pbsSoftLogic User Manual

Linux/WinCE/Win32 Target

Version: 1.7.0

Author: Kamjoo Bayat – kamjoo.bayat@pbscontrol.com

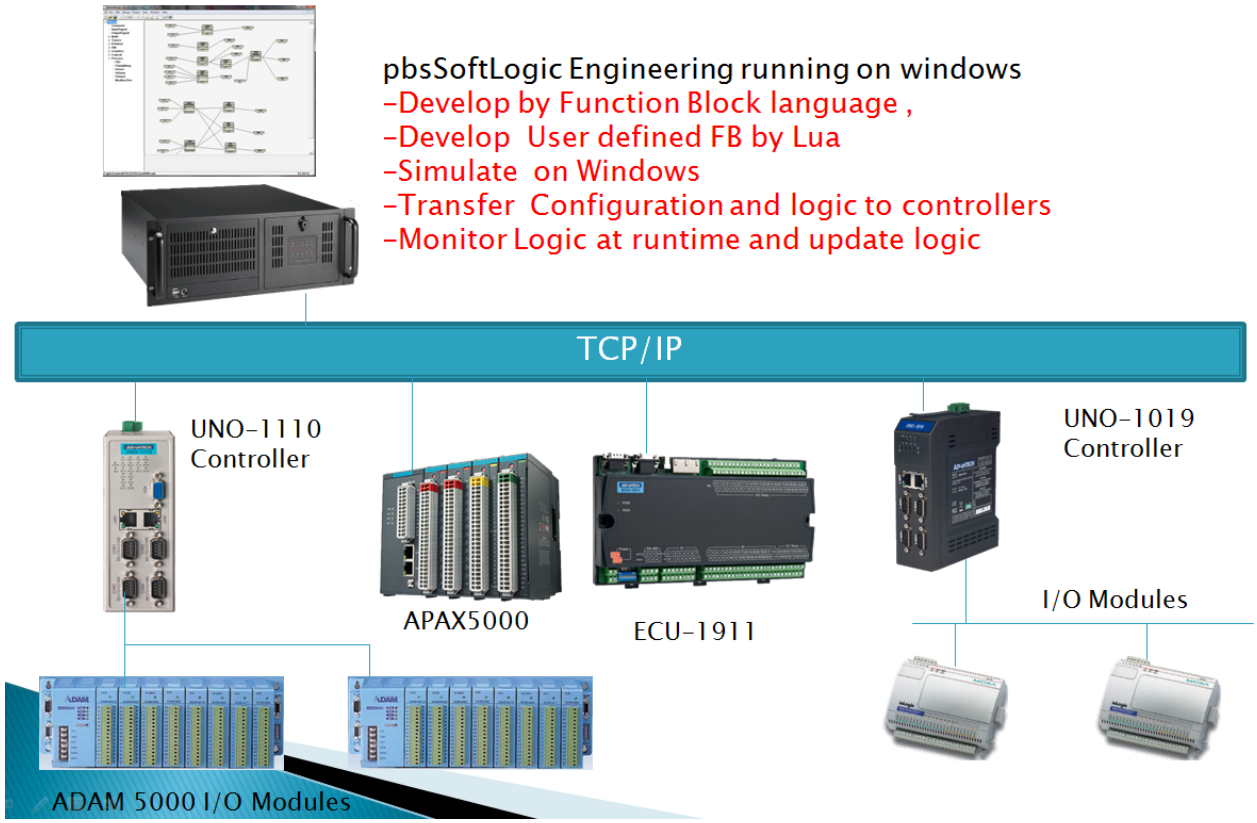
Date: Q2 2014

1 – Introduction

pbsSoftLogic is open RTU/PLC Programming Environment from pbsControl. pbsSoftlogic is developed by Dot Net technology . Its development version is running on Windows operating system.

pbsSoftLogic has following specifications :

- Standard Function Block programming Environment
- Lua (scripting Language) is supported for user defined Function Blocks development
- Developed application can be run on Embedded Windows , WinCE and Embedded linux OS
- Offline simulation of developed application on windows
- More than 100 Ready and tested Function block for easy programming.



For update version of pbsSoftLogic please visit www.pbscontrol.com

Current Version: 1.7.0

Date: July 2014

Supported and tested Hardware:

1- ECU-1911 with WinCE5.0
2- UNO1019 with WinCE5.0
3- IA240 with embedded Linux
4- UC7112 –Plus with embedded Linux
5- W406 with embedded Linux
6- APAX5522LX with embedded Linux /WinCE
7- UNO-1110 with windows CE 6.0
8- Any PC based controller with windows 32 OS.

2 – pbsSoftLogic installation

pbsSoftLogic – Eng is running on following Operating systems :

WindowsXP , Windows Vista , windows7 , Windows Server 2008 and windows Server 2010 .

You need to install Dot Net Frame 3.5 on your machine for proper operation of pbsSoftLogic .

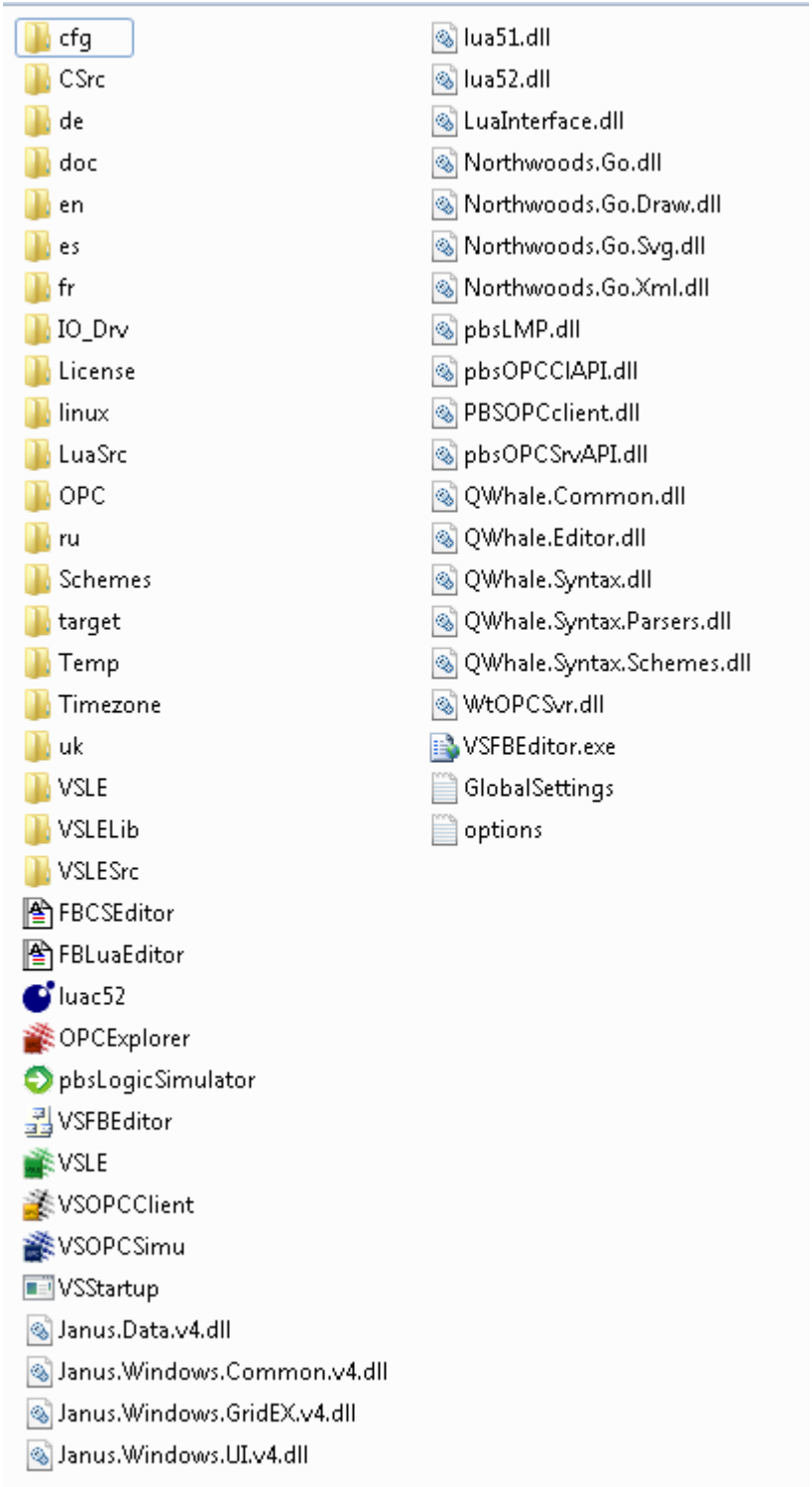
You can download pbsSoftLogic from http://www.pbscontrol.com/download.html

Simply unzip PSLE.rar file and run VSFBEditor.exe .

pbssoftLogic files and directories :

- FBCSEditor.exe User defined Function block editor for Windows and simulation Target
- OPCExplorer.exe OPC Configuration file for connecting to OPC servers with Windows Target
- pbsLogicSimulator.exe Logic Simulator .
- VSFBEditor.exe Main Application for developing Function block projects .
- VSOPCClient.exe OPC configuration runtime for windpws Target . Please refer to user manual of pbsSoftlogic with Windows Target .
- pbsOPCSimu.exe OPC server simulator .
- VSLE.exe Runtime engine of pbsSoftLogic for windows Target .
- VSStartup.exe Startup application for windows Target . This application is running all required OPC Clients and servers based on a predetermined sequence at startup of Windows Controller .
- pbsLMP.dll Logic Monitoring Protocol . will use for Logic monitoring in Linux Controller .
- options.xml basic options of pbsSoftlogic .
- cfg Directory : basic definition of Function blocks , OPC simulator and startup sequence .
- doc Directory : user manual of pbsSoftlogic
- OPC Directory : saved OPC Configuration files by OPCExplorer.exe application
- LuaSrc Directory : Source code of Lua Function blocks
- CSrc Directory : Source code of C Function blocks

- Timezone Directory: Time Zone file for Linux controller
- VSLE Directory: default developed application with pbsSoftLogic . you can put application and its deriver at any location
- VSLELib Directory : inside Function blocks implementation by C# language .(Compiled source code)
- VSLESrc Directory : Source code of Function block implementation by C# Language .


cfg
CSrc
de
doc
en
es
fr
IO_Drv
License
linux
LuaSrc
OPC
ru
Schemes
target
Temp
Timezone
uk
VSLE
VSLELib
VSLESrc
FBCSEditor
FBLuaEditor
luac52
OPCExplorer
pbsLogicSimulator
VSFBEditor
VSLE
VSOPCClient
VSOPCSimu
VSStartup
Janus.Data.v4.dll
Janus.Windows.Common.v4.dll
Janus.Windows.GridEX.v4.dll
Janus.Windows.UI.v4.dll
lua51.dll
lua52.dll
LuaInterface.dll
Northwoods.Go.dll
Northwoods.Go.Draw.dll
Northwoods.Go.Svg.dll
Northwoods.Go.Xml.dll
pbsLMP.dll
pbsOPCCIAPI.dll
PBSOPCclient.dll
pbsOPCSrvAPI.dll
QWhale.Common.dll
QWhale.Editor.dll
QWhale.Syntax.dll
QWhale.Syntax.Parsers.dll
QWhale.Syntax.Schemes.dll
WtOPCSvr.dll
VSFBEditor.exe
GlobalSettings
options

3 – Basic concepts

Writing logic for industrial automation plants and SCADA systems is a critical task. It is not recommended to use low level language like c/c++ and C# for such projects because of following reasons :

1- Not reusable
2- Difficult to transfer project to others and train other engineers for continuing project
3- High risk in application runtime for stability and error free
4- Not future proof
5- Getting Long time for project development

Function Block language is a language for control engineers. They can focus on process logic without

Worry about software part. FB is full graphical language with many tested and ready functions inside.

Using function block language has following benefits:

1 – 100% reusable. There are many tested and ready functions that can be used in different projects with complete document.

2 – It is very easy to train Control and process engineers for using and programming.

3 – pbsSoftLogic is used in many projects and sites in last few years , so there aren't error in the runtime and development environment .

4- You can use pbsSoftLogic and Function block language as framework for whole your Automation Projects. Life time of pbsSoftLogic will be 15 years minimum.

5 – Very easy and shortly you can develop process logic, Simulate and load to controllers.

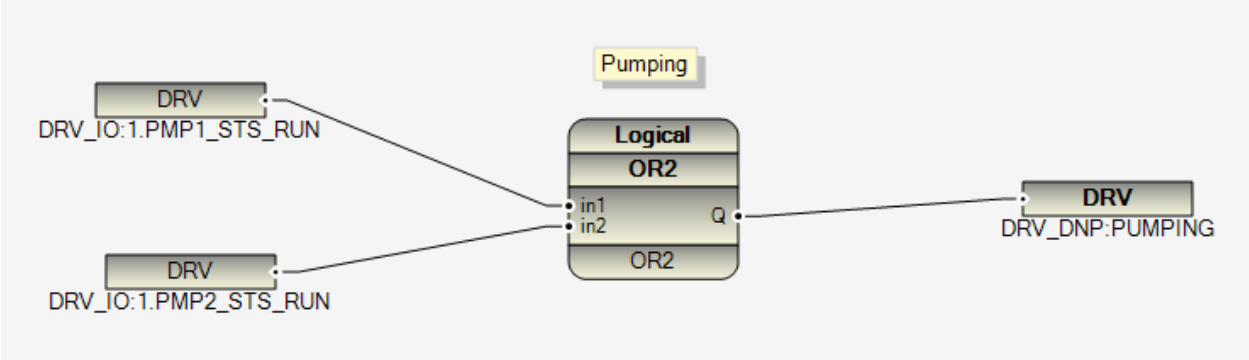
pbssoftLogic is an IDE for developing Function Blocks programs , Simulate , and downloading to Linux /WinCE based controllers . You can use Lua Scripting language for developing new FB .

All FB source code of pbsSoftLogic are open source .

4 – Function Block Programming Language

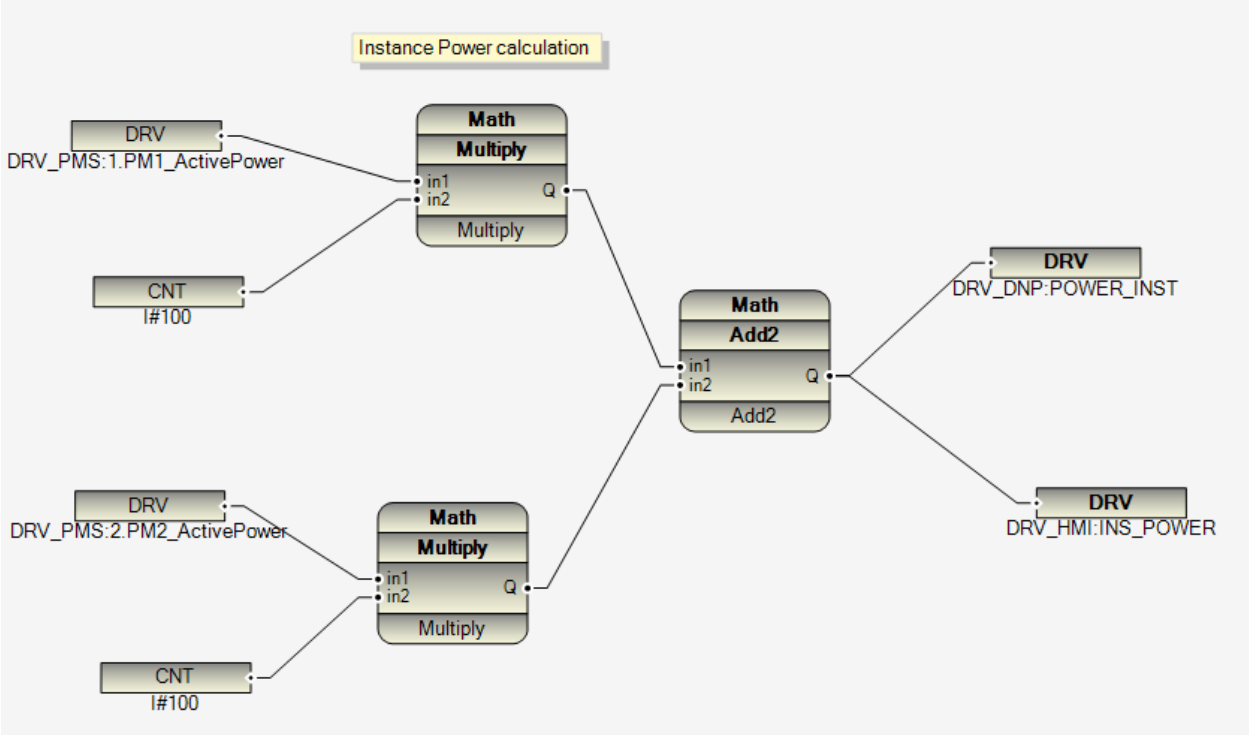
Main element of a Function Block program is FB (Function Block). In Following you can see a few simple examples.

Example1:



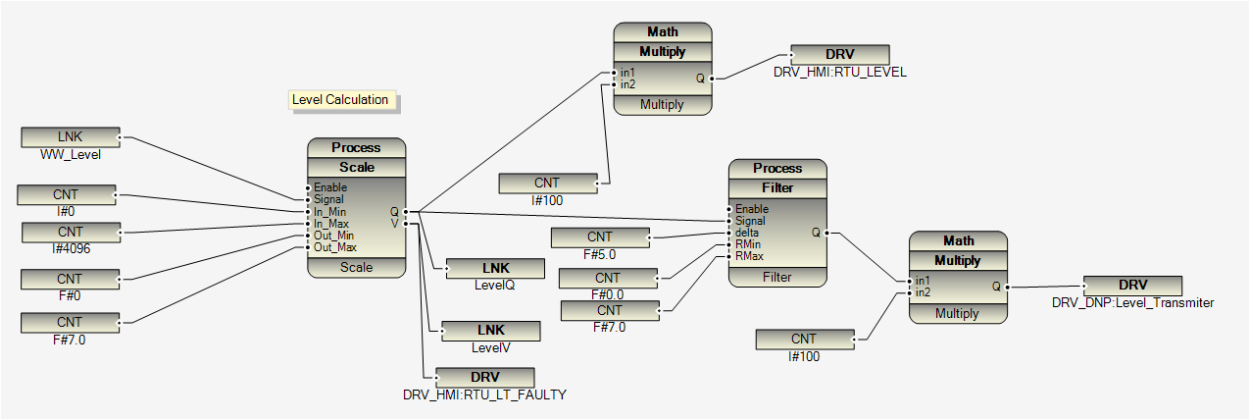
In this logic, two signals PMP1_STS_RUN and PMP2_STS_RUN are input to OR FB and Output will write to PUMPING Signal.

Example 2:



In Example 2 , PM1_ActivePower is multiply by 100 , PMP2_Activepower is multiply by 100 and both results will add together and will write to Power_Instance Signals . (Write on two different sources)

Example 3:



Main Element of a Function Block Program:

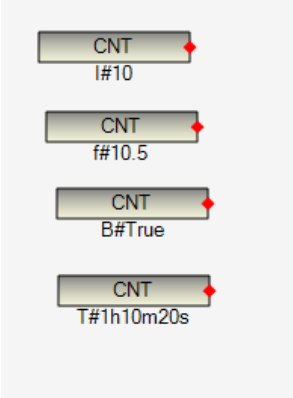
1 – Input /Output Signals: Normally links to Communication Drivers and Local I/O

2 – FB: Ready Function Blocks.

3 – Interconnection between I/O Signals, FBs and between FBs.

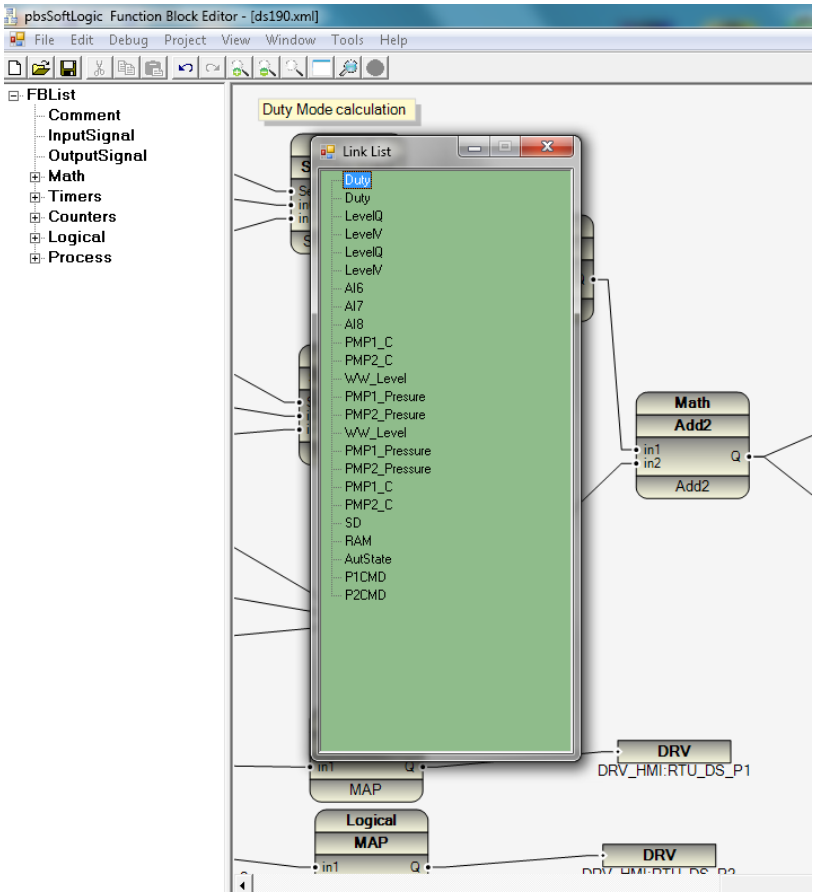
4 – Constant signals: different type of Constant Signals: Integer (I), Float (F), Boolean (B), Time (T)

Constant Signal Format: Type # Value.



5-Internal Link Tags: unlimited internal link tag is possible in logic, but each instance should have different name. Links with same name has same value in logic.

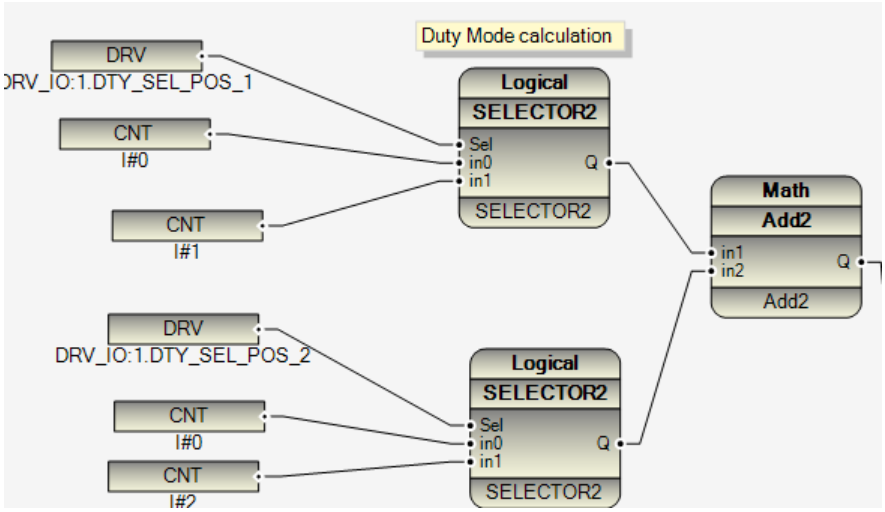
You can see list of all Link Tags in Debug Menu, Link List Item.



By double click on each link item; Logic will focus on Link Signal. So you can easily browse and check all link signals.

