   In the runtime, select **Options**.

2.   Set the COM port number to which the hardware key is connected.
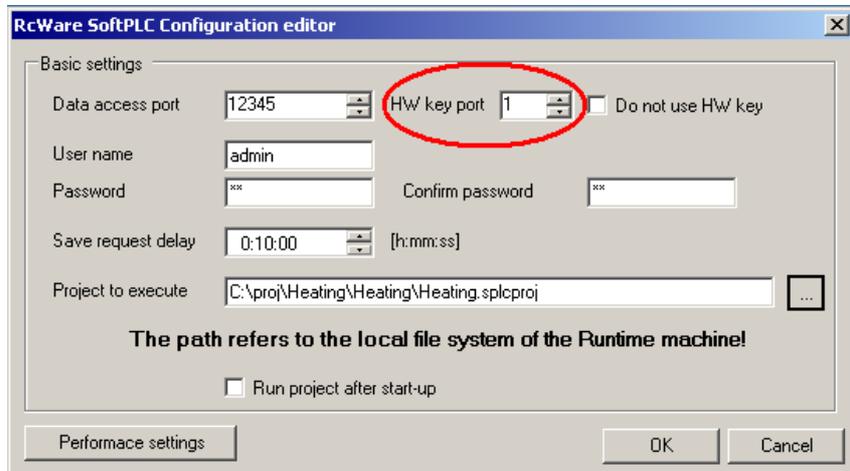
3.   Make sure that the „Do not use HW key" option is unchecked.

4.   Click OK.

5.   Plug the hardware key to the selected COM port.

6.   Restart the runtime.

Configuration of the IDE for hardware licencing:

1.   In the **ESG.SoftPLC.IDE.exe.config** file locate the HW key section:

```
<!--
   COM port that contains HW key.
   Value "0" will disable the functionality.
 -->
   <add key="COM_NUM_HW_KEY" value="0" />
```

2.   Instead of „0", enter the COM port number, e.g. „1".

3.   Save the config file.

4.   Plug the hardware key to the selected COM port.

5.   Restart the IDE.

In case the key does not work properly, copy the **ESG.IO.Serial.dll** file (in **C:\Program Files\RcWare\SoftPLC\Channels**) to the **C:\Program Files\RcWare\SoftPLC** directory.

3.3   Software licencing

If there is a dedicated hardware, such as process station, it may be easier to use software licence. The licences are issued on request and have to be activated for a particular machine.

The licencing process follows those steps:

support@domat.cz   **customer**

ordering of licence
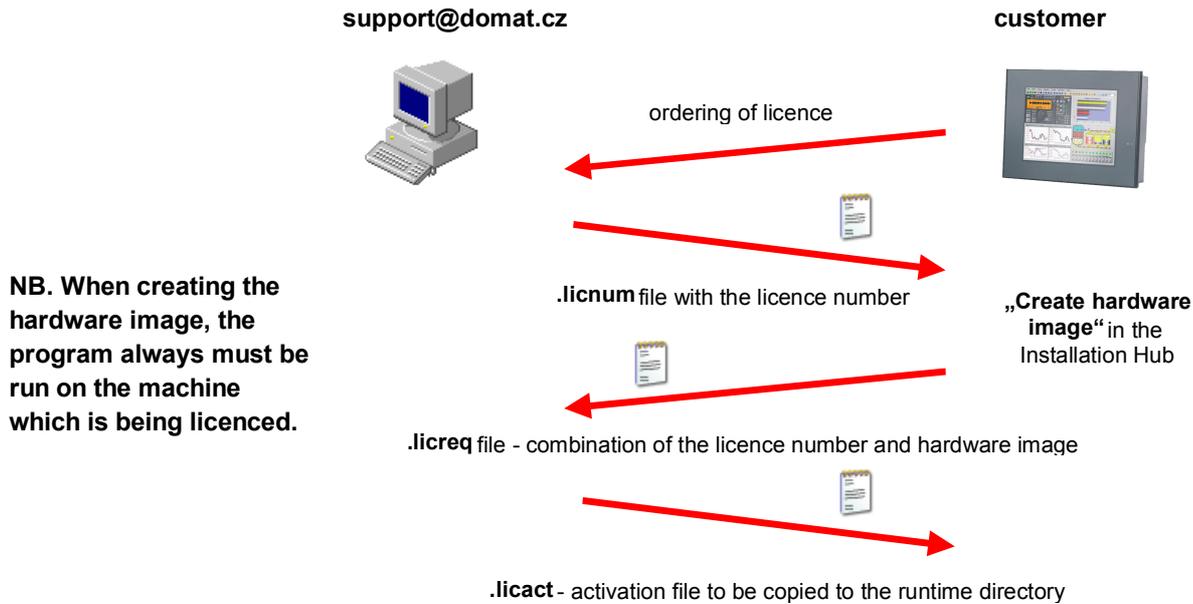
NB. When creating the hardware image, the program always must be run on the machine which is being licenced.

.licnum file with the licence number   „**Create hardware image**" in the Installation Hub

.licreq file - combination of the licence number and hardware image

.licact - activation file to be copied to the runtime directory

It is recommended to save the activation file for case of reinstallation of the operating system etc. However, the file can be retrieved from *support@domat.cz* free of charge.

Configuration of the runtime for software licencing:

1. Get the licence activation file as described above.

2. Make sure that the licence activation file is copied in the runtime directory (the same where the **ESG.SoftPLC.Host.WinForms.exe** file resides, by default **C:\Program Files\RcWare\SoftPLC.Runtime**).
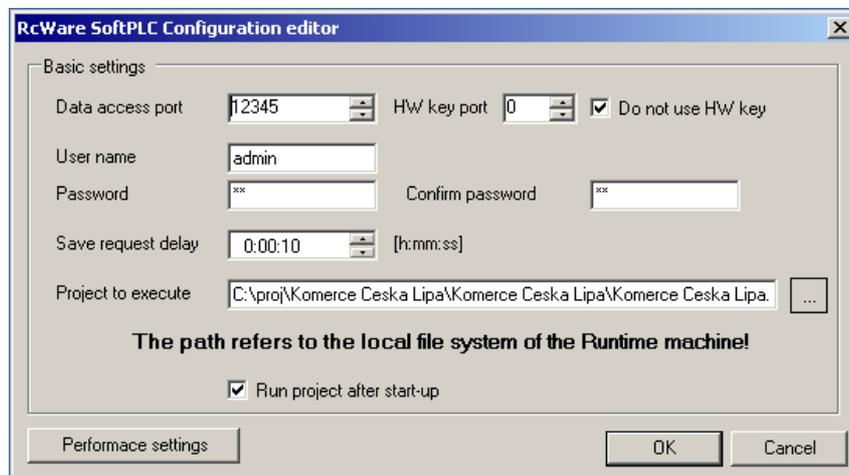
NB. After having set the complete runtime configuration, do not forget to store the data permanently (Commit CF and reboot) if Windows XP Embedded is the runtime operating system.

3. In the runtime, select **Options**.

4. Select the "Do not use HW key" option.

5. Click OK.

6. Restart the runtime.

The licence has unlimited number of datapoints and unlimited expiration time. However, there are "soft limits" for datapoint amount connected to one bus – consult the *System Design* handbook.

4 Runtime configuration

The runtime must be installed on the target machine. If your process station has no runtime preinstalled or you need to install a more recent version follow those steps:

(tbc)

Runtime Configuration

Integration into RcWare Vision – see separate document