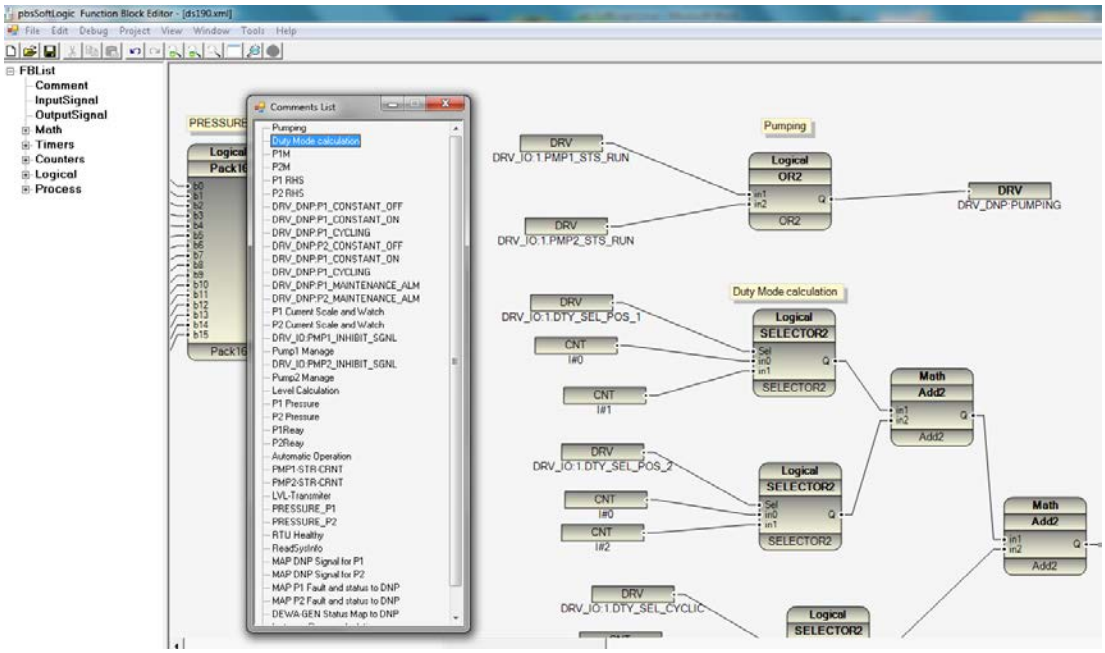
6 – Comments: you can put comment everywhere in logic. Drag a Comment element from FBList and

Drop it in the logic. Then click on Comment and change its content. Comment is like a dynamic size yellow text box.



By selecting comments items from Debug menu, you can see list of all Comments in the logic.

By Double click on any comment, logic will focus there and you can easily browse all logic by comments.



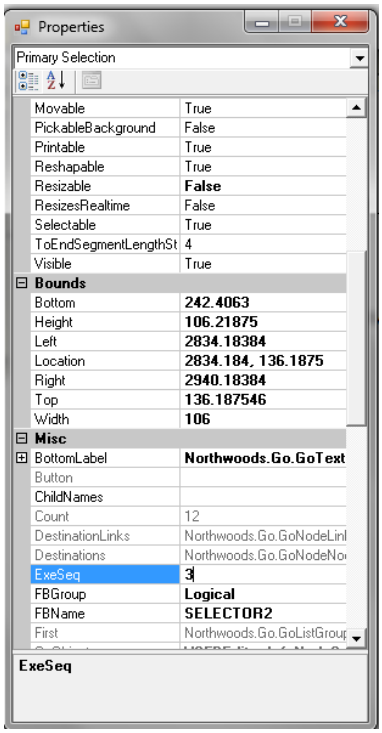
Function Block Programming Rules:

1 – FB Inputs (Left side) always connect to one source. You can connect one source (I/O Signal, Internal Link Tag, and Constant) to different FB Inputs; But Multiple Source to One FB Input is not valid.

2 – FB Outputs (Right Side) can be connecting to different Signals. (Not Constant Signals)

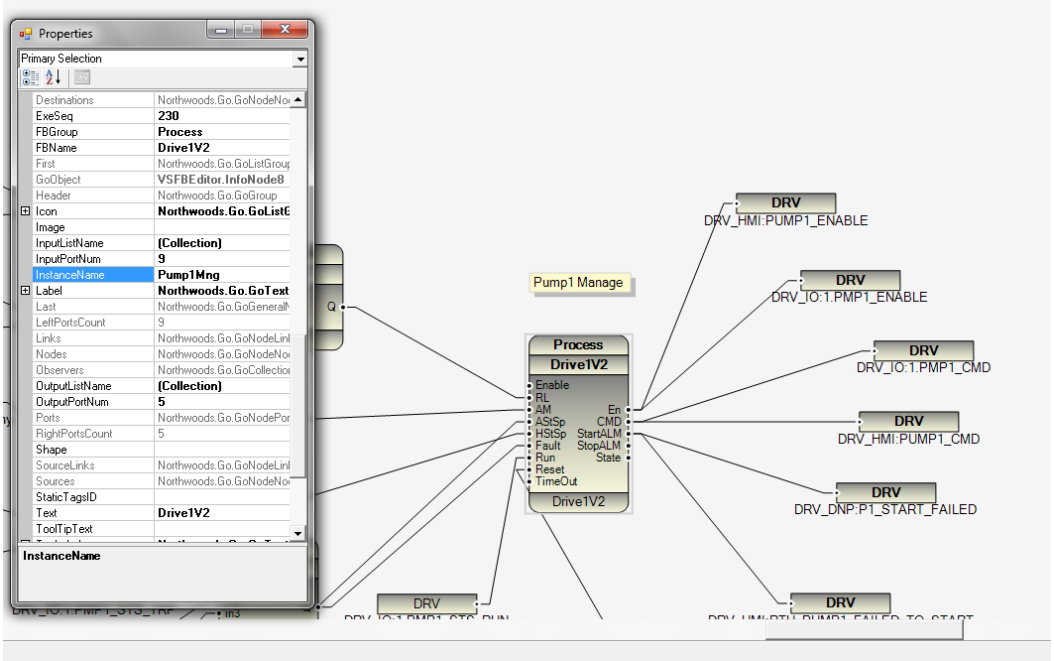
3 – There is no limitation on number of FB interconnections level.

4 – Logic execution: each FB has an Execution number. Click on FB and press F4 , you can see FB properties window . Scroll properties to find ExeSeq .

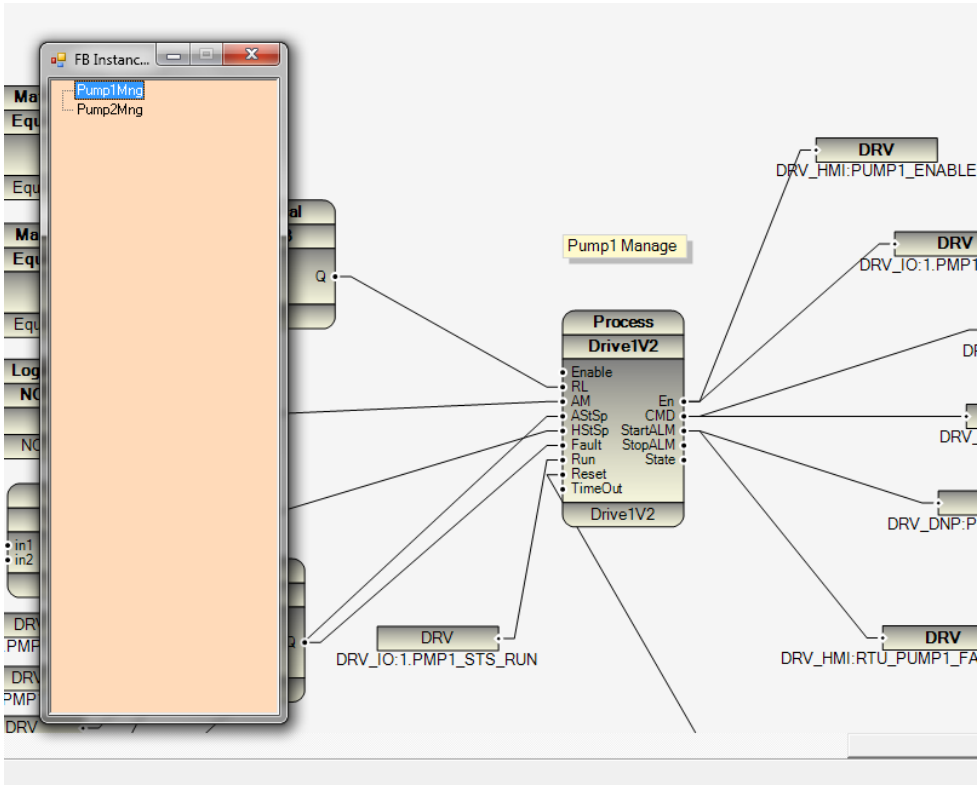


When you start to develop logic , FBEditor will increase ExeSeq number for each FB that you use automatically . but you can change its sequence and by this way , you can control execution sequence of logic . We advise to set all ExeSq numbers manually , because when you copy paste some part of logic , FBEditor will put same values for pasted elements . FBEditor will sort all Fbs by ExeSeq number and compile and make output file by ExeSeq order .

5 – Logic FB Instance name : each FB has FBName and instance name . these two properties are equal by default . but you can change Instance name to any unique name in your logic . Suppose you are controlling a Pump by Drive1V2 FB . By changing FB Instancename to “Pump1Mng” , Compiler will use Pump1Mng as identification of FB at compile time . By default it is using PartID property which is always unique in the logic.



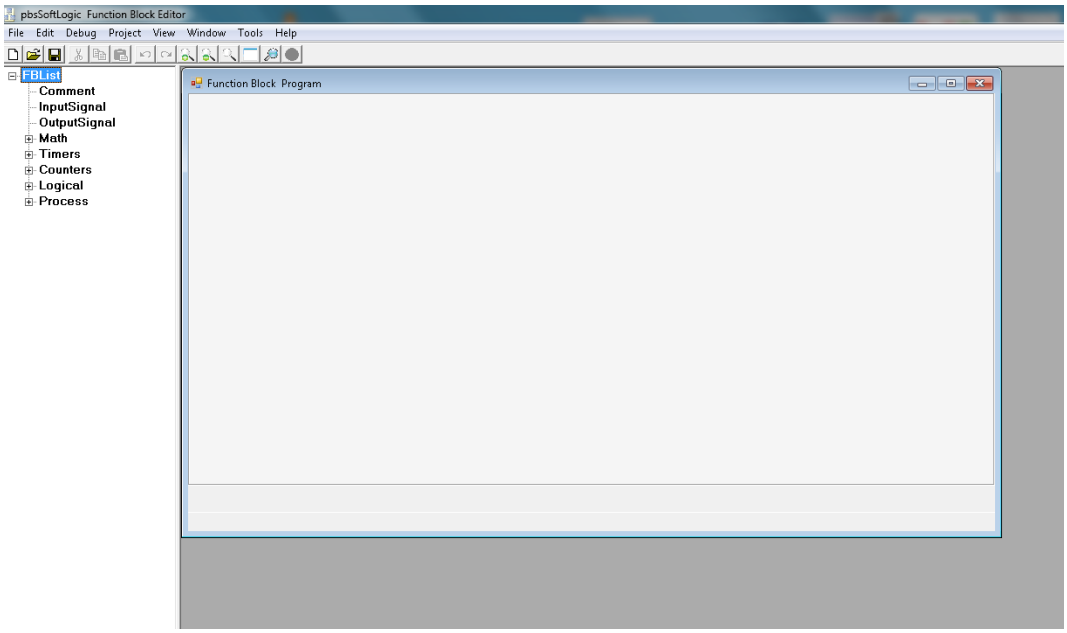
You can browse logic by FB Instance name from Debug menu, FB Instance List item. By Double clicking on Instance name, Logic will focus on that part.



5 – Quick Startup

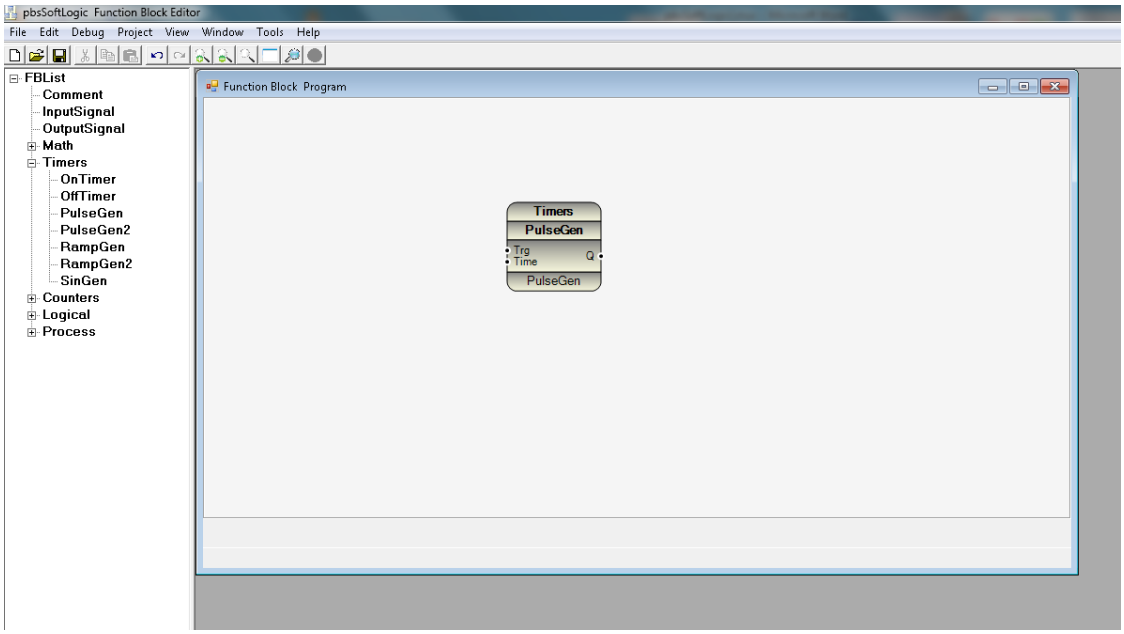
In this segment, you will write a simple logic with PSLE and Simulate and run on Linux controller.

Step1: Make a new Application with PSLE. Run VSFBEditor.exe . In File Menu, Select New.



In left panel, you can see different ready FB, and in right panel Function Block application area.

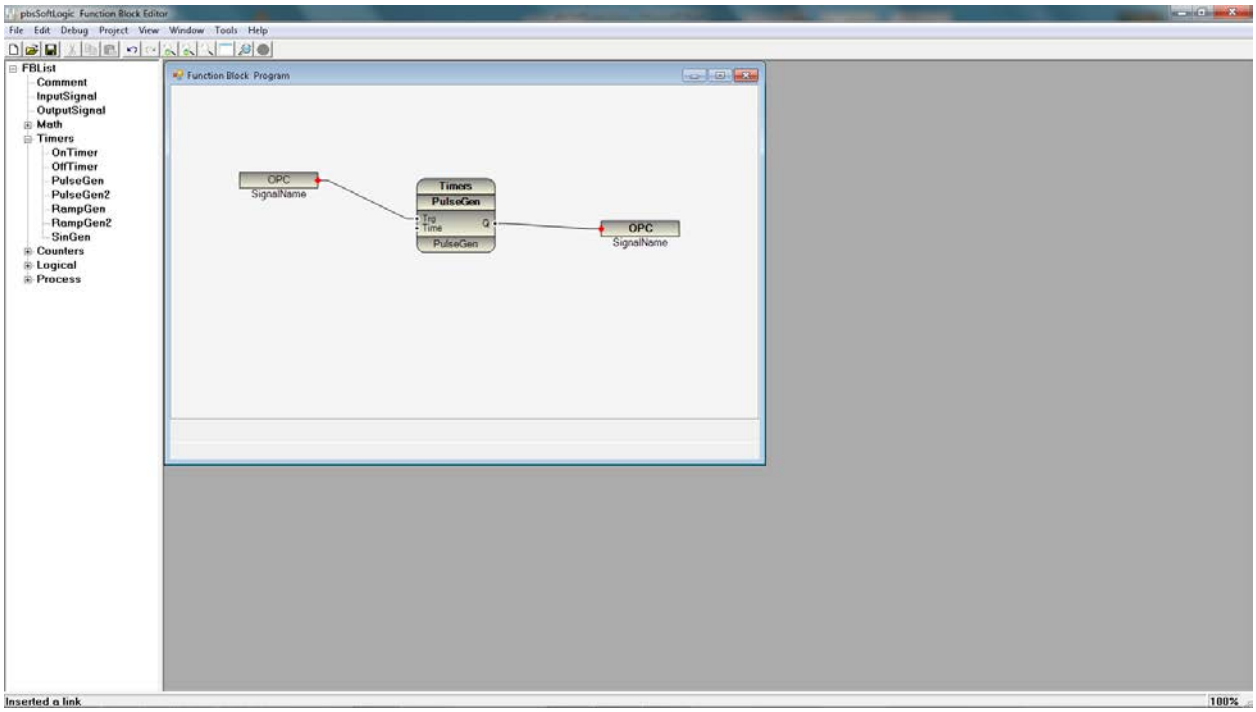
Open Timers Group and select PulseGen . Drag and Drop it to program area .



PulseGen is generating continues pulse, with same time duration (Low and High).

When Trg input is changing from low to high (0 to 1), Pulse train will start at Q output with Low and High Duration equal to Time input.

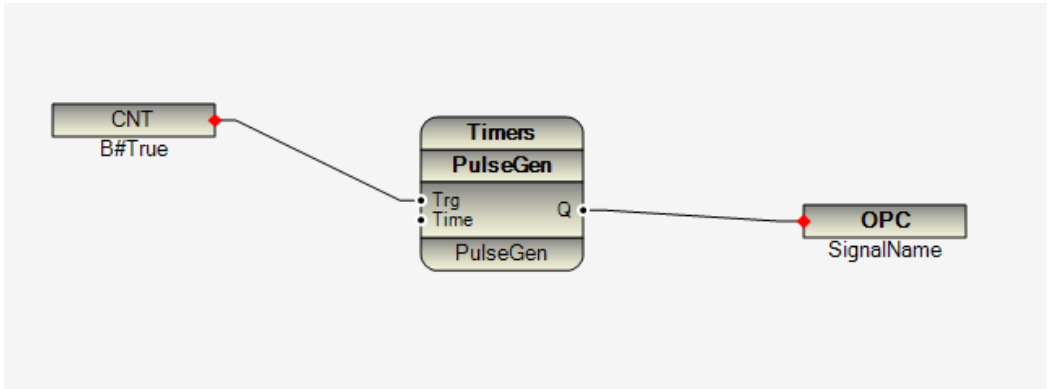
In FB list panel, drag and drop Inputsignal and connect it to Trg Input. Then Drag and drop OutputSignal and link it to Q output. Leave Time input without any connection.



When an Input Port is not connecting to any signal, it will take default value that is preset for each FB (you can change FB Input Default values).

Click on InputSignal which is connected to Trg Input of PulseGen FB . Click on OPC name and change it to CNT (Constant).

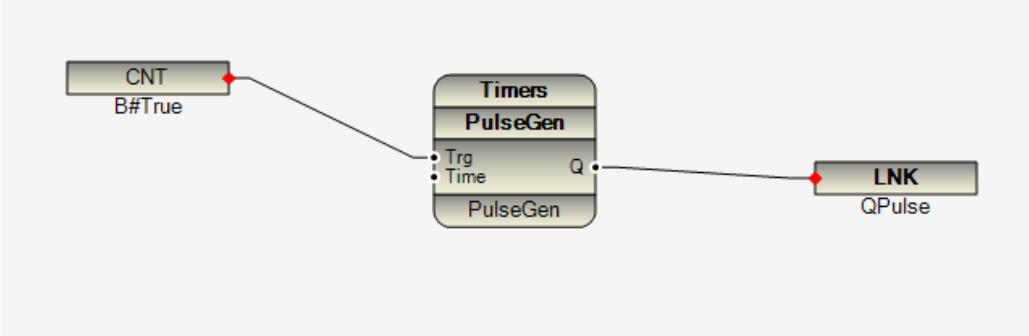
Click on SignalName and write B#True.



Click on OutputSignal that is connected to Q Output of PulseGen FB .

Change OPC name to LNK(Link) . Click on SignalName and Write any name you want. Like “QPulse”.

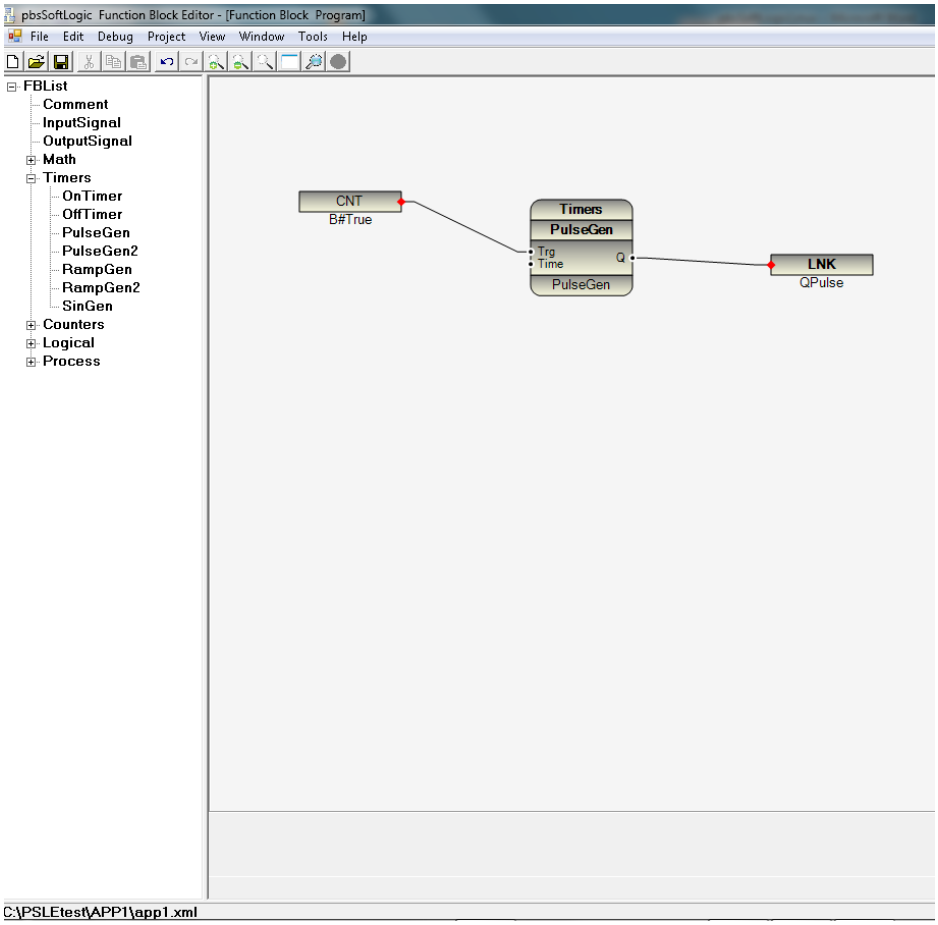
LNK is like internal tag in Logic and you can have any number of internal Tags.



Save you logic. Click on save Button at top.

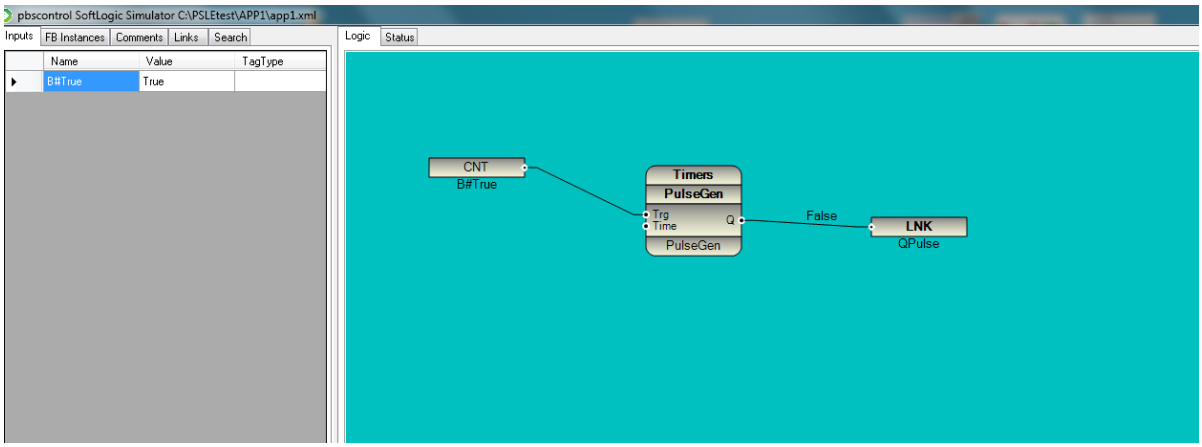
Make a new Directory in C:\PSLETest\APP1 and save your logic with app1 name inside APP directory .

At bottom part of Editor you can see full path and name of your logic .



Step2: simulate your logic.

From Project menu, select Simulation. You will see following page:



In the left panel you can see Logic signals and in right side, you can see your application.

Note: for proper running of Simulator you need to set following parameters in c:\psle\options.xml file.

```
<Node>
    <Name>ResourcePath</Name>
    <Desc>Resource Directory Path</Desc>
    <Value>e:\Resource</Value>
</Node>
<Node>
    <Name>TempPath</Name>
    <Desc>Temp Directory Path</Desc>
    <Value>e:\Temp</Value>
</Node>
```

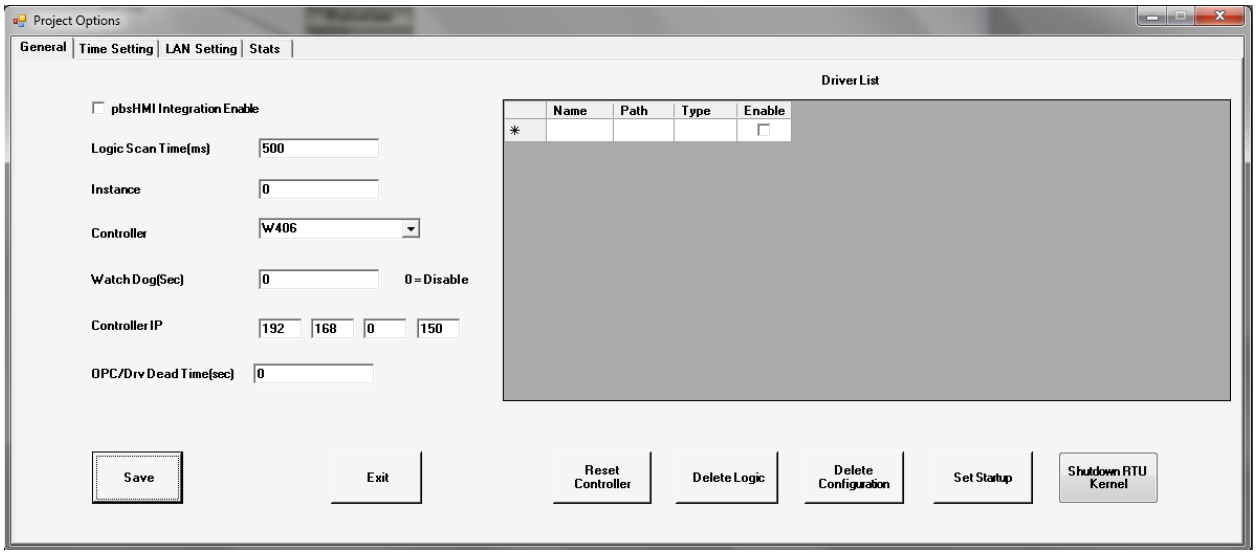
We advise to install RAM Disk Driver on your PC , because Simulator and Windows Runtime Engine will write all Function block static data to TempPath .

You can download very professional and free RAM Disk Driver from http://memory.dataram.com/products-and-services/software/ramdisk Web Site. We tested Data ram disk in many projects and it is 100% compatible with pbsSoftLogic .

After you install Data RamDisk change TempPath and ResourcePath to RAMDisk Drive.

Step3: Edit Project Settings

Open Project menu and select settings. Select Controller type and type IP address of controller.



You can do following tasks with Project Settings:

1 – Changing Time Zone and current Time of Controller (Linux Kernel)

2 – Changing LAN Configuration of Controller (Linux Kernel)

3 – See status of controller. (Linux Kernel)

4 – Defining Communication protocols for logic.

5 – Setting Controller scan time, IP address, controller Type and Watch dog parameters.

6 – Deleting Controller Logic and Configuration

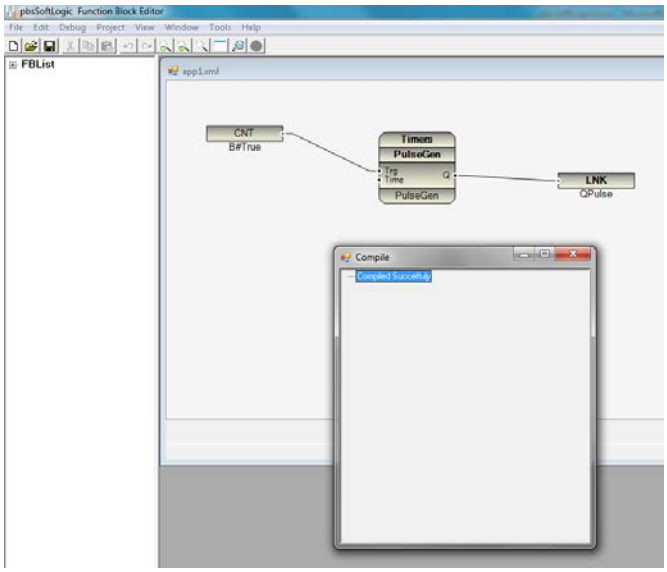
7 – Reset Controller

8 – Shutdown Controller Kernel (Linux Kernel)

9 – Set Startup parameter in Controller. (Linux Kernel)

Step4: Compile Logic

From project menu, Select Compile. It will show error list of your logic if there are errors in logic.



Compiled logic is an XML file with the same name of Logic but with extension c11 . So your compiled file for APP1 Logic is APP1.c11 .

Step5: Transfer Logic and configuration to Controller.

Logic and Configuration files are transferred by FTP to controller, so no need to be connected to controller for transferring logic and configuration.

From Project menu, select Transfer Logic. It will transfer your compiled Logic to Controller.

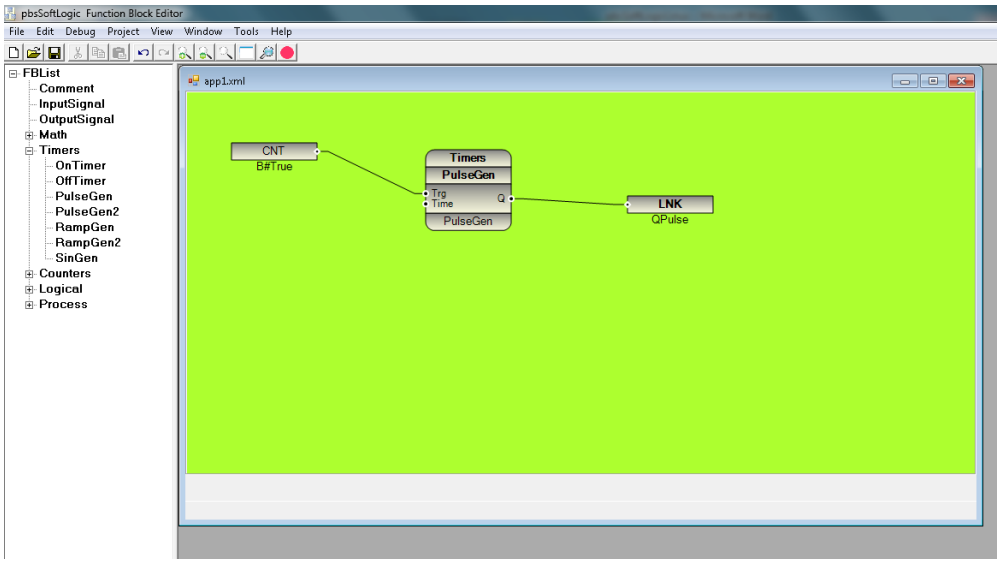
From Project Menu, Select Transfer Configuration. It will transfer App1.lx file to controller.

When Configuration is transferred completely, it will show “Configuration Transferred” Message box.

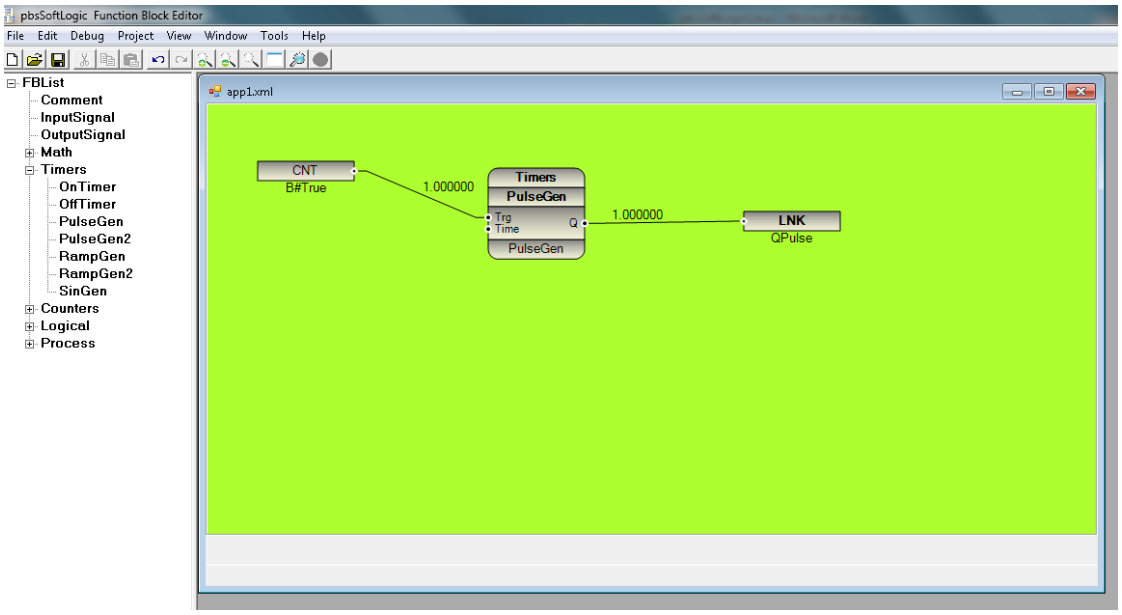
Reset Controller from Setting Page.

Step6: Monitor your logic at Controller

From project menu, Select Connect to controller. If your controller is connected to your development PC and runtime kernel of PSLE is installed on Linux/WinCE Controller, It will connect and your logic page color will change to green.



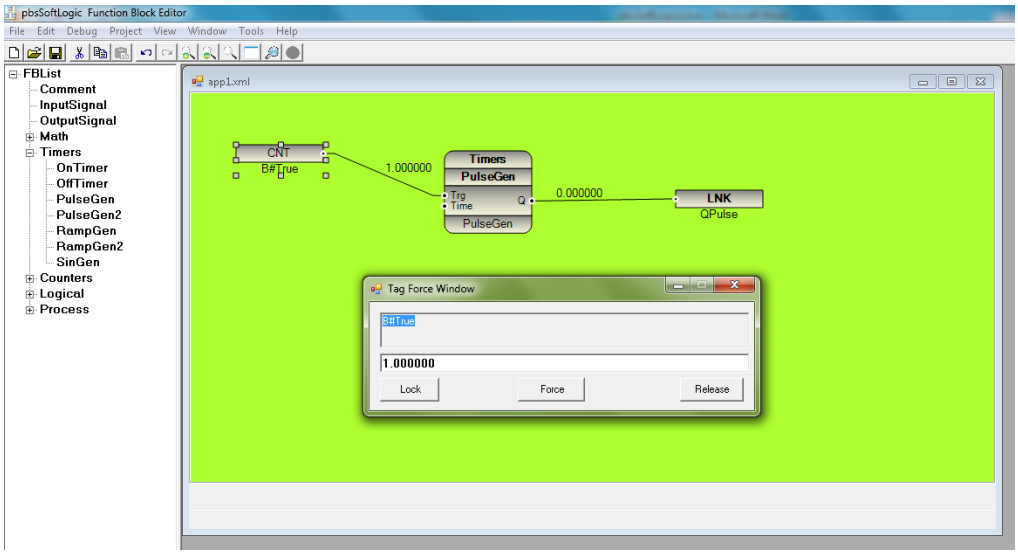
From Project Menu, Select Connect to controller, Then Select Monitoring On.



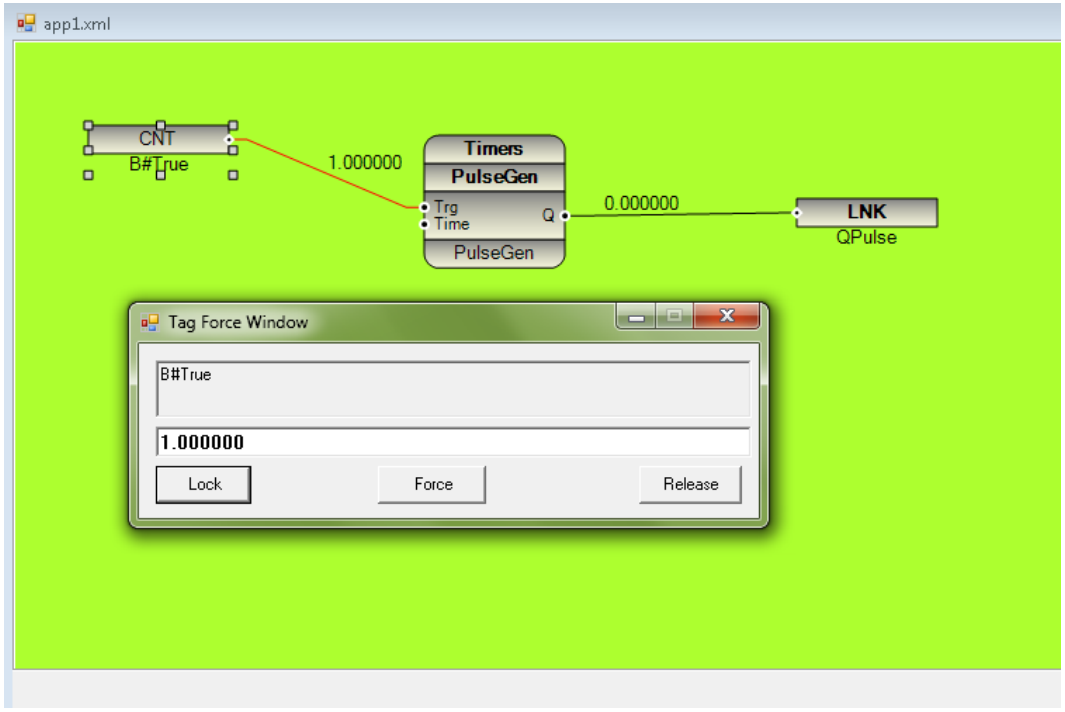
Small LED at top will start to blink and it shows that logic is in monitoring state.

Step7: Force signals

Right Click on CNT:B#True input Signal and select force .



In Tag Force Window, Click on Lock Button. The link between InputSignal and PulseGen/Trg input will be change to red.



Change Value of Signal to 0 and click on Force. PulseGen/Q output will stop. Again change signal value to 1 and click on force button. Again PulseGen/Q output starts to change its value between 1 and 0.

Click on Release button and link color will change to black.

Note:

1 – In Linux/WinCE Kernel runtime, all signals type is double. So you will see value 0 and 1 for Boolean (Digital) Signals, not true/ false.

2 – When you Force a tag, it will force just input signal of FB not tag. It means if you use TAG1 in different location in logic, and when you force it in a FB, it will force only for that FB not whole Logic.

3 – Only FB inputs can be force.

Step8: Disconnect from Logic.

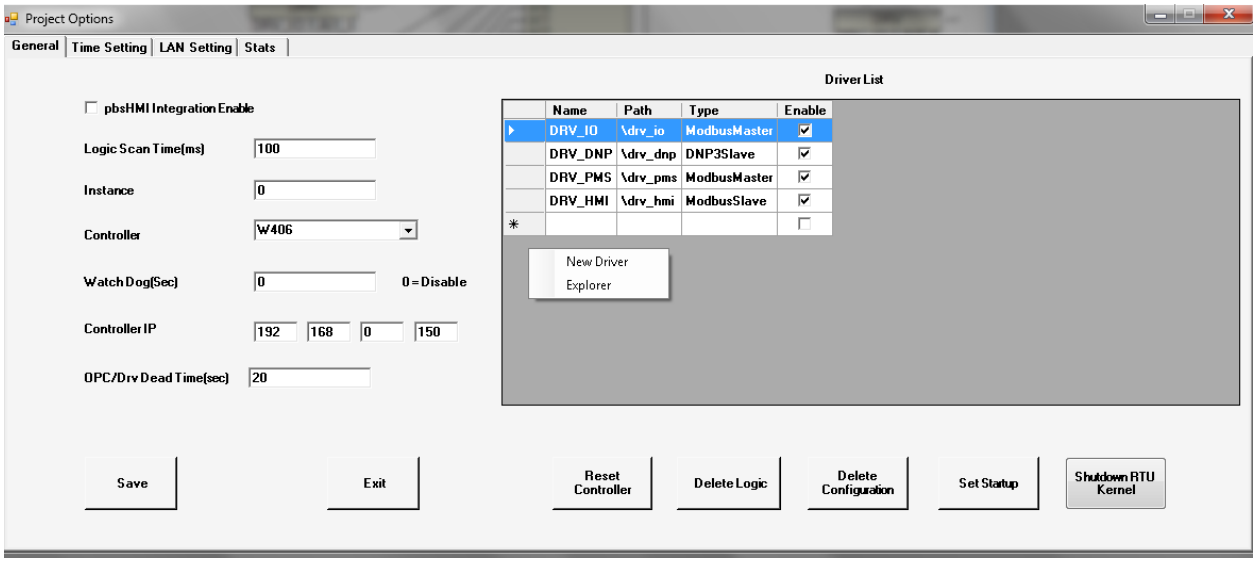
In project Menu, Select Disconnect From Controller. Logic page color will come back to smoke white and all logic monitoring values will hide.

6 – Modbus Master Configuration and integration with remote I/O Modules

PbsSoftLogic supports Modbus Master Driver for communication with I/O Modules and other Modbus Slave Devices. You can set modbus master driver communication parameter from project setting page.

In project setting page, you can see list of configured drivers for your logic.

Right click on driver list, you can add a new driver or explore defined driver.



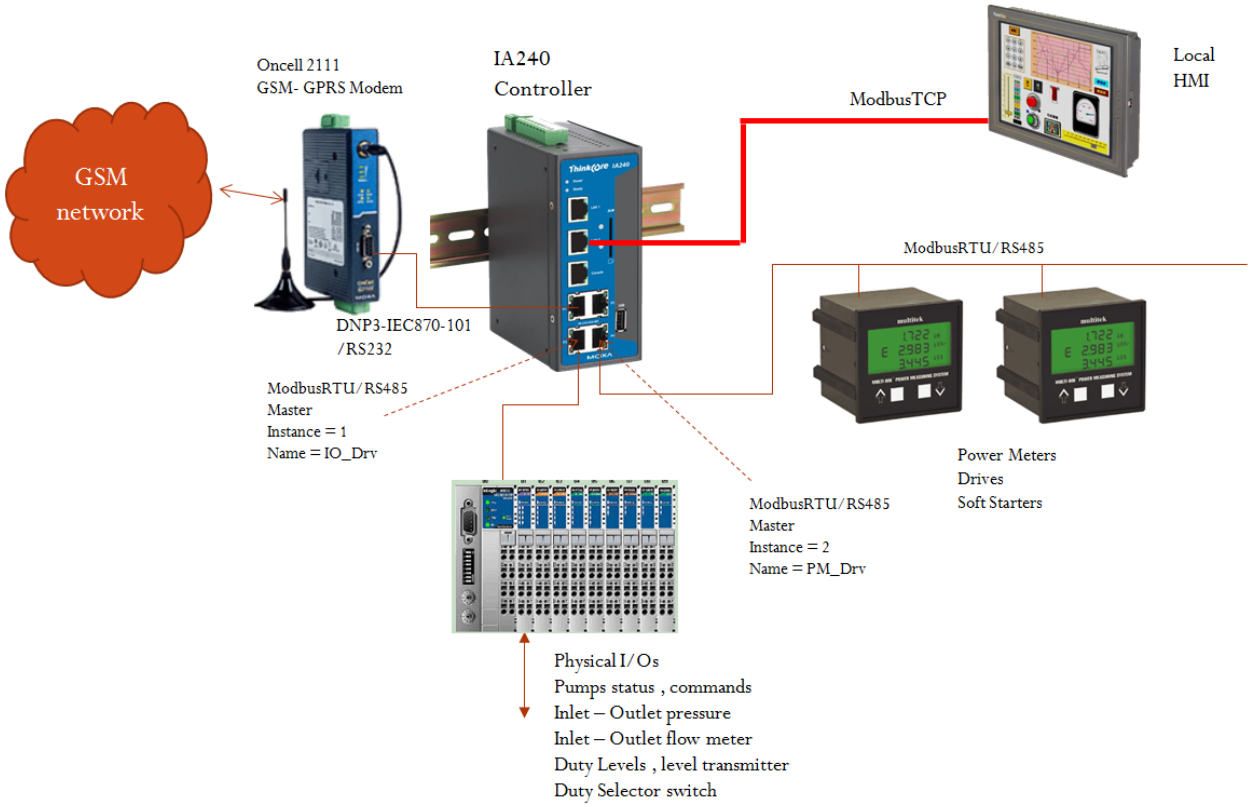
For defining a new Modbus Master Driver, right click on Driver list and select New Driver.

In new driver page, select communication protocol, Type Driver Name and select Driver instance.

Driver : pbsSoftLogic supports ModbusMaster , ModbusSlave , DNP3Slave and IEC870-5 Slave protocols . Select ModbusMaster for Modbus Master protocol.

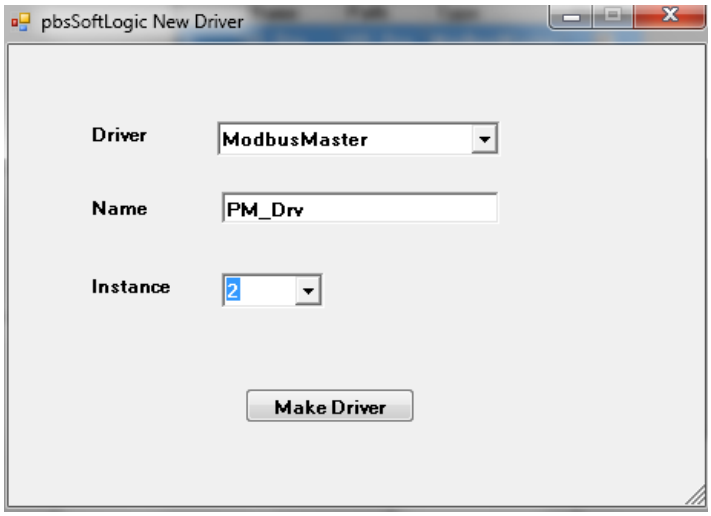
Name: Unique Driver Name.

Instance = Instance number for each type of Driver. If you have two Modbus Master Network in project , then you need to define two ModbusMaster Driver with Different name and different instance number. Look at following example, IA240 should connect to I/O Modules and Power monitor network by two different Modbus Master networks.

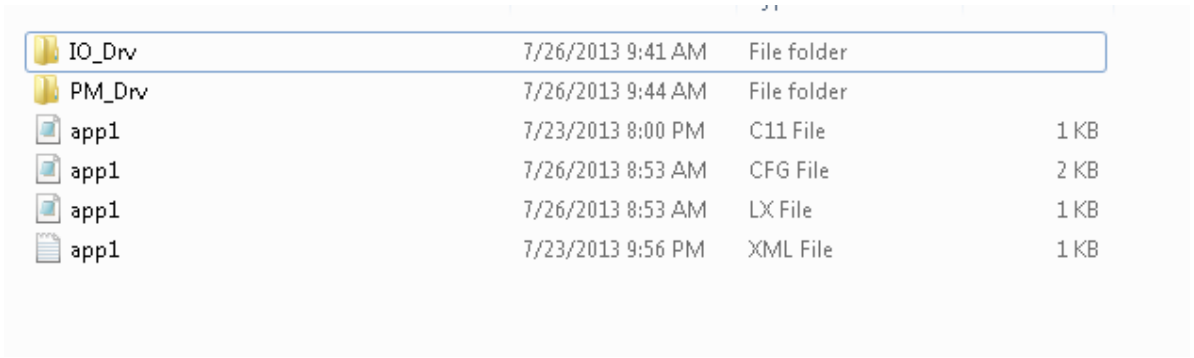


Configuration for Modbus Master Driver for I/O Modules:

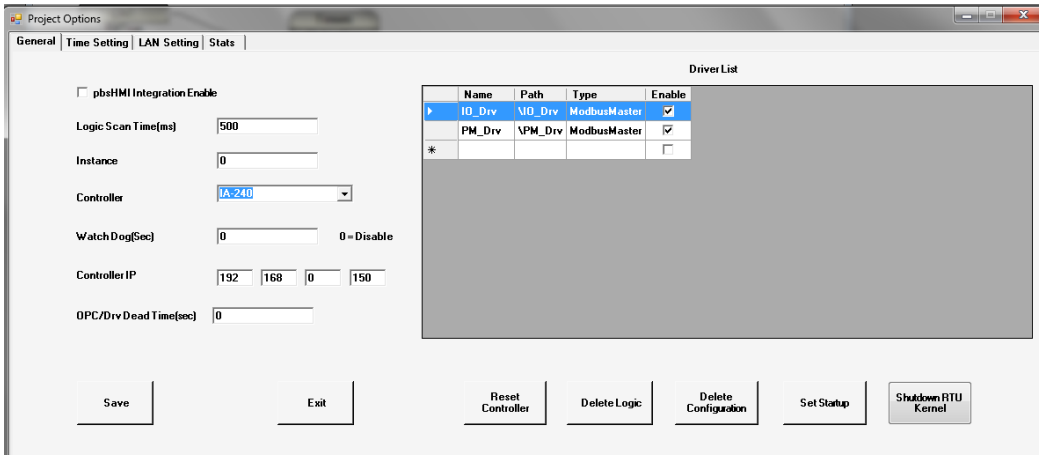
Configuration for Modbus Master Driver for Power Monitor Devices:



Click on Make Driver button. pbsSoftlogic will make separate directories with same name of Driver at logic path .



Following items are adding to Driver list in setting page:



Right click on IO_Drv and select explorer. pbsSoftLogic will open IO_Drv directory .

Three files are generated by pbsSoftlogic at this directory.

Options.xml : communication parameter . Like Serial Port, Baud rate …

ModbusBlocks.xml : Modbus Block Definitions

ModbusTags.xml : Modbus Tags Definitions

Edit options.xml file. You can set following parameters for ModbusMaster Driver. Each XML node has a name (Don't change it), Desc (Don't change it) and Value (Set based on Description)

```
<Node>
    <Name>PhysicalLayer</Name>
    <Desc>RS232 , RS485 , RS424 , TCP</Desc>
    <Value>RS232</Value>
</Node>
```

PhysicalLayer : For Modbus RTU Select one of RS232 , RS485 and RS422 . For ModbusTCP select TCP

```
<Node>
    <Name>COMPort</Name>
    <Desc>Serial Port for Communication 1,2,3,4,5,...</Desc>
    <Value>1</Value>
</Node>
```

COMPort :will be used for ModbusRTU protocol .

```
<Node>
    <Name>BaudRate</Name>
    <Desc>9600,19200,36400,52700,115200</Desc>
    <Value>9600</Value>
</Node>
```

BaudRate :will be used for ModbusRTU protocol .

```
<Node>
    <Name>DataBit</Name>
    <Desc>7,8</Desc>
    <Value>8</Value>
</Node>
```

DataBit :will be used for ModbusRTU protocol .

```
<Node>
    <Name>StopBit</Name>
    <Desc>1,2</Desc>
    <Value>1</Value>
</Node>
```

StopBit :will be used for ModbusRTU protocol .

```
<Node>
    <Name>Parity</Name>
    <Desc>None,Even,Odd</Desc>
    <Value>None</Value>
</Node>
```

Parity :will be used for ModbusRTU protocol .

```
<Node>
    <Name>Instance</Name>
    <Desc>Instance</Desc>
    <Value>1</Value>
</Node>
```

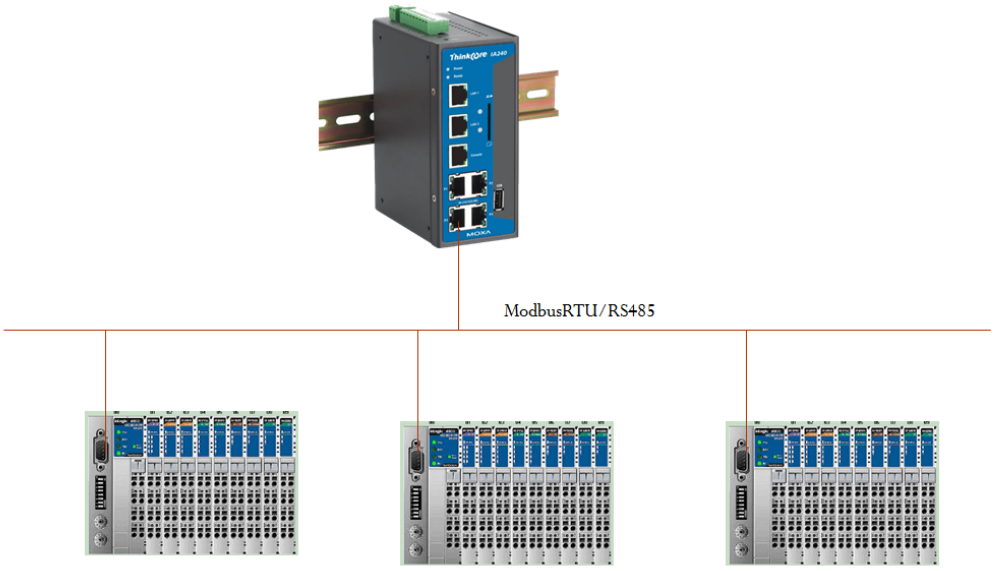
Instance: Driver Instance Number.

```
<Node>
    <Name>TCPPort</Name>
    <Desc>TCPPort</Desc>
    <Value>502</Value>
</Node>
```

TCPPort: ModbsuTCP Port number. Default Value 502

ModbusBlocks.xml : pbsSoftlogic Modbus modeling is based on Block Concept.

We start with a simple example to show concepts of Block. Suppose we want to configure Modbus network for following system:



ID = 1
DI = 64 (Bit 1 to 64)
DO = 32 (Bit 4096 to 4128)
AI = 8 (Word 10 to 17)

ID = 2
DI = 64 (Bit 1 to 64)
DO = 32 (Bit 4096 to 4128)
AI = 8 (Word 10 to 17)

ID = 3
DI = 64 (Bit 1 to 64)
DO = 32 (Bit 4096 to 4128)
AI = 8 (Word 10 to 17)

ModbusBlocks.xml for above configuration:

```
<?xml version="1.0"?>
<OPCSrvTags>
     <Version>1.0.0</Version>
     <Block Name="DI1" Type="BI" SlaveID="1" IP="" StartAdd="1" Count="64" Wait="200" Enable="True" />
     <Block Name="DO1" Type="BO" SlaveID="1" IP="" StartAdd="4096" Count="32" Wait="200" Enable="True" />
     <Block Name="AI1" Type="AI" SlaveID="1" IP="" StartAdd="10" Count="8" Wait="200" Enable="True" />
     <Block Name="Diag1" Type="SYS" SlaveID="1" IP="" StartAdd="100" Count="8" Wait="100" Enable="True" />

     <Block Name="DI2" Type="BI" SlaveID="2" IP="" StartAdd="1" Count="64" Wait="200" Enable="True" />
     <Block Name="DO2" Type="BO" SlaveID="2" IP="" StartAdd="4096" Count="32" Wait="200" Enable="True" />
     <Block Name="AI2" Type="AI" SlaveID="2" IP="" StartAdd="10" Count="8" Wait="200" Enable="True" />
     <Block Name="Diag2" Type="SYS" SlaveID="2" IP="" StartAdd="100" Count="8" Wait="100" Enable="True" />


     <Block Name="DI3" Type="BI" SlaveID="3" IP="" StartAdd="1" Count="64" Wait="200" Enable="True" />
     <Block Name="DO3" Type="BO" SlaveID="3" IP="" StartAdd="4096" Count="32" Wait="200" Enable="True" />
     <Block Name="AI3" Type="AI" SlaveID="3" IP="" StartAdd="10" Count="8" Wait="200" Enable="True" />
     <Block Name="Diag3" Type="SYS" SlaveID="3" IP="" StartAdd="100" Count="8" Wait="100" Enable="True" />

</OPCSrvTags>
```

Block Name = Unique name if Block.

Type: Block Type

BI = DI: Digital Input = Modbus Input status

BO= DO: Digital Output = Modbus Coil

BOS=DOS: Digital Output Status = Modbus Coils Status

AI: Analog Input = Modbus input Register

AO: Analog Output = Modbus Holding Register

AOS: Analog Output Status = Modbus Holding Register status

SYS: Internal for pbsSoftLogic . Can be used for reading status of communication.

SlaveID = ID of Slave Device.

IP = IP address of Slave Device. Will use for ModbusTCP network.

StartAddress = Start Address of Modbus Block . For Digital (Bit) and for analog (Word)

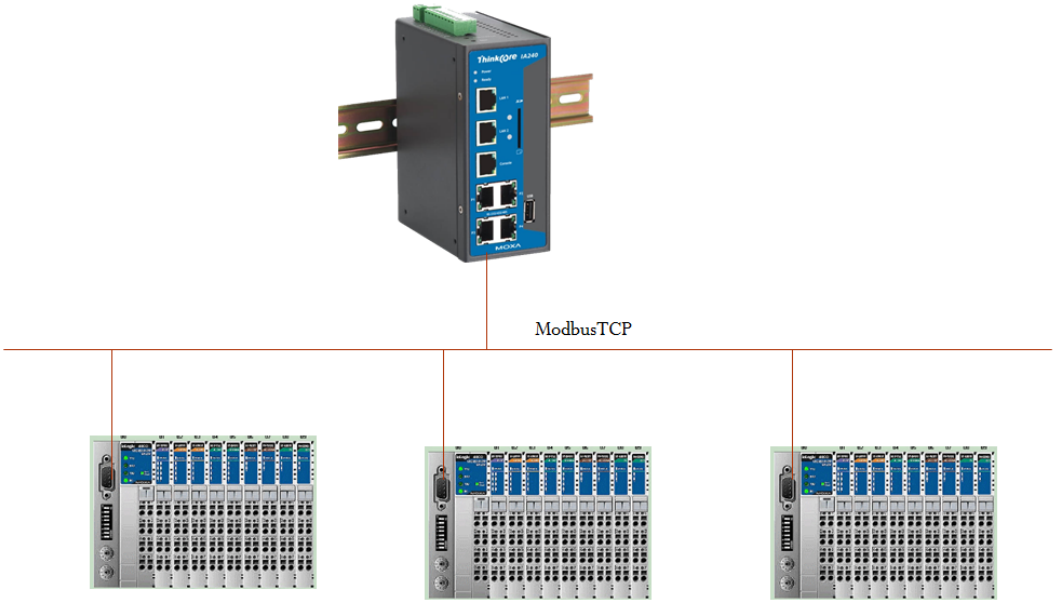
Count = Channel Count

Wait = Time for driver to wait for getting answer from Slave Device.

Enable = It is Enable or Not. If it is not enable, it is not polling by driver.

For SYS Block type, Start Address is dummy and it is not use by driver. So always put it 100. If you have another block with same start address, it is not making any conflict.

ModbusBlocks.xml file for ModbusTCP :



ID = 1
IP = 192.168.1.100
DI = 64 (Bit 1 to 64)
DO = 32 (Bit 4096 to 4128)
AI = 8 (Word 10 to 17)

ID = 1
IP = 192.168.1.101
DI = 64 (Bit 1 to 64)
DO = 32 (Bit 4096 to 4128)
AI = 8 (Word 10 to 17)

ID = 1
IP = 192.168.1.102
DI = 64 (Bit 1 to 64)
DO = 32 (Bit 4096 to 4128)
AI = 8 (Word 10 to 17)

ModbusBlocks.xml file for above configuration:

```
<?xml version="1.0"?>
<OPCSrvTags>
     <Version>1.0.0</Version>
     <Block Name="DI1" Type="BI" SlaveID="1" IP="192.168.1.100" StartAdd="1" Count="64" Wait="200" Enable="True" />
     <Block Name="DO1" Type="BO" SlaveID="1" IP="192.168.1.100" StartAdd="4096" Count="32" Wait="200" Enable="True" />
     <Block Name="AI1" Type="AI" SlaveID="1" IP="192.168.1.100" StartAdd="10" Count="8" Wait="200" Enable="True" />
     <Block Name="Diag1" Type="SYS" SlaveID="1" IP="192.168.1.100" StartAdd="100" Count="8" Wait="100" Enable="True" />

     <Block Name="DI2" Type="BI" SlaveID="1" IP="192.168.1.101" StartAdd="1" Count="64" Wait="200" Enable="True" />
     <Block Name="DO2" Type="BO" SlaveID="1" IP="192.168.1.101" StartAdd="4096" Count="32" Wait="200" Enable="True" />
     <Block Name="AI2" Type="AI" SlaveID="1" IP="192.168.1.101" StartAdd="10" Count="8" Wait="200" Enable="True" />
     <Block Name="Diag2" Type="SYS" SlaveID="1" IP="192.168.1.101" StartAdd="100" Count="8" Wait="100" Enable="True" />


     <Block Name="DI3" Type="BI" SlaveID="1" IP="192.168.1.102" StartAdd="1" Count="64" Wait="200" Enable="True" />
     <Block Name="DO3" Type="BO" SlaveID="1" IP="192.168.1.102" StartAdd="4096" Count="32" Wait="200" Enable="True" />
     <Block Name="AI3" Type="AI" SlaveID="1" IP="192.168.1.102" StartAdd="10" Count="8" Wait="200" Enable="True" />
     <Block Name="Diag3" Type="SYS" SlaveID="1" IP="192.168.1.102" StartAdd="100" Count="8" Wait="100" Enable="True" />

</OPCSrvTags>
```

Modbus Master Driver is polling Devices based on Modbus Block File. (For ModbusRTU and ModbusTCP)

For Above ModbusBlocks.xml file , Modbus Driver will do following sequence :

1- Send DI1 Block , Update Diag1 Send Counter
2- Wait for 200 msec
3- Get Answer and update Modbus Tags , Update Diag1 Rec Counter , Diag1.ErrorCounter = 0 , Diag1.Online = 1

4- If There is no answer from Device Increase Diag1.ErrorCounter , if it Is more than 3 , Make Device offline Diag1.Online = 0
5- Check Write Queue for Writing on DO or AO Blocks , If there is any item in Write Queue , Write it to Device otherwise send Request for Next Block
6- Send AI1 Block , Update Diag1.SendNum
7- Wait for 200 Msec
8- Get Answer and update Modbus Tags , Update Diag1 Rec Counter , Diag1.ErrorCounter = 0 , Diag1.Online = 1
9- If There is no answer from Device Increase Diag1.ErrorCounter , if it Is more than 3 , Make Device offline Diag1.Online = 0
10- Check Write Queue for Writing on DO or AO Blocks , If there is any item in Write Queue , Write it to Device otherwise send Request for Next Block
11- Repeat Steps 1 to 10 for Device 2.
12- Repeat Steps 1 to 10 for Device 3.

Scan Time Calculation: for above configuration Scan time for whole signals will be calculate as following:

200(DI1)+50+200(AI1)+50+

200(DI2)+50+200(AI2)+50+

200(DI3)+50+200(AI3)+50 = 3000 msec = 3 sec . (if there is no write command)

If you want to reduce scan time, you can increase BaudRate and reduce Block Wait time.

Or you can separate Modbus Network to two or three separate network.

ModbusTags.xml : All Modbus Tags will define in this file . FEEditor used this file for accessing tags .

Modbus Tag has following format In ModbusTags,xml file :

```
<Tag SlaveID="1" BlockName="DI1" Address="1" Name="DITag1" />
<Tag SlaveID="1" BlockName="DI1" Address="2" Name="DITag2" />
<Tag SlaveID="1" BlockName="DI1" Address="3" Name="DITag3" />
<Tag SlaveID="1" BlockName="DI1" Address="4" Name="DITag4" />
<Tag SlaveID="1" BlockName="DI1" Address="5" Name="DITag5" />
<Tag SlaveID="1" BlockName="DI1" Address="6" Name="DITag6" />
<Tag SlaveID="1" BlockName="DI1" Address="7" Name="DITag7" />
<Tag SlaveID="1" BlockName="DI1" Address="8" Name="DITag8" />
```

SlaveID : Slave ID . You need to put it SlaveID here for faster access of FbEditor .

Blockname : Same Block name in ModbusBlocks.xml

Address = Modbus Tag Address. Start from 0. No need to Write like Modbus Format (like 10001) . Just write address of Tag .

Name = Modbus Tag Name . SlaveID+ Tag Name should be unique for Modbus Master Driver.

For All other salve drivers (Modbus, DNP3 and IEC8705) Tag Name should be unique .

For Diag Block you need to define following tags :

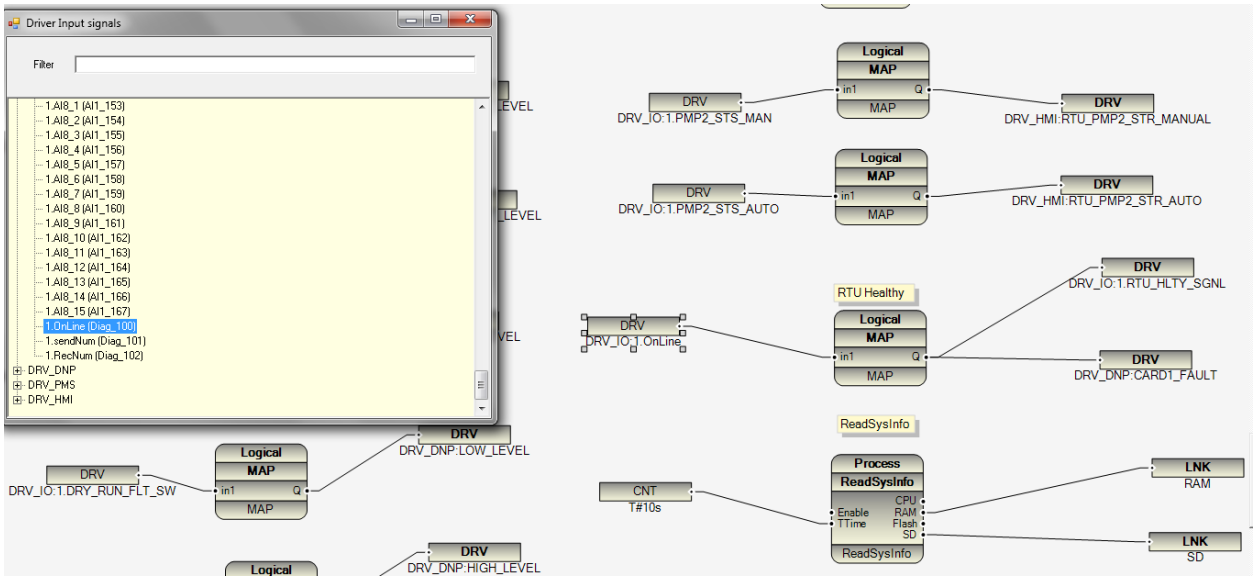
```
<Tag SlaveID="1" BlockName="Diag" Address="100" Name="OnLine" />
<Tag SlaveID="1" BlockName="Diag" Address="101" Name="sendNum" />
<Tag SlaveID="1" BlockName="Diag" Address="102" Name="RecNum" />
```

First tag is Online . If device is answer to Driver request its value is 1 otherwise it is 0.

sendNum : Number of Send Request by driver . Maximum value is 10,000

RecNum : Number of received Answer to driver . Maximum value is 10,000

You can use above tags like normal Modbus Tag in your logic.



Number of Modbus Master Driver for each controller : 8 Instance

Number of Modbus tags for each Instance: 1024

Number of Modbus Blocks for each instance: 64

Number of Modbus Devcices for each instance: 32

7- Modbus Slave Configuration

pbssoftLogic supports Modbus slave Driver for communication with HMI Devices or any other Modbus Master systems .

You can run Modbus master and salve on the same Controller in the same time but they should have separate resource. For example COM Port 1 can be Modbus Master and COM Port 2 Modbus Slave.

There is Software limitation for number of Instances for any protocol in pbsSoftLogic (maximum 8) . You can run 8 instances of Modbus Slave on the same Controller and connect to different modbus master in the same time.

For each instance of Modbus slave Driver we have following tables:

Digital inputs Tags: 1024

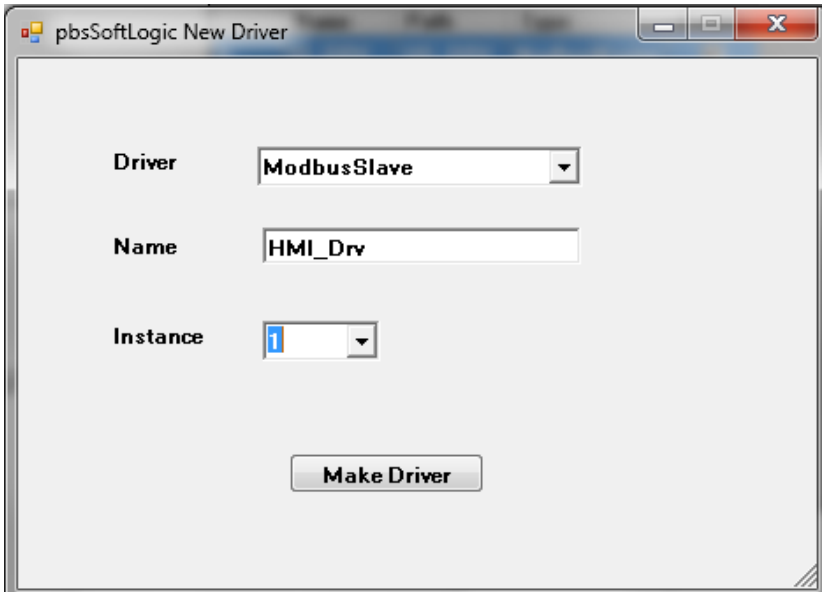
Digital Output Tags: 1024

Analog input Tags: 1024

Analog Output Tags: 1024

For Adding Modbus Slave Driver to an Application, open Project settings and right click on Driver list .

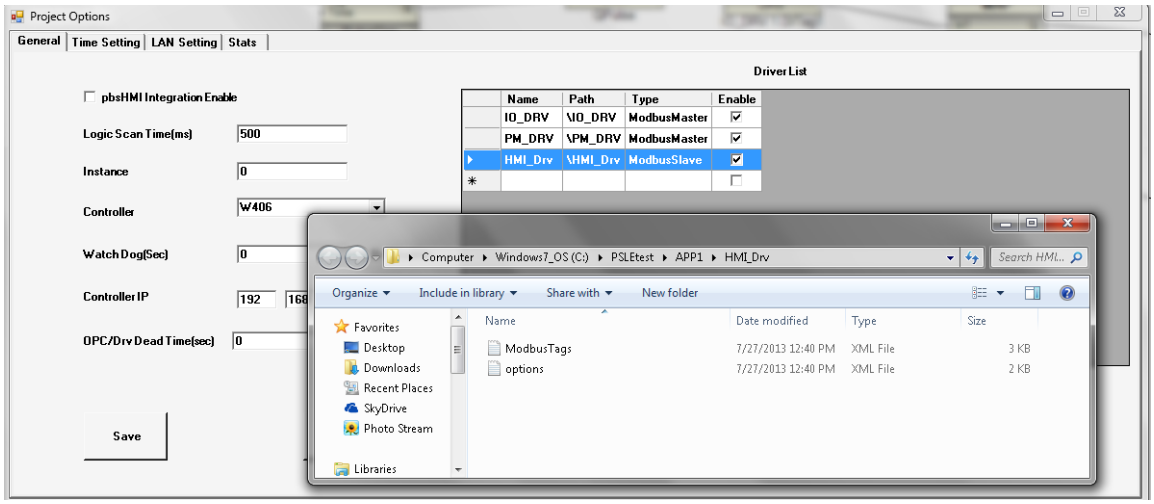
Select ModbusSlave Driver and fill other fields.



Click on make Driver Button. pbsSoftlogic will make basic files for Modbus Slave Communication .

Close this page, Modbus Slave Driver is added to Driver list.

Right click on Modbus Slave Driver and select explorer. You can see two files in HMI_Drv directory.



Options.xml : communication basic parameter

ModbusTags.xml : Modbus Slave Tags

```
<Node>
    <Name>PhysicalLayer</Name>
    <Desc>RS232 , RS485 , RS422 ,TCP</Desc>
    <Value>RS232</Value>
</Node>
```

PhysicalLayer : Physical layer . for Modbus RTU select one of RS232, RS485 or RS422 for ModbusTCP Select TCP

```
<Node>
    <Name>Protocol</Name>
    <Desc>RTU,ASCII</Desc>
    <Value>RTU</Value>
</Node>
```

Protocol : Modbus RTU or ASCII . This version supports RTU Only.

```
<Node>
    <Name>COMPort</Name>
    <Desc>Serial Port for Communication 1,2,3,4,5,...</Desc>
    <Value>1</Value>
```

```
</Node>
```

COMPort : Serial Com Port for ModbsuRTU

```
<Node>
    <Name>BaudRate</Name>
    <Desc>9600,19200,36400,52700,115200</Desc>
    <Value>9600</Value>
</Node>
```

BaudRate : Modbus RTU Baudate for communication .

```
<Node>
    <Name>DataBit</Name>
    <Desc>7,8</Desc>
    <Value>8</Value>
</Node>
```

DataBit : ModbusRTU Data Bits . 7 or 8

```
<Node>
    <Name>StopBit</Name>
    <Desc>1,2</Desc>
    <Value>1</Value>
</Node>
```

StopBit : ModbusRTU Stop Bit .

```
<Node>
    <Name>Parity</Name>
    <Desc>None,Even,Odd</Desc>
    <Value>None</Value>
</Node>
```

Parity : Modbus RTU Parity Communication

```
<Node>
    <Name>SlaveAddress</Name>
    <Desc>SlaveAddress</Desc>
    <Value>3</Value>
</Node>
```

SlaveAddress: Modbus RTU/TCP slave ID

```
<Node>
  <Name>FlowControl</Name>
  <Desc>NO_FLOW_CONTROL,HW_FLOW_CONTROL,SW_FLOW_CONTROL</Desc>
  <Value>NO_FLOW_CONTROL</Value>
</Node>
```

FlowControl: Flow Control for ModbusRTU

```
<Node>
  <Name>PhysicalLayerScanTime</Name>
  <Desc>PhysicalLayerScanTime</Desc>
  <Value>100</Value>
</Node>
```

PhysicalLayerScanTime : Modbus Slave Driver will read Serial or TCP port every PhysicalLayerScanTime msec . if master request is large (like Writing many Modbus Signals , it is better to increase this value . 100 msec is optimized for may applications .

```
<Node>
  <Name>Instance</Name>
  <Desc>Instance</Desc>
  <Value>1</Value>
</Node>
```

Instance : If you have many ModbusSlave Driver on a controller , each one must has unique Instance number .(maximum 8)

```
<Node>
  <Name>TCPPort</Name>
  <Desc>TCPPort</Desc>
  <Value>502</Value>
</Node>
```

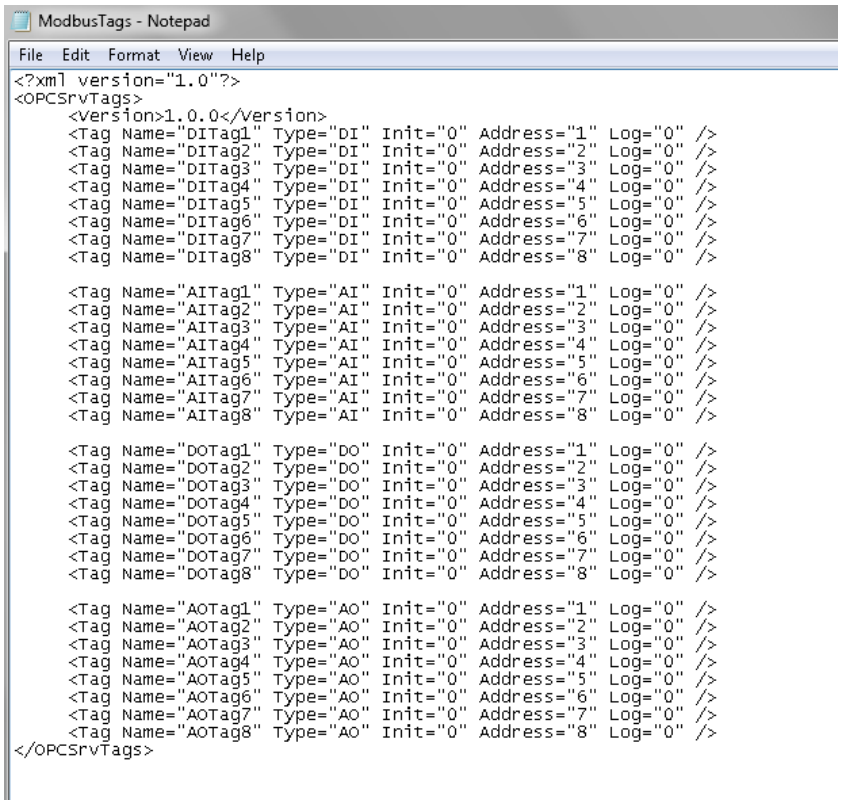
TCPPort : ModbusTCP Communication port . Default value is 502

```
<Node>
  <Name>ShiftAddress</Name>
  <Desc>ShiftAddress</Desc>
  <Value>0</Value>
```

</Node>

Shift Address : this value with add to all Modbus Slave Address that is request from master .

Modbustags.xml file: in following figure you can see typical Modbus Slave Tags that is generate by pbsSoftLogic when you make a new Modbus Slave Driver.



Each Modbus Tag has following properties:

Name: Unique Modbus Tag Name. pbsSoftLogic will read this names and you can use Tags name in your logic .

Type: Tag Type (all Input Types must be writing in logic and all Output types must read in logic)

Input Types :

DI: Digital input.

AI : Analog input

FI : Floating point Input . In AI Space , will take 2 Address (Register)

INTI : Long input . In AI Space , will take 2 Address (Register)

INTUI : unsigned long . In AI Space , will take 2 Address (Register)

SFI : Swap Floating point Input . In AI Space , will take 2 Address (Register)

SINTI : Swap Long input . In AI Space , will take 2 Address (Register)

SINTUI : Swap unsigned long . In AI Space , will take 2 Address (Register)

Output Types :

DO: Digital Output.

AO : Analog Output

FO : Floating point Output. In AO Space , will take 2 Address (Register)

INTO : Long Output. In AO Space , will take 2 Address (Register)

INTUO : unsigned long . In AO Space , will take 2 Address (Register)

SFO : Swap Floating point Output. In AO Space , will take 2 Address (Register)

SINTO : Swap Long Output. In AO Space , will take 2 Address (Register)

SINTUO : Swap unsigned long . In AO Space , will take 2 Address (Register)

Init: init value of Modbus Slave Tag

Address: Modbus Slave Tag Address.

Log : If Log value is 1 , Driver will always used latest value of Modbus tag not Init Value . Suppose you define a set point with init value of 10 . If Modbus Master change this value to 12.0 and you restart controller, Modbus Slave Driver will use 12 as init value of Tag .

Note 1 :This facility is just works for AO , DO and FO Tags . (Modbus Slave Output tgs)

Note 2 : Runtime kernel in Controller will check every min for Modbus Slave changes and will copy changes to internal flash memory . so if you change set points by Modbus master and restart controller before one min pass , then controller is not keeping last value of set points .

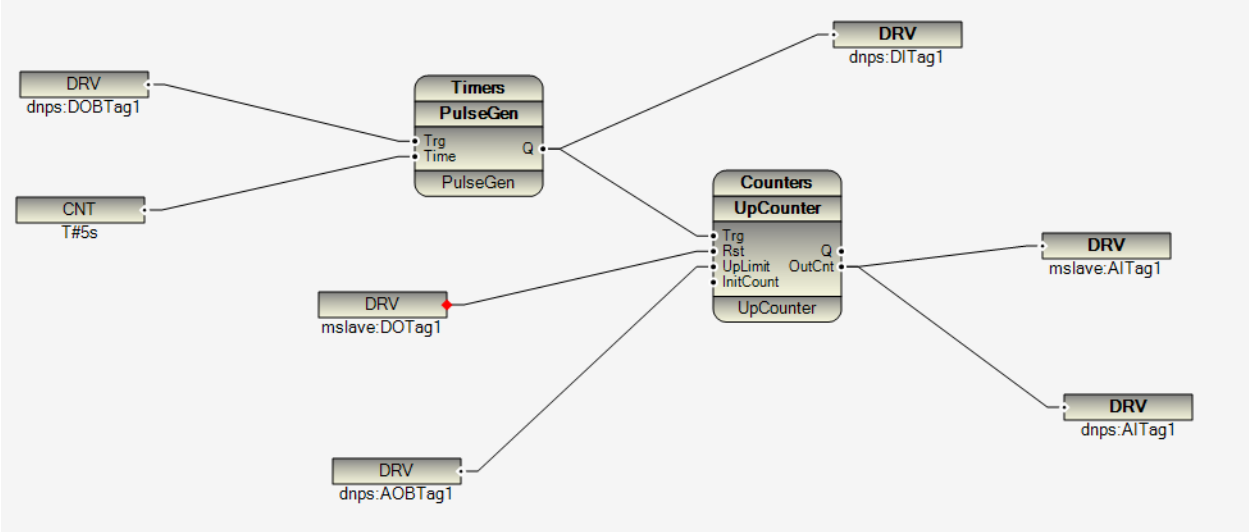
Modbus Slave Driver operation:

1-Modbus master is reading all Input Tags (DI, AI, FI,…) by polling.

You should write on all Modbus Slave Input Signal on your logic . (Connect to FB output ports)

2 - Modbus master is writing all output signals (DO, AO, FO, ..) .

You should read output tags in your logic. (Connect to FB input ports)



In above sample logic mslave:DOTag1 is an output signal from Modbus master (Linked to FB input port) and mslave:AiTag1 is an input signal to modbus master (Linked to FB output ports)

8 – DNP3 Slave Configuration

pbsSoftlogic supports DNP3 slave driver for Linux/WinCE controllers . Please refer to www.dnp.org web site for detail information about DNP3 protocol.

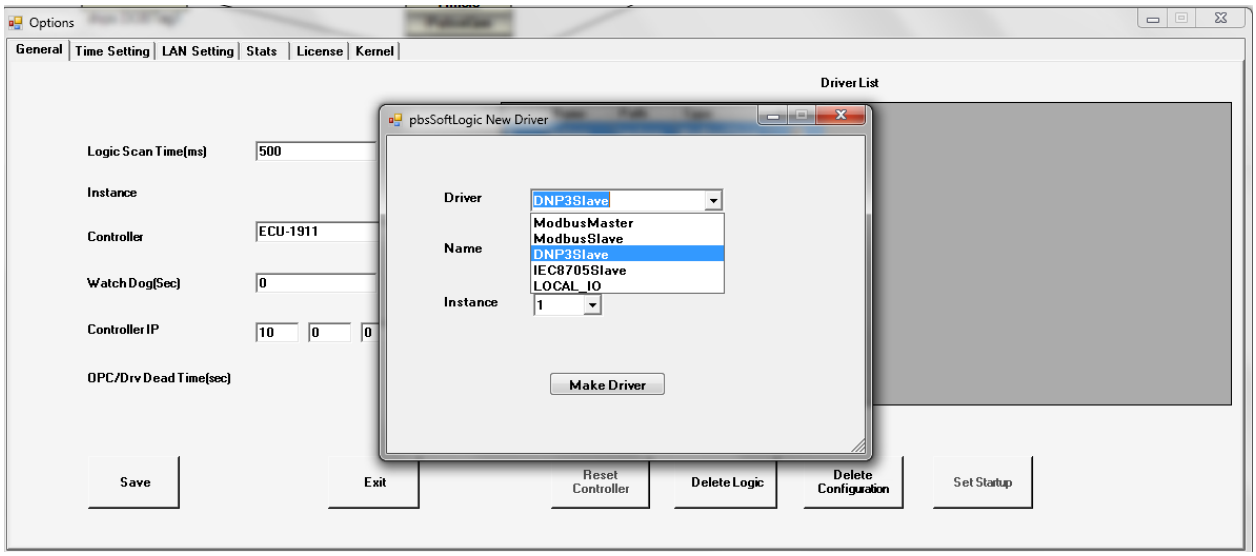
You can define up to 4 dnp3 slave instances for a controller. Each DNP3 slave instance can be connected to separate DNP3 master SCADA.

At each instance you can define 1024 DNP tags .

As physical layer you can select RS232 and TCP/IP.

Defining new DNP3 slave driver:

- Open project setting
- Right click on driver list
- Select New Driver
- Select DNP3Slave as Driver type
- Type a unique name for Driver name
- Select unique Instance for driver



- Click on make driver button .

pbsSoftLogic will make option file and DNP3 Slave tags files and will make a new directory with the same name of Driver name in logic path .

- Options.xml define communication parameters
- DNP3Tags.xml define dnp3 tags

Options.xml parameters:

```
<Node>
    <Name>PhysicalLayer</Name>
    <Desc>RS232 , TCP</Desc>
    <Value>TCP</Value>
</Node>
```

You can select physical layer between RS232 and TCP.

```
<Node>
    <Name>COMPort</Name>
    <Desc>Serial Port for Communication 1,2,3,4,5,...</Desc>
    <Value>2</Value>
</Node>
```

Controller Serial port for RS232 Communication.

```
<Node>
    <Name>BaudRate</Name>
    <Desc>9600,19200,36400,52700,115200</Desc>
    <Value>19200</Value>
</Node>
```

Communication baud rate

```
<Node>
    <Name>SlaveAddress</Name>
    <Desc>SlaveAddress</Desc>
    <Value>3</Value>
</Node>
```

RTU DNP3 Address

```
<Node>
    <Name>MasterIPAddress</Name>
    <Desc>MasterIPAddress</Desc>
    <Value>10.0.0.11</Value>
</Node>
```

DNP3 master SCADA IP address

```
<Node>
    <Name>TCPIPPort</Name>
    <Desc>TCPIPPort</Desc>
    <Value>20000</Value>
</Node>
```

TCP Port for using in TCP Connection , by default it is 20000

```
<Node>
    <Name>MasterAddress</Name>
    <Desc>MasterAddress</Desc>
    <Value>1</Value>
</Node>
```

DNP3 Master SCADA Address

```
<Node>
    <Name>LocalIPAddress</Name>
    <Desc>LocalIPAddress</Desc>
    <Value>10.0.0.10</Value>
</Node>
```

Controller LAN Port for communication with master SCADA

<Node>

<Name>PhysicalLayerScanTime</Name>

<Desc>PhysicalLayerScanTime</Desc>

<Value>100</Value>

</Node>

<Node>

<Name>Instance</Name>

<Desc>Instance</Desc>

<Value>1</Value>

</Node>

Driver instance number 1 ,2,3,4

<Node>

<Name>TCPIPMode</Name>

<Desc>0 = TCP Listening End Point , 1= UDP endpoint , 2 = TCP Dual End Point</Desc>

<Value>0</Value>

</Node>

<Node>

<Name>AppFrameSize</Name>

<Desc>AppFrameSize</Desc>

<Value>2000</Value>

</Node>

<Node>

<Name>SBOTimeOut</Name>

<Desc>SBOTimeOut(Sec)</Desc>

<Value>10</Value>

</Node>

Select before Operate delay

<Node>

<Name>NoCommTimeout</Name>

<Desc>NoCommTimeout(Sec)</Desc>

<Value>0</Value>

</Node>

Time that RTU is checking communication, if there is no any communication in this period, RTU will close connection in TCP Mode. 0 means communication checking is disabling. Unit is in second.

DNP3Tags.xml

When you make a new driver, pbsSoftLogic will make a default DNP3 Tags file. You can edit this file and add or remove tags.

Name: Tag Name. It should be unique in your Logic.

Type: DNP3 Tag Type. We support following types:

- DI : Digital input Read By Master with different variations , DNP Group1 , 2
- AI : Analog input Read By Master with different variations , DNP Group 30,31,32,33
- CNT : Counter Read By Master with different variations DNP Group 20,21,22,23
- FI : Float Input : DNP Group 100
- DOB : Digital Output Block Write by master with different mode DNP Group 12 ,13
- AOB : Analog Output Block Write by master with different mode , DNP Group 41
- DO : DO Status Read By Master with different variations , DNP Group 10,11
- AO : AO Status Read By Master with different variations , DNP Group 40
- DPI : Double Bit Binary Read By Master with different variations , DNP Group 3,4

Class : Based on DNP3 Standard we have class 0 ,1,2,3,4

Class 0 means current value of tags without event buffering . So if you put class 0 for a tag, RTU is not buffering tag changes and every time master read tag , RTU will send current value .

Class 1,2,3,4 there is no different or priority between different classes. So if you put class 1,2,3 or 4 for a tag RTU will buffer all tag changes with time and will report to Master SCADA .

There is a cyclic buffer with 10,000 events for each DNP Type in RTU.

Address: DNP3 tag address. AI and FI are using same address range.

Log : When set to 1 for DOB and AOB Tags , RTU will keep last value of Set Point in internal memory flash and if you restart RTU , it will use latest set points from Master SCADA . RTU will check AOB and DOB changes every min and if it detect changes, it will save them on internal flash memory.

Init : Init Value of a tag .

```
<Tag Name="AITag7" Type="AI" Class="1" Init="0" Address="7" Log="0" />
<Tag Name="AITag8" Type="AI" Class="1" Init="0" Address="8" Log="0" />
<Tag Name="CNTTag1" Type="CNT" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="CNTTag2" Type="CNT" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="CNTTag3" Type="CNT" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="CNTTag4" Type="CNT" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="CNTTag5" Type="CNT" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="CNTTag6" Type="CNT" Class="1" Init="0" Address="6" Log="0" />
```

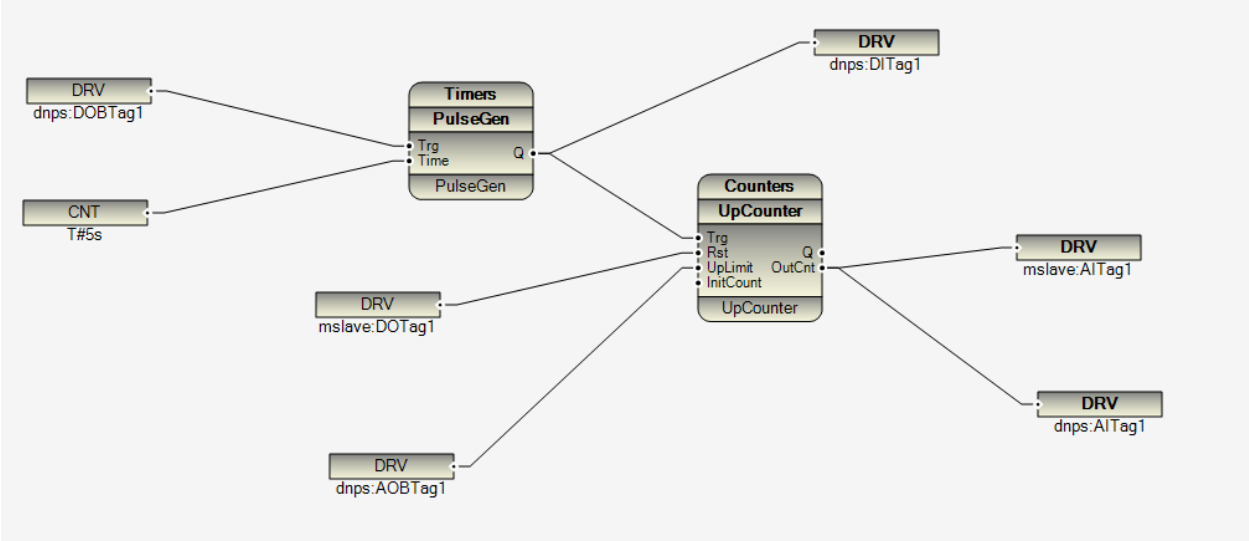
DNP3 Slave driver Operation:

1 - Master SCADA will read all Input Signals (DI , AI , FI , DO , AO , DPI)

- You need to write all Input Signals in your logic.(Link to FB right ports)

2 – Master SCADA will write Output Signals (DOB , AOB)

- You need to read all Output Tags in your logic (Link to FB left Ports)



In above logic we have following DNPs signals:

- Dnps: DOBTag1 is a DOB signal which is written by DNP Master.
- dnps:DITag1 is a Di signal which is read by DNP Master
- dnps:AOBTag1 : AOB signal (Analog Output) which is written by DNP Master
- dnps:AITag1 : AI (Analog input) Signal which is read by DNP Master

DNP3 function codes which are implemented:

- Read class 0,1,2,3,4
- Integrity command
- Read Event by exception (RBE)
- Time synchronization
- Enable /Disable unsolicited communications (Transfer data from RTU to Master SCADA)
- Dynamic Class assign
- Freezing counters
- Write

9 – IEC870-5 Slave (101-104) Configuration

pbsSoftLogic supports IEC870-5-101/104 protocols for communication with master SCADA .

You can setup maximum four IEC slave instance for each RTU. It means you can connect to 4 Separate SCADA master in the same time.

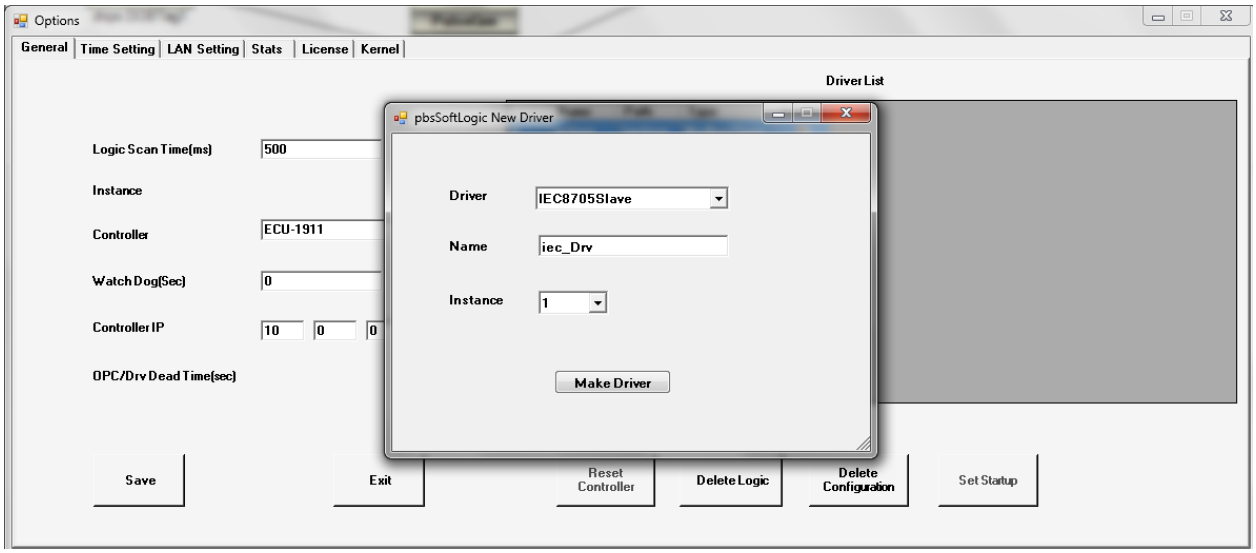
IEC870-5-101 is communicating over RS232 and IEC870-5-104 is communicating over TCP.

For each IEC Driver instance you can define 1024 IEC Tags.

Defining new IEC Driver:

Open project setting and right click on driver list. Select new driver and then select IEC8705Slave .

Type Deriver name and select instance as following figure.



pbsSoftlogic will make a default configuration and IEC tags in a directory located at logic path . Directory name is name of driver.

IEC870-5 driver files:

- Options.xml define communication parameters
- IECSTags.xml define IEC slave tags

Communication parameters : optione.xml file content :

```
<Node>
    <Name>PhysicalLayer</Name>
    <Desc>RS232 , TCP</Desc>
    <Value>RS232</Value>
</Node>
<Node>
    <Name>COMPort</Name>
    <Desc>Serial Port for Communication 1,2,3,4,5,...</Desc>
    <Value>1</Value>
</Node>
<Node>
    <Name>BaudRate</Name>
    <Desc>9600,19200,36400,52700,115200</Desc>
    <Value>19200</Value>
</Node>
<Node>
    <Name>SlaveAddress</Name>
    <Desc>SlaveAddress</Desc>
    <Value>3</Value>
</Node>
```

```
<Node>
    <Name>MasterIPAddress</Name>
    <Desc>MasterIPAddress</Desc>
    <Value>127.0.0.1</Value>
</Node>
<Node>
    <Name>TCPIPPort</Name>
    <Desc>TCPIPPort</Desc>
    <Value>2404</Value>
</Node>
<Node>
    <Name>MasterAddress</Name>
    <Desc>MasterAddress</Desc>
    <Value>1</Value>
</Node>
<Node>
    <Name>LocalIPAddress</Name>
    <Desc>LocalIPAddress</Desc>
    <Value>127.0.0.1</Value>
</Node>
<Node>
    <Name>PhysicalLayerScanTime</Name>
    <Desc>PhysicalLayerScanTime</Desc>
    <Value>100</Value>
</Node>
```

```
<Node>
    <Name>Instance</Name>
    <Desc>Instance</Desc>
    <Value>1</Value>
</Node>
<Node>
    <Name>COTZ</Name>
    <Desc>Cause of Transmition Size 1,2 </Desc>
    <Value>1</Value>
</Node>
<Node>
    <Name>CAOAZ</Name>
    <Desc>Common Address of ASDU Size 1,2 </Desc>
    <Value>1</Value>
</Node>
<Node>
    <Name>IOZ</Name>
    <Desc>Information Object Size Size 1,2,3</Desc>
    <Value>1</Value>
</Node>
<Node>
    <Name>MODE</Name>
    <Desc>Communication Mode Balance(B) , Unbalan(U)  </Desc>
    <Value>B</Value>
</Node>
```

<Node>

<Name>KParam</Name>

<Desc>KParameter 1~ 32767 max difference recive sequence number to send state variable</Desc>

<Value>12</Value>

</Node>

<Node>

<Name>WParam</Name>

<Desc>WParameter 1~ 32767 Latest ACK after reciving W I-format APDUs</Desc>

<Value>8</Value>

</Node>

<Node>

<Name>T0Param</Name>

<Desc>T0Parameter Timeout of Connection establishment(sec)</Desc>

<Value>30</Value>

</Node>

<Node>

<Name>T1Param</Name>

<Desc>T1Parameter Timeout of Send test APDU(sec)</Desc>

<Value>15</Value>

</Node>

<Node>

<Name>T2Param</Name>

<Desc>T2Parameter Timeout for ACK in case of no data message (sec)</Desc>

<Value>10</Value>

</Node>

```
<Node>
   <Name>T3Param</Name>
   <Desc>T3Parameter  Timeout for sending test frames in case of a long idle state  (sec)</Desc>
   <Value>20</Value>
</Node>
```

IEC Slave Tag file: IECSTags.xml

Name: Tag Name. Should be unique in your logic

Type: IEC Tags type. Following type is supported:

- DI (Digital input) IEC Tag Type 1 ,30, M_SP_NA_1
- AI (Analog Input) IEC Tag Type 9,34,M_ME_NA_1 ,M_ME_TD_1
- FI(Float Input) IEC Tag Type 13 ,36 M_ME_NC_1 ,M_ME_TF_1
- CNT (Counter) IEC Tag Type 15 , 37 M_IT_NA_1,M_IT_TB_1
- DPI (Double Point Information) IEC Tag Type 3,4 ,M_DP_NA_1,M_DP_TA_1
- DO (Digital Output) IEC Tag Type 45 , C_SC_NA_1
- AO (Analog Output) IEC Tag Type 48 , C_SE_NA_1
- FO (Float Output) IEC Tag Type 50 ,C_SE_NC_1
- DPO(Double command) IEC Tag Type 46 , C_DC_NA_1

- **Process information in monitor direction**
- <1> := Single-point information (M_SP_NA_1)
- <3> := double-point information (M_DP_NA_1)
- <4> := double-point information with time tag (M_DP_TA_1)
- <9> := Measured value, normalized value (M_ME_NA_1)
- <13> := Measured value, short floating point value (M_ME_NC_1)
- <15> := Integrated totals (M_IT_NA_1)
- <21> := Measured value, normalized value without quality descriptor (M_ME_ND_1)
- <30> := Single-point information with time tag CP56Time2a (M_SP_TB_1)
- <34> := Measured value, normalized value with time tag CP56Time2a(M_ME_TD_1)
- <36> := Measured value, short floating point value with time tag CP56Time2a (M_ME_TF_1)
- <37> := Integrated totals with time tag CP56Time2a (M_IT_TB_1)

Process information in control direction

<45> := Single command (C_SC_NA_1)

<46> := double command (C_DC_NA_1)

<48> := Set point command, normalized value (C_SE_NA_1)

<50> := Set point command, short floating point value (C_SE_NC_1)

System information in monitor direction

<70> := End of initialization (M_EI_NA_1)

System information in control direction

<100>:= Interrogation command (C_IC_NA_1)

<101>:= Counter interrogation command (C_CI_NA_1)

<103>:= Clock synchronization command (C_CS_NA_1)

Basic application functions

Station initialization

Cyclic data transmission

Spontaneous transmission

Global Station interrogation

Clock synchronization

Command transmission

- Direct command transmission
- Direct set point command transmission
- Select and execute command
- Select and execute set point command
- **Transmission of integrated totals**
- Mode B: Local freeze with counter interrogation
- Counter read
- Counter freeze without reset
- Counter freeze with reset
- Counter reset
- General request counter

Class : IEC Supported two classes , Class1 and Class2 .

From IEC870-5-101 standard:

The polling procedure is supported by the link layer which requests user data of classes 1 and 2. In general, ASDUs containing the causes of transmission periodic/cyclic are assigned to be transmitted with the link layer data class 2 and all time tagged or spontaneously transmitted ASDUs are assigned to be transmitted with the link layer data class 1. Other ASDUs with other causes of transmission of low priority such as background scan may also be assigned to data class 2 and must be listed in the interoperability document.
In this case, it has to be considered that the link request of class 1 occurs at a different point of time (to or from) the link request of class 2, which may influence the correct sequence of the ASDUs delivered to the application layer of the controlling station.
In response to a class 2 poll, a controlled station may respond with class 1 data when there is no class 2 data available.

Init : IEC Tag Init Value

Address : IEC Tag Address

: Log : When set to 1 for DO , AO ,FO and DPO Tags , RTU will keep last value of Set Point in internal memory flash and if you restart RTU , it will use latest set points from Master SCADA . RTU will check AO , DO , FO and DPO changes every min and if it detect changes, it will save them on internal flash memory.

```
<Tag Name="FITag6" Type="FI" Class="1" Init="0" Address="6" Log="0" />
<Tag Name="FITag7" Type="FI" Class="1" Init="0" Address="7" Log="0" />
<Tag Name="FITag8" Type="FI" Class="1" Init="0" Address="8" Log="0" />
<Tag Name="CNTTag1" Type="CNT" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="CNTTag2" Type="CNT" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="CNTTag3" Type="CNT" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="CNTTag4" Type="CNT" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="CNTTag5" Type="CNT" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="CNTTag6" Type="CNT" Class="1" Init="0" Address="6" Log="0" />
<Tag Name="CNTTag7" Type="CNT" Class="1" Init="0" Address="7" Log="0" />
<Tag Name="CNTTag8" Type="CNT" Class="1" Init="0" Address="8" Log="0" />
<Tag Name="DPITag1" Type="DPI" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="DPITag2" Type="DPI" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="DPITag3" Type="DPI" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="DPITag4" Type="DPI" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="DPITag5" Type="DPI" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="DPITag6" Type="DPI" Class="1" Init="0" Address="6" Log="0" />
<Tag Name="DPITag7" Type="DPI" Class="1" Init="0" Address="7" Log="0" />
<Tag Name="DPITag8" Type="DPI" Class="1" Init="0" Address="8" Log="0" />
<Tag Name="DOTag1" Type="DO" Class="1" Init="0" Address="1" Log="0" />
```

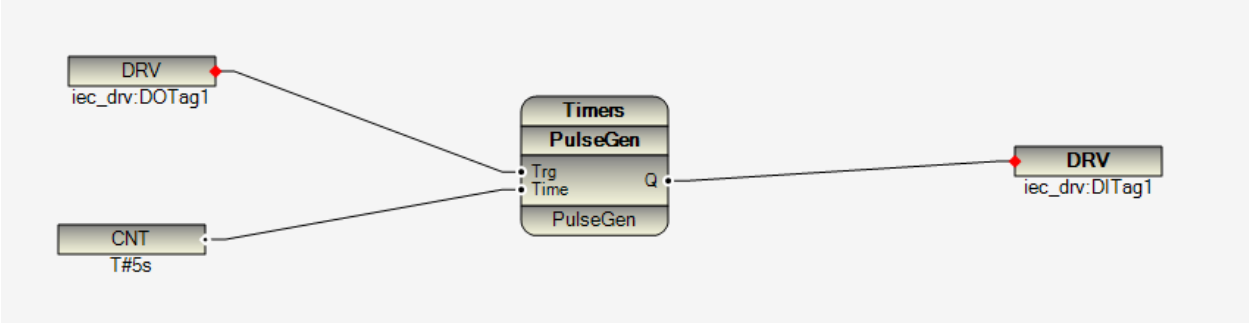
IEC 870-5 Slave driver Operation:

1 - Master SCADA will read all Input Signals (DI , AI , FI , DPI ,CNT)

- You need to write all Input Signals in your logic.(Link to FB right ports)

2 – Master SCADA will write Output Signals (DO , AO,FO ,DPO)

- You need to read all Output Tags in your logic (Link to FB left Ports)



In above logic master will write to iec_drv:DOTag1 and will read iec_drv:DITag1

10 - User defined function block

pbsSoftLogic has open structure for adding new Function block by user to platform .

User defined FB (UDF) has the same performance as internal FB in pbsSoftLogic.

There are two ways for adding new FB to pbsSoftLogic:

1 – Using C# for simulator and Windows Runtime and GCC Cross compiler for embedded linux

2 – Using Lua scripting language.

In this section, we will describe both solutions.

First solution is difficult and you need to develop C# and C code for your FBand compile it by GCC under Linux OS.

Second solution is very simple and effective and no need to do any cross compiling for Linux kernel.

Lua – www.Lua.org- is one the most famous scripting language in the market and it is used in many projects and applications world wised.

pbsSoftLogic Linux runtime engine supports Lua Ver 5.2.2 which is latest version .

For learning Lua language , please refer to www.lua.org web site .

10 – 1 C# and C UDF development

Three steps are required for adding UDF to pbsSoftLogic:

1- Defining FB Input / Output structure
2- Writing C# code for simulator and Windows Kernel
3- Writing C Code for Linux Kernel and Cross compiling for embedded linux .

C# and C Source code of all internal FB are included in pbsSoftLogic. You can use these source codes to make new FB and expand platform.

For Editing and compiling C# Codes, pbsSofLogic has integrated professional C# editor and compiler, so you don't need to use other IDE or compiler for Writing C# code.

For editing and compiling C Codes, you need to use linux Operating system, MOXA Cross compilers for different Controllers and Eclipse IDE.

In this section, we will describe details of above steps with implementation of a simple UDF.

UDF is not related to a specific project, but it will include to platform.

You can find C# Source code of all internal FB at \PSLE\VSLESrc Directory.

For defining new UDF, you need to define new FB Group. FB Group includes many FBs.

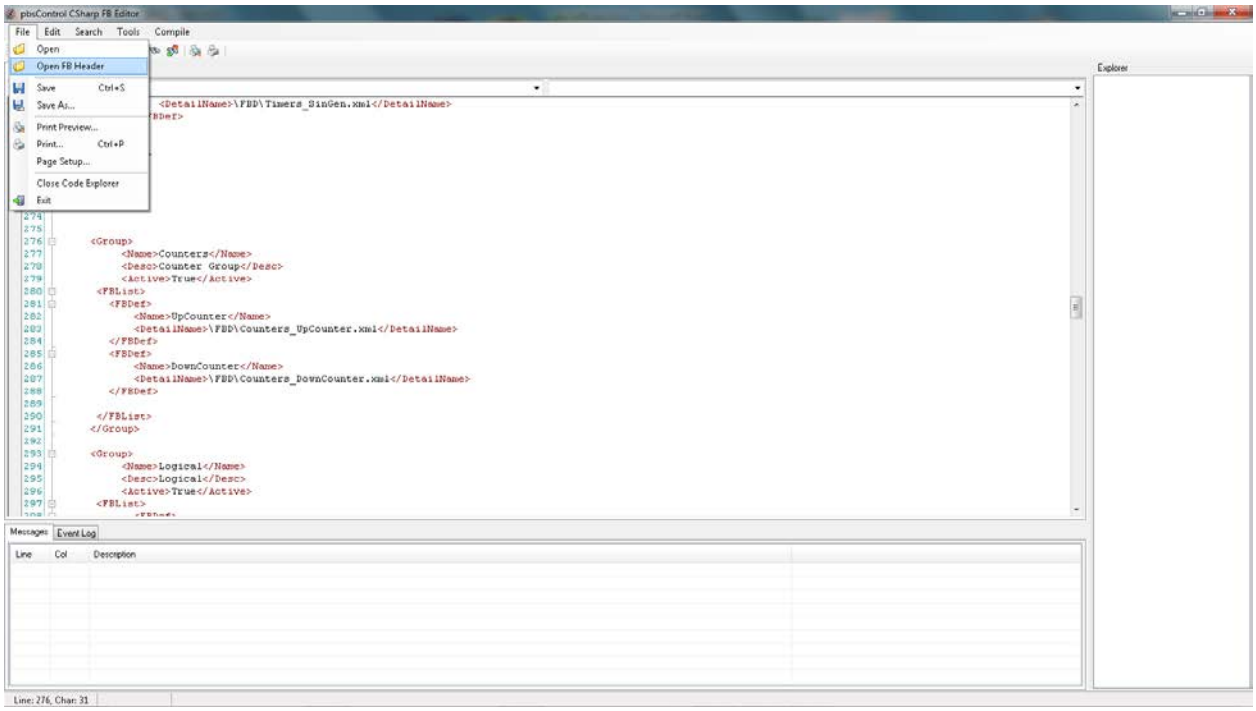
Suppose we want to define a new FB Groups for IEC1131-3 standard and add two Function Block for RS and SR Flip Flop. In following figure you can see the definition of RS and SR flip flop from IEC1131-3 standard.


```
FUNCTION_BLOCK      SR          (* flip flop set dominant *)
  VAR_INPUT
    S1    :    BOOL;
    R     :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q1    :    BOOL;
  END_VAR
  Q1      :=   S1 OR ( NOT R AND Q1);
END_FUNCTION_BLOCK

FUNCTION_BLOCK      RS          (* flip flop reset dominant *)
  VAR_INPUT
    S     :    BOOL;
    R1    :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q1    :    BOOL;
  END_VAR
  Q1      :=   NOT R1 AND ( S OR Q1);
END_FUNCTION_BLOCK
```

Step1: Define FB body. Edit FBDefh.xml file in \PSLE\cfg directory.

Run Windows FB Editor Utility from Tools menu in pbsSoftLogic Editor. FB editor can be use for defining new FB body and CSharp implementation for Simulator and Windows Runtime Kernel.

Open FB header file from File menu and select “open FB Header”. It will open FBDefh.xml file.



FBDefh.xml file contains all pbsSoftLogic FB header (internal and UDF).

For each FB group, there is a Group Tag in FBDefh.xml file with following format:

```
<Group>
      <Name>Counters</Name>
      <Desc>Counter Group</Desc>
      <Active>True</Active>
 <FBList>
   <FBDef>
       <Name>UpCounter</Name>
       <DetailName>\FBD\Counters_UpCounter.xml</DetailName>
   </FBDef>
   <FBDef>
       <Name>DownCounter</Name>
       <DetailName>\FBD\Counters_DownCounter.xml</DetailName>
   </FBDef>

 </FBList>
</Group>
```

Above Group definition is for Counters Group. FBList tag contains all FB for this group. For each FB, there is an FBDef Tag with Name and DetailName elements.

Copy and paste counters Group Tag and change its tags as following:

```
<Group>
    <Name>IEC11313</Name>
    <Desc>IEC11313 Group</Desc>
    <Active>True</Active>
    <FBList>
        <FBDef>
            <Name>RS</Name>
            <DetailName>\FBD\IEC11313_RS.xml</DetailName>
        </FBDef>
        <FBDef>
            <Name>SR</Name>
            <DetailName>\FBD\IEC11313_SR.xml</DetailName>
        </FBDef>

    </FBList>
</Group>
```

Save FBDefh.xml file. You can define any number of FB header definition in FBList tag .

DetailName value is relative path of FB body definition XML file. Name value is Name of FB that is shown in FBeditor .

As a naming standard we will use following format for FB body definition file:

{groupName}_{FBName}.xml and all FB Body files are locate at \FBD\ directory .

Open \FBD\ directory and copy and paste one of existing FB Body files, change its name to IEC11313_RS.xml .change its content as following:

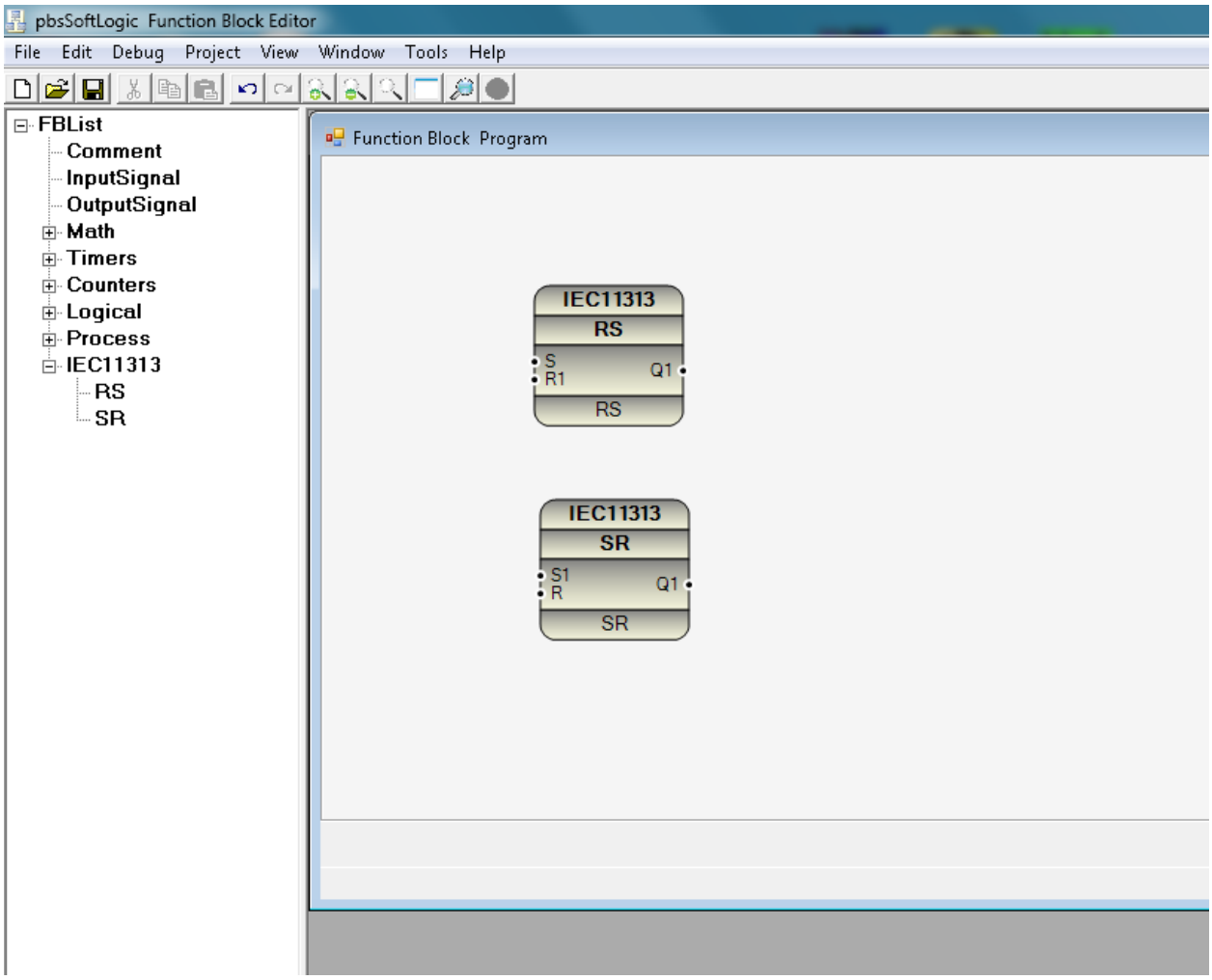
```
<?xml version="1.0"?>
<FBDef>
      <Version>1.0.0</Version>
      <FBDD>
            <Name>RS</Name>
            <Desc>RS Flip Flop</Desc>
            <Active>True</Active>
       <Interface></Interface>
       <InputList>
         <Input>
              <Name>S</Name>
              <Desc>Set Input</Desc>
              <Type>bool</Type>
              <Init>False</Init>
         </Input>
         <Input>
              <Name>R1</Name>
              <Desc>Reset Input</Desc>
              <Type>bool</Type>
              <Init>False</Init>
         </Input>

       </InputList>
       <OutputList>
         <Output>
              <Name>Q1</Name>
              <Desc>Q1 Output</Desc>
              <Type>bool</Type>
              <Init>False</Init>

         </Output>

       </OutputList>
      </FBDD>
</FBDef>
```

Open \FBD\ directory and copy and paste one of existing FB Body files, change its name to IEC11313_SR.xml .change its content as following:

```
<?xml version="1.0"?>
<FBDef>
      <Version>1.0.0</Version>
      <FBDD>
            <Name>SR</Name>
            <Desc>SR Flip Flop</Desc>
            <Active>True</Active>
       <Interface></Interface>
       <InputList>
         <Input>
              <Name>S1</Name>
              <Desc>Set Input</Desc>
              <Type>bool</Type>
              <Init>False</Init>
         </Input>
         <Input>
              <Name>R</Name>
              <Desc>Reset Input</Desc>
              <Type>bool</Type>
              <Init>False</Init>
         </Input>

       </InputList>
       <OutputList>
         <Output>
              <Name>Q1</Name>
              <Desc>Q1 Output</Desc>
              <Type>bool</Type>
              <Init>False</Init>

         </Output>

       </OutputList>
      </FBDD>
</FBDef>
```

In this stage you can use IEC11313 group in SoftLogic Editor. Close FBEditor and run it again.

You can see a new IEC11313 group is added to FBEditor and it has two Function Blocks.



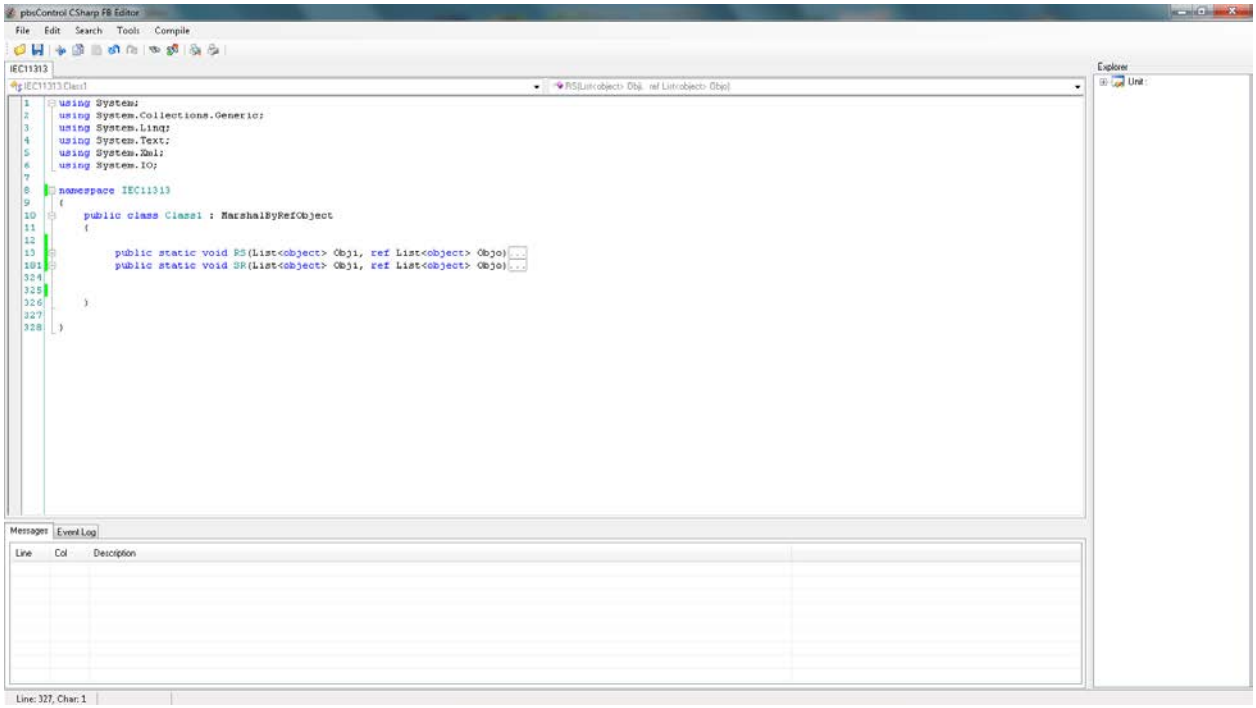
Drag and drop RS and SR Flip flops in a new application. RS and SR Flip Flops are ready to use in any Function block application.

Making C# Implementation of RS and SR Flip Flops:

Step1: make a new directory in VSLESrc named: IEC11313. It should be the same name of Group.

Step2: Copy and paste one of existing FB group source code to IEC11313 directory. You can use Timers directory class1.cs code. Copy class1.cs file to IEC11313 directory. Do not change Calss1.cs to any other name.

Step3: By Windows FB Editor Utility, open \VSLESrc\IEC11313\class1.cs code



Change Source code of IEC11313 groups as following:

```
namespace IEC11313
{
    public class Class1 : MarshalByRefObject
    {

        public static void RS(List<object> Obji, ref List<object> Objo)...
        public static void SR(List<object> Obji, ref List<object> Objo)...


    }

}
```

namespace name should be name of group.(IEC11313)

Class name should be Class1.Donot remove MarshalByRefObject from code.

Any FB Function has following format:

Public static void [name of FB] (List<object> Obji , ref List<object> Objo)

Change name of FB and do not change any other variables.

List<object> Obji : list of objects that is passed to FB

List<object> Objo : list of output Objects that is return from Function .

All pbssoftLogic FB Source code has following format:

```
public static void RS(List<object> Obji, ref List<object> Objo)
{

    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();

    Read Inputs and define Variables

    Read Static data

    Solve Logic

    Map Old Data

    Write Static data

    Write Outputs

}
```

Obji[0] : First item in Input Object List is always pbsSoftLogic Temporary path for reading and writing Static data of FBs.

Obji[1]:second item in Input Object List is always Unique Identifier for FB. Each FB instance has a unique identifier in logic.

Open FB properties window and find PartID. PartID has always unique value for each FB in logic by pbsSoftLgic automatically.

Open FB properties window and find InstanceName and FBName. By Default InstanceName and FBName has same value , but if you change InstanceName to any unique name in logic ,pbsSoftLogic

will use InstanceName as unique Identifier and InstanceNane will pass as second item in Input Object list.

Obji[2]:third item in Input Object List is always Logic name .

```
public static void RS(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    #region Read Inputs and define Variables
    bool S = bool.Parse(Obji[3].ToString());
    bool R1 = bool.Parse(Obji[4].ToString());


    bool  Q1 = false;

    bool S_Old = false;
    bool R1_Old = false;
    #endregion

    Read Static data

    Solve Logic

    Map Old Data

    Write Static data

    Write Outputs


}
```

pbsSoftLogic will pass FB Input values from Obji[3] . For example for RS Flip Flop value of S input is passed to Obji[3] and value of R1 is passed to Obji[4] .

You can have maximum 32 Input and 32 output port for each FB.

This is your responsibilities to change type of Inputs inside FB, pbsSoftlogic is passing all values as object to FB.

Because we need to detect Rising edge of S and R1 signals, so we must define two more variables: S_Old and R_Old .

Q1 is FB Output signal. All Output Signals should be static.

Read Static data:

```
#region Read Static data

string mPath = TmpPath + "\\RS_" + PID + "_" + VSLEName;

if (File.Exists(mPath) == true)
{
    XmlDocument xmldoc = new XmlDocument();
    xmldoc.Load(mPath);
    XmlNodeList TmpXmlNodeList = xmldoc.GetElementsByTagName("Tag");

    foreach (XmlNode TmpXMLNode in TmpXmlNodeList)
    {
        switch (TmpXMLNode.Attributes["Name"].InnerText)
        {
            case "Q1":
                Q1 = bool.Parse(TmpXMLNode.Attributes["Value"].InnerText);
                break;
            case "S_Old":
                S_Old = bool.Parse(TmpXMLNode.Attributes["Value"].InnerText);
                break;
            case "R1_Old":
                R1_Old = bool.Parse(TmpXMLNode.Attributes["Value"].InnerText);
                break;

        }
    }

}

#endregion
```

Any FB has one code but separate static data for each FB instance. When runtime kernel of pbsSoftLogic is calling a FB, inside FB Code, it will find Static data file by combination of FB Name, Unique Identifier (PID) and Logic name.

string mPath = TmpPath + "\\RS_" + PID + "_" + VSLEName;

Static data is any data that you want to keep its value when you return from function.

In RS FB sample, Output signal Q1, old value of S and old value of R1 are static data.

Runtime kernel of pbsSoftLogic is using RamDisk and XML files for modeling static data file.

So any instance of a FB has one XML file in RAM Disk which all Static data will keep there.

Solve Logic:

```
 public static void RS(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    #region Solve Logic
    if((S == true) && (S_Old == false))
    {
        // Raising Edge detect for S
        Q1 = true;
    }
    if((R1 == true) && (R1_Old == false))
    {
        // Raising Edge detect for R1
        Q1 = false;
    }

    #endregion

    Map Old Data

    Write Static data

    Write Outputs


}
```

For RS FB, when Raising Edge detect for S, Q1 value will set to true and if Raising edge detect for R1 , Q1 value will reset to false . Because R1 raising edge detection is after s, if both of them happened, Q1 will be rest to false.

Map Old data:

```
 public static void RS(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    Solve Logic

    #region Map Old Data

    S_Old = S;
    R1_Old = R1;

    #endregion

    Write Static data

    Write Outputs


}
```

S_Old and R1_Old should map to S and R1. So S_Old and R1_Old has value of S and R1 but for one cycle before.

Write Static data:

```
#region Write Static data

FileStream fs = new FileStream(mPath, FileMode.Create);
XmlTextWriter w = new XmlTextWriter(fs, null);

w.Formatting = Formatting.Indented;
w.Indentation = 5;


w.WriteStartDocument();
w.WriteStartElement("StaticData");

w.WriteStartElement("Tag");
w.WriteAttributeString("Name", "Q1");
w.WriteAttributeString("Value", Q1.ToString());
w.WriteEndElement();//OPCTagNode

w.WriteStartElement("Tag");
w.WriteAttributeString("Name", "S_Old");
w.WriteAttributeString("Value", S_Old.ToString());
w.WriteEndElement();//OPCTagNode

w.WriteStartElement("Tag");
w.WriteAttributeString("Name", "R1_Old");
w.WriteAttributeString("Value", R1_Old.ToString());
w.WriteEndElement();//OPCTagNode

w.WriteEndElement();//StaticData
w.Close();

#endregion
```

And end of FB, you need to save value of Static data on XML file for using in next cycle.

Write Outputs:

```
 public static void RS(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    Solve Logic

    Map Old Data

    Write Static data

    #region Write Outputs

    Objo.Add(Q1);

    #endregion


}
```

And at the end, value of all output signals should be writing on Objo list.

SR FB Implementation:

```
public static void SR(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    Solve Logic

    Map Old Data

    Write Static data

    Write Outputs


}
```

```
public static void SR(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    #region Read Inputs and define Variables
    bool S1 = bool.Parse(Obji[3].ToString());
    bool R = bool.Parse(Obji[4].ToString());


    bool Q1 = false;

    bool S1_Old = false;
    bool R_Old = false;
    #endregion

    Read Static data

    Solve Logic

    Map Old Data

    Write Static data

    Write Outputs


}
```

```
#region Read Static data

string mPath = TmpPath + "\\SR_" + PID + "_" + VSLEName;

if (File.Exists(mPath) == true)
{
    XmlDocument xmldoc = new XmlDocument();
    xmldoc.Load(mPath);
    XmlNodeList TmpXmlNodeList = xmldoc.GetElementsByTagName("Tag");

    foreach (XmlNode TmpXMLNode in TmpXmlNodeList)
    {
        switch (TmpXMLNode.Attributes["Name"].InnerText)
        {
            case "Q1":
                Q1 = bool.Parse(TmpXMLNode.Attributes["Value"].InnerText);
                break;
            case "S1_Old":
                S1_Old = bool.Parse(TmpXMLNode.Attributes["Value"].InnerText);
                break;
            case "R_Old":
                R_Old = bool.Parse(TmpXMLNode.Attributes["Value"].InnerText);
                break;

        }
    }

}

#endregion
```

```
public static void SR(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    #region Solve Logic
    if((R == true) && (R_Old == false))
    {
        // Raising Edge detect for R1
        Q1 = false;
    }

    if((S1 == true) && (S1_Old == false))
    {
        // Raising Edge detect for S
        Q1 = true;
    }

    #endregion

    Map Old Data

    Write Static data

    Write Outputs


}
```

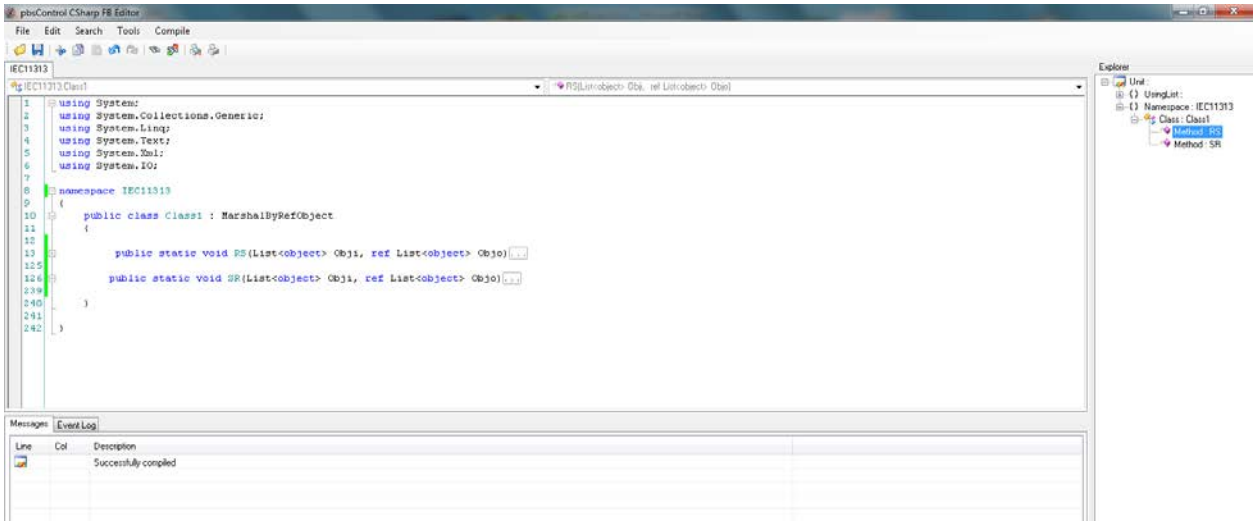
```
public static void SR(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    Solve Logic

    #region Map Old Data

    S1_Old = S1;
    R_Old = R;

    #endregion

    Write Static data

    Write Outputs


}
```

```
#region Write Static data

FileStream fs = new FileStream(mPath, FileMode.Create);
XmlTextWriter w = new XmlTextWriter(fs, null);

w.Formatting = Formatting.Indented;
w.Indentation = 5;


w.WriteStartDocument();
w.WriteStartElement("StaticData");

w.WriteStartElement("Tag");
w.WriteAttributeString("Name", "Q1");
w.WriteAttributeString("Value", Q1.ToString());
w.WriteEndElement();//OPCTagNode

w.WriteStartElement("Tag");
w.WriteAttributeString("Name", "S1_Old");
w.WriteAttributeString("Value", S1_Old.ToString());
w.WriteEndElement();//OPCTagNode

w.WriteStartElement("Tag");
w.WriteAttributeString("Name", "R_Old");
w.WriteAttributeString("Value", R_Old.ToString());
w.WriteEndElement();//OPCTagNode

w.WriteEndElement();//StaticData
w.Close();

#endregion
```

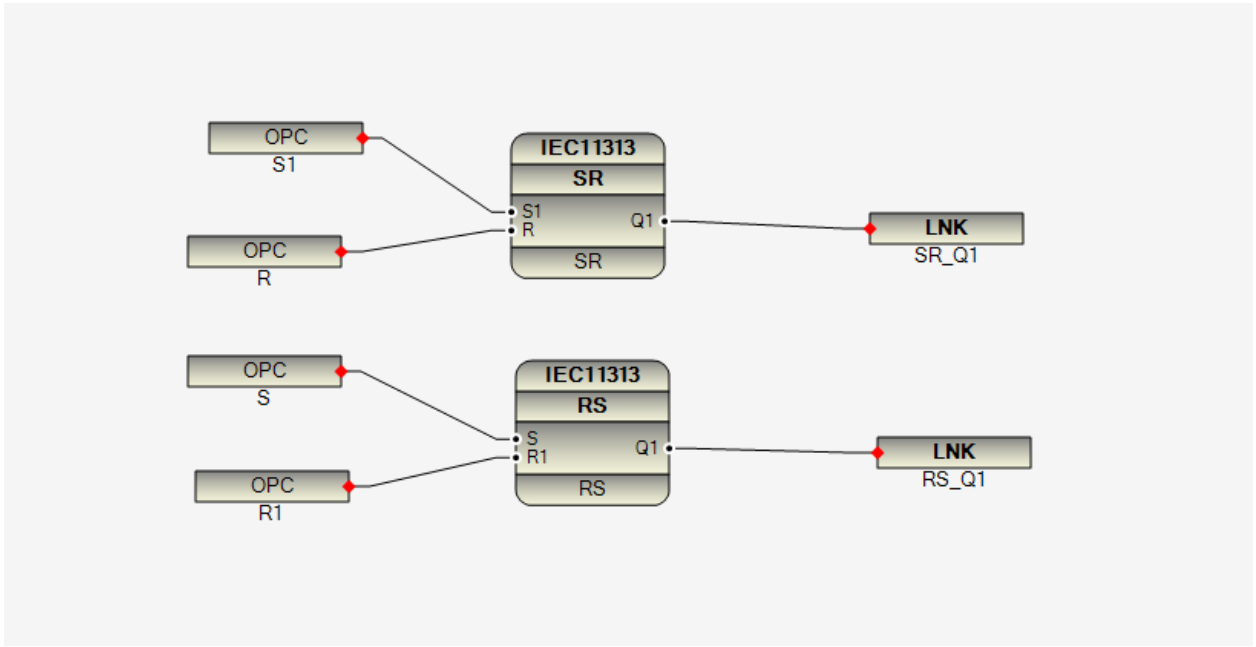
```
public static void SR(List<object> Obji, ref List<object> Objo)
{


    string TmpPath = Obji[0].ToString();
    string PID = Obji[1].ToString();
    string VSLEName = Obji[2].ToString();


    Read Inputs and define Variables

    Read Static data

    Solve Logic

    Map Old Data

    Write Static data

    #region Write Outputs

    Objo.Add(Q1);

    #endregion


}
```

In this stage, you can compile IEC11313 group. Select “Make FB” from Compile Menu.

FB compiler will make IEC11313.dll file at \PSLE\VSLELib Directory.

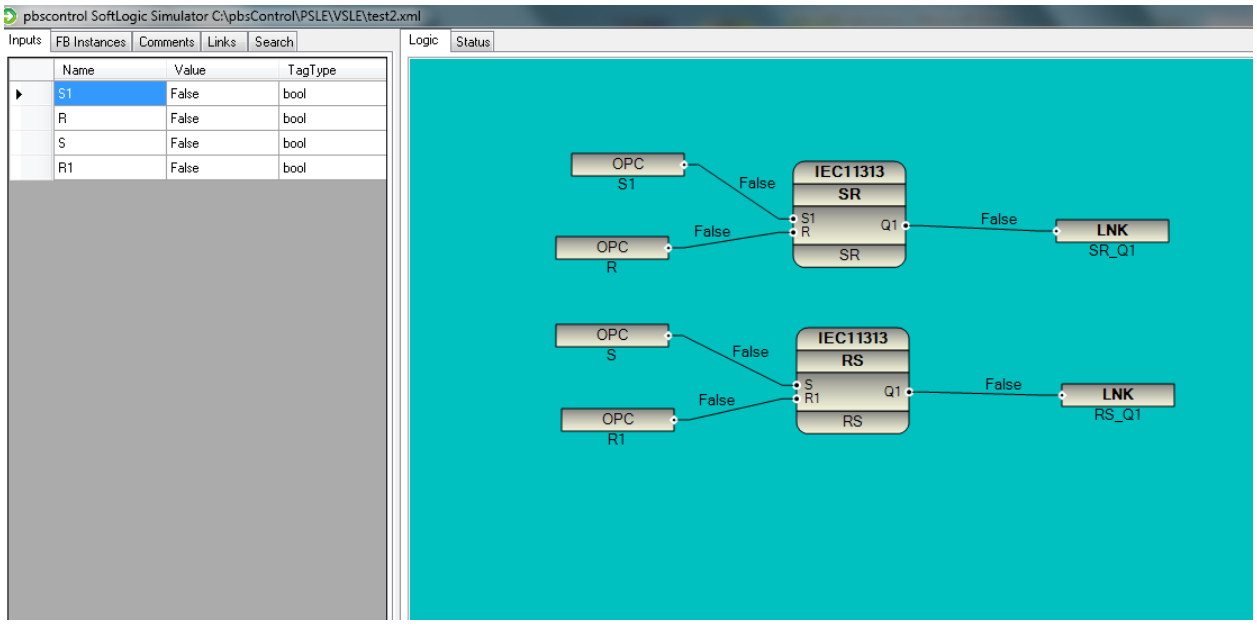


For testing RS and SR function blocks, you can write a simple logic as following:

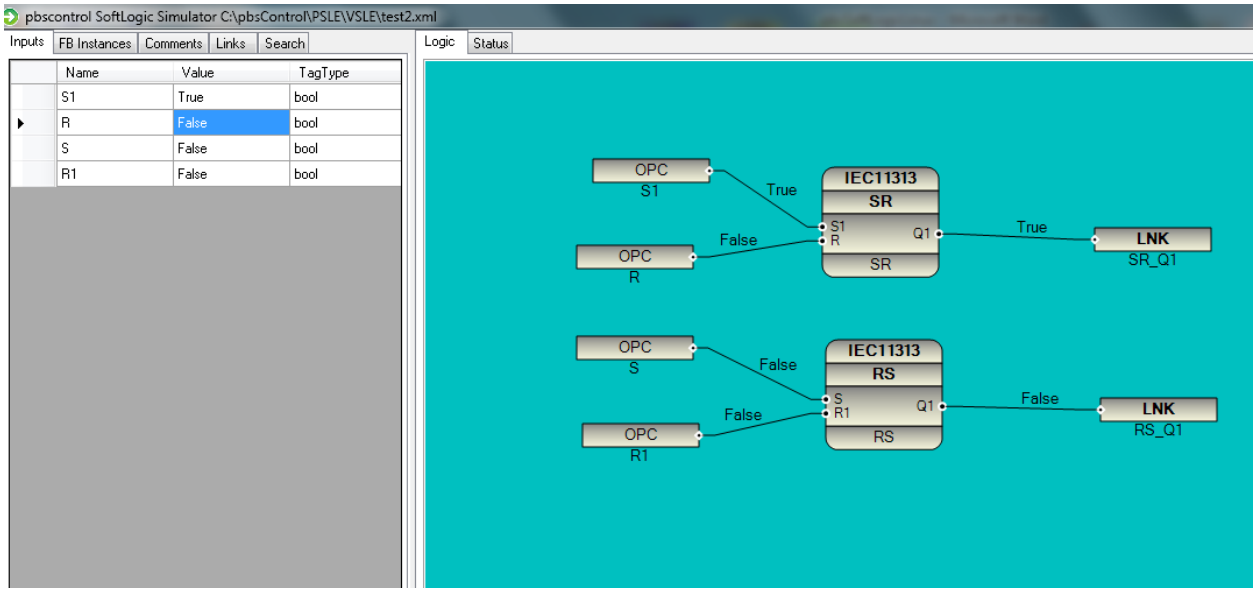


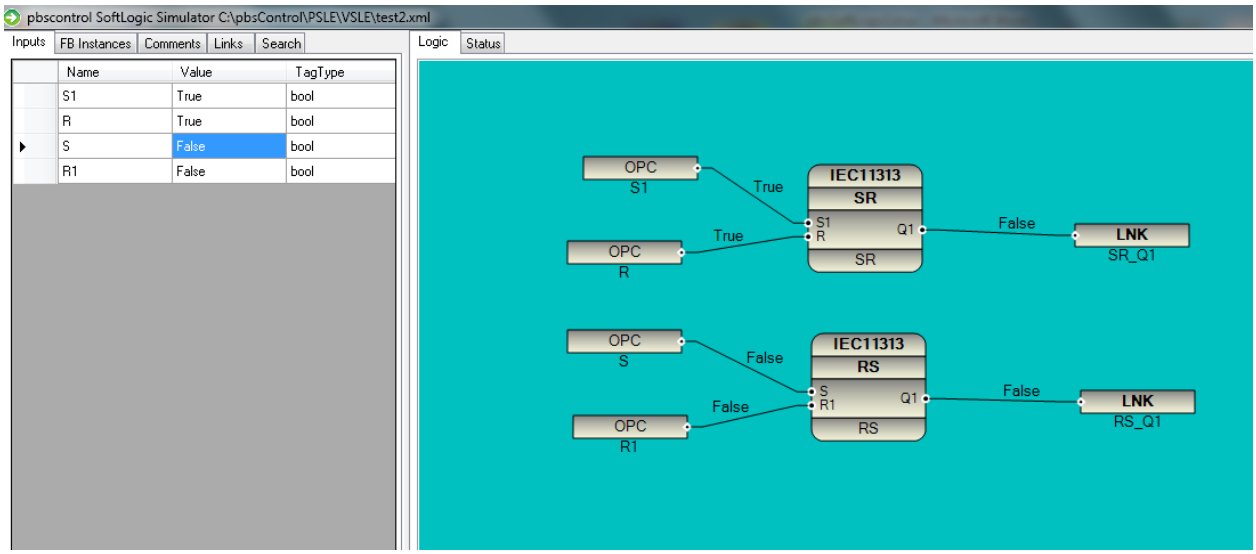
In above logic OPC tags S1 , R , S and R1 are dummy tags but LNK SR_Q1 and LNK RS_Q1 are internal tags in logic .

You can simulate above logic by Simulator utility.

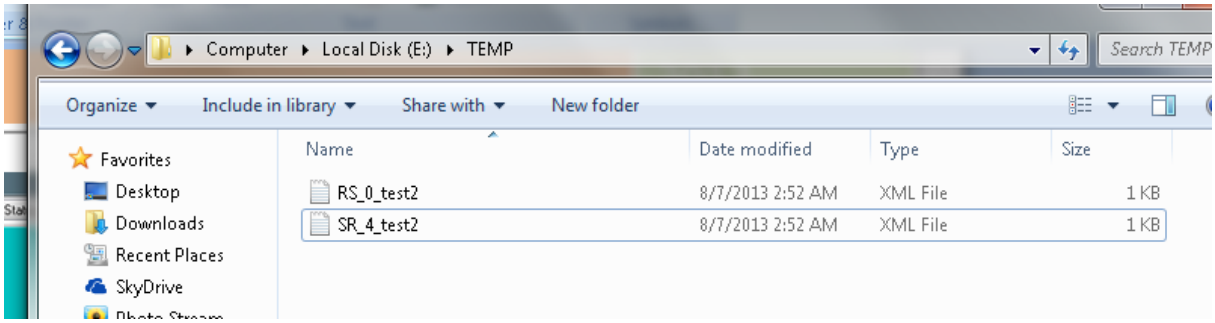


This logic is not running on real pbsSoftLogic Windows Kernel because you need to have real OPC tags but it works under Simulator.





Open ram disk drive and open temp directory, you can see two XML files:



These are static data files for RS and SR Function blocks. Open Static data file for RS function block you can see following items:

```
<?xml version="1.0"?>
<StaticData>
      <Tag Name="Q1" Value="False" />
      <Tag Name="S_Old" Value="False" />
      <Tag Name="R1_Old" Value="False" />
</StaticData>
```

This is the same XMl file that you save in RS C# code.

Writing C code for Linux and cross compiling of UDF

We developed an UDF with C# and compile it for simulator and windows runtime. If you want to run your UDF on linux controller like W406 , IA240 , UC7112 you must write UDF by C and compile it for embedded linux .

For cross compiling for embedded linux you need to have following software's:

1 – Ubuntu Linux distribution. You can download from http://www.ubuntu.com/download/desktop

2 – Install ubuntu on a Virtual Machine like VMWare , or install it on a PC .

3 – Download eclipse IDE from http://www.eclipse.org/downloads/ and download Eclipse IDE for C/C++ Developer for linux 32 or 64 bit .

4 – MOXA has different cross compiler for different linux based controller. Based on your controller download cross compiler from following links:

1 - **Tool-Chain for Linux Kernel 2.6.x for IA24X, W3XX-LX Series, UC-7112-LX Plus, IA3341**

http://www.moxa.com/support/sarch_result.aspx?type=soft&prod_id=121&type_id=6

2 - **Tool-chain for W406, IA26X-LX and EM2260-LX**

http://www.moxa.com/support/sarch_result.aspx?type=soft&prod_id=34&type_id=6

3 - **Tool-chain for UC-8400-LX Series**

http://www.moxa.com/support/sarch_result.aspx?type=soft&prod_id=443&type_id=6

4 - **Linux Tool-Chain for ioPAC 8500-C Series**

http://www.moxa.com/support/sarch_result.aspx?type=soft&prod_id=939&type_id=6

5 - **Linux Tool-Chain for ioLogik W5348-HSDPA-C and ioPAC 8020-C series**

http://www.moxa.com/support/sarch_result.aspx?type=soft&prod_id=605&type_id=6

5 – Install cross compiler for your hardware on ubuntu by sh command as following :

sudo sh { path and name of tool chain}

you should runsh command with sudo command as linux supervisor .

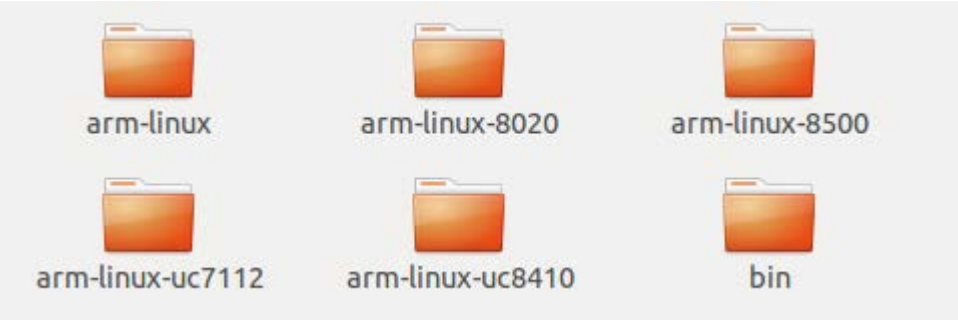
ubuntu will ask your supervisor password and start to install cross compiler at usr/local/arm-linux directory .

NOTE : if you installed other cross compilers in ubuntu rename them before installing new cross compiler . You can rename arm-linux directory with following command:

sudo nautilus

This will open a file explorer with supervisor right, so you have access to rename arm-linux directory.

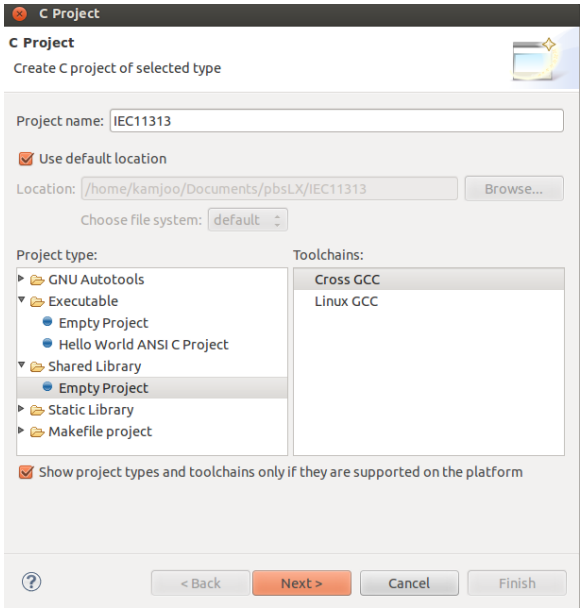
In following figure you can see my ubuntu usr/local directory with different cross compilers.



Active cross compiler has arm-linux directory name. In above figure active cross compiler is W406 series. If you want to compile for ioPAC8020 , rename arm-linux to arm-linux-w406 then rename arm-linux-8020 to arm-linux .

You can find source code of all pbsSoftLogic at c:\PSLE\CSrc directory.

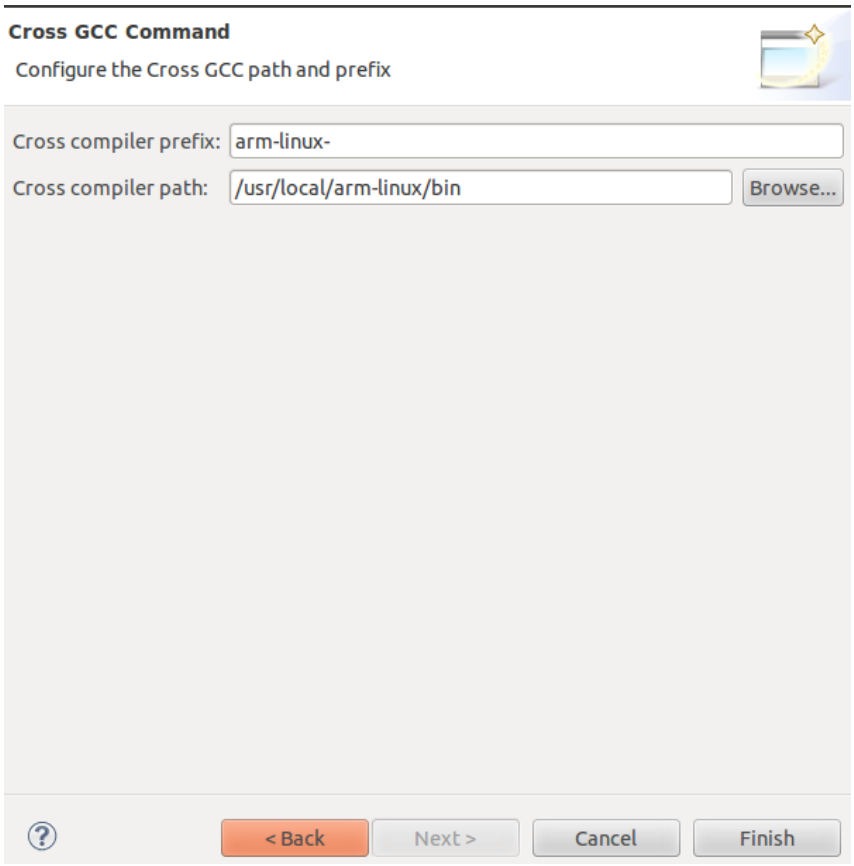
Open eclipse IDE and make a new C Project. Project name should be exactly same name of UDF group, for our example “IEC11313”



Project Type: Share Library

Toolchain : Cross GCC

Click on Next.



Set Cross Compiler prefixes and cross compiler path as above figure.

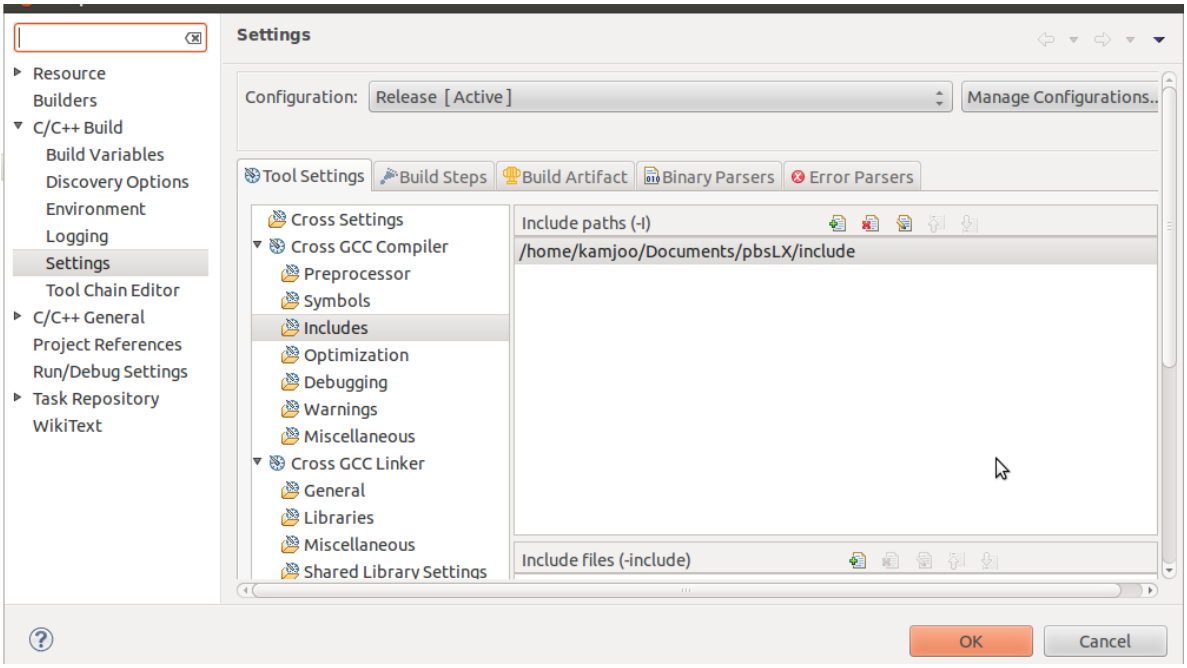
Click on finish button.

Copy paste one of existing source code from CSrc directory to new project directory .

Suppose you will copy counter source file (MainCounters.c) to IEC11313 directory . Rename MainCounters.c to MainIEC11313.c .

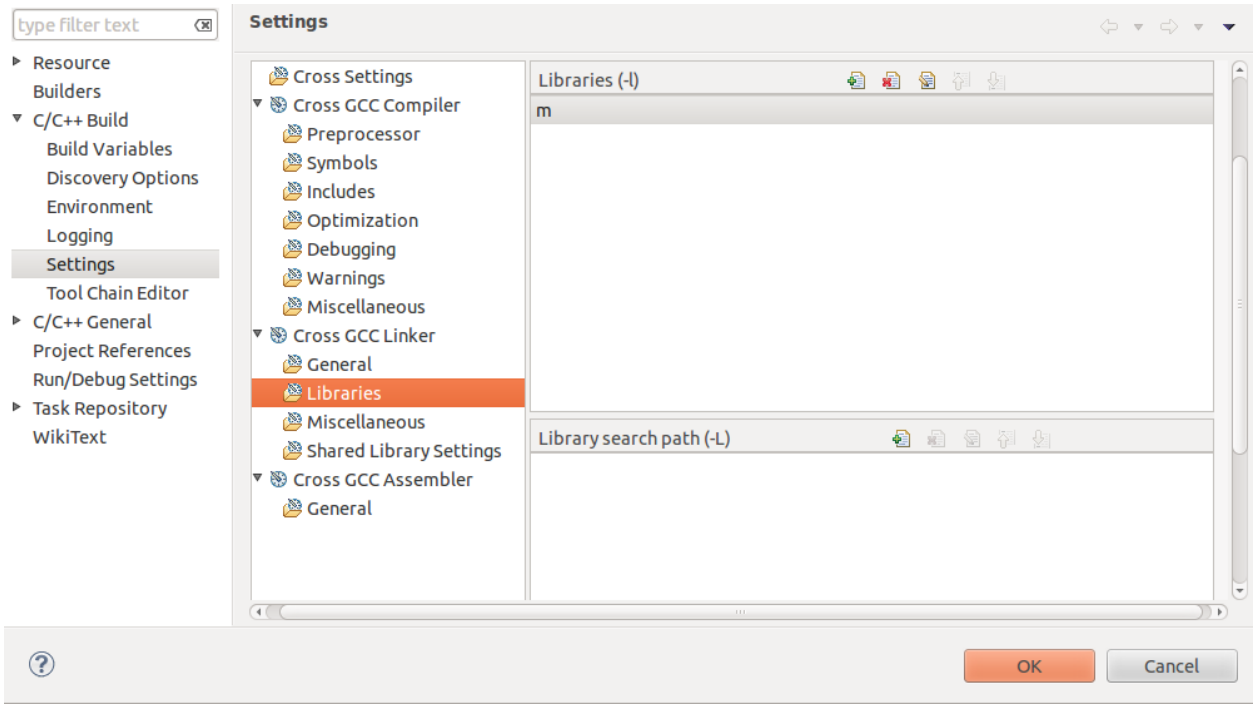
Select IEC11313 project in eclipse and refresh project to include MainIEC11313.c file to project .

There is include directory in CSrc folder that need to be included to IEC11313 project . open project properties in eclipse and add Include directory path to project .

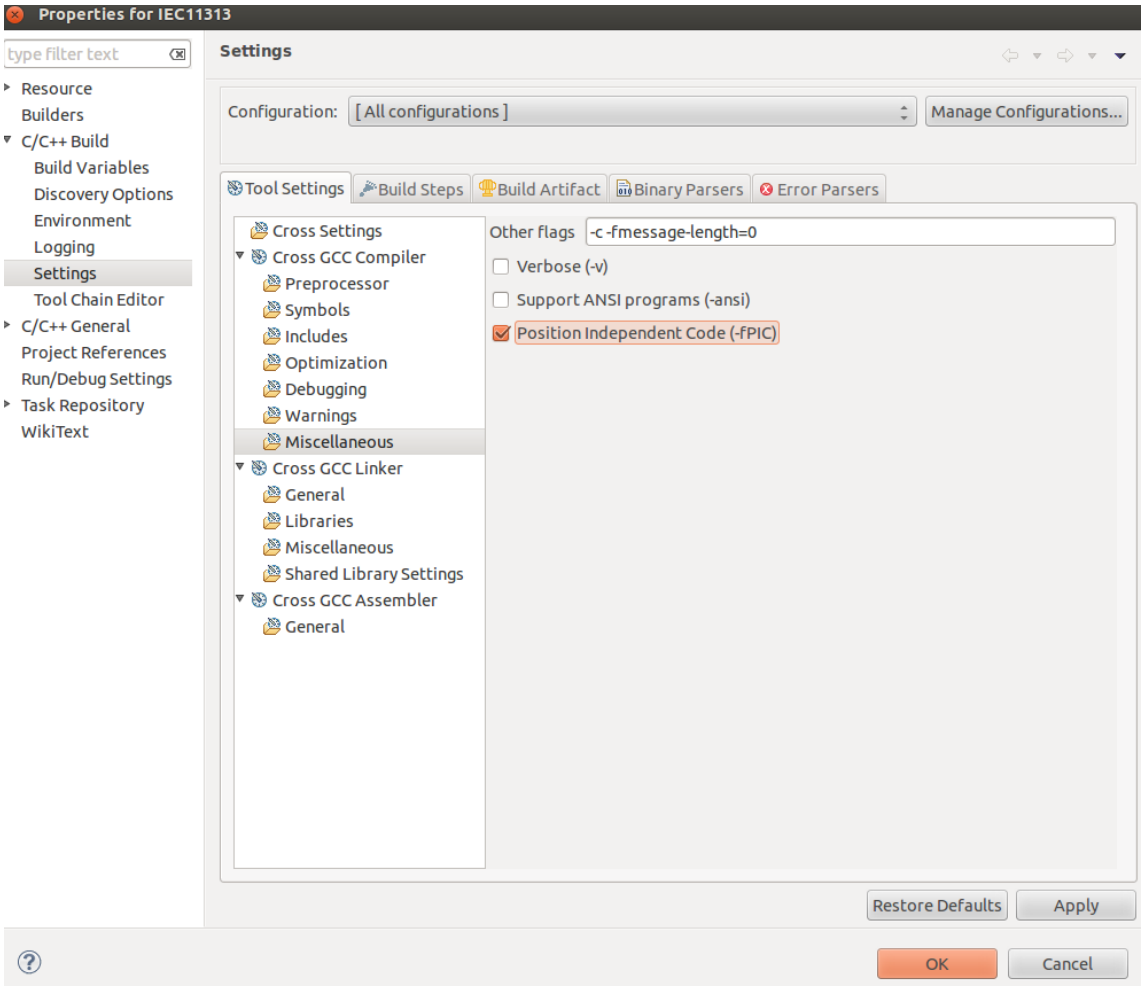


Select release mode as active mode in manage configuration. Add include directory for Debug and release configuration.

If you use GCC mathematical library in UDF , you need to add m library to project .



In project settings /Miscellaneous enable Position Independent Code –PIC .



Open MainIEC11313.c source code in eclipse. Change name of UpCounter functions to RS and DownCounter function to SR.

Any C FB has following format:

```
void RS(pbsObject *  Obji, pbsObject  * Objo)
```

Don't change function format and just change name of function to SR and RS.

obji is list of all inputs to function .

objo is list of all FB outputs .

In Linux kernel first FB input is passed by index 5 of obji . **int** S = Obji[5].dvalue;

```
void RS(pbsObject *  Obji, pbsObject  * Objo)
{
        char  TmpPath[64] ;
        char PID[32];
        char ProgName[32];
        char  TmpSRamPath[64] ;
        char TmpSDPath[64];



        int S = Obji[5].dvalue;
        int R1 = Obji[6].dvalue;
```

Index 0 to 4 is used for passing system data for reading /writing static data.

```
        strcpy(TmpPath , Obji[0].strvalue);
        strcpy(PID , Obji[1].strvalue);
        strcpy(ProgName , Obji[2].strvalue);

        strcpy(TmpSRamPath , Obji[3].strvalue);
        strcpy(TmpSDPath ,Obji[4].strvalue);

        // Read Static data

        FILE * m_db ;
        DBStruct db_elem;
        char TmpStaticDataPath[128];
        strcpy(TmpStaticDataPath,TmpPath);
        strcat(TmpStaticDataPath,"/RS_");
        strcat(TmpStaticDataPath,ProgName);
        strcat(TmpStaticDataPath,"_");
        strcat(TmpStaticDataPath,PID);
        strcat(TmpStaticDataPath,".dat");
```

Reading Static data:

```
m_db = fopen(TmpStaticDataPath, "rb");
if(m_db==NULL)
{
        // first time generate this file . User default value for static  data

}
else
{
        // Read Value of static data
        while (feof(m_db)==0)
        {
                fread(&db_elem, sizeof(db_elem), 1, m_db);
                if(strcmp(db_elem.name ,"SOld")==0)
                {
                        SOld = atoi(db_elem.value);

                }
                if(strcmp(db_elem.name ,"R1Old")==0)
                {
                        R1Old = atoi(db_elem.value);

                }

                if(strcmp(db_elem.name ,"Q1")==0)
                {
                        Q1 = atoi(db_elem.value);
                }


        }
        fclose(m_db);
}
```

In RS FB, old status of S, R 1 and Q1 value should be static.

You need to make all outputs in FB with static data as static tags.

Solve logic , map old values and write outputs :

```
if((S==1)&&(SOld==0))
{
        Q1 = 1;
}

if((R1==1)&&(R1Old==0))
{
        Q1 = 0;
}


Objo[0].dvalue = Q1;

// Map New Static data to old one
SOld = S;
R1Old = R1;
```

Write static data :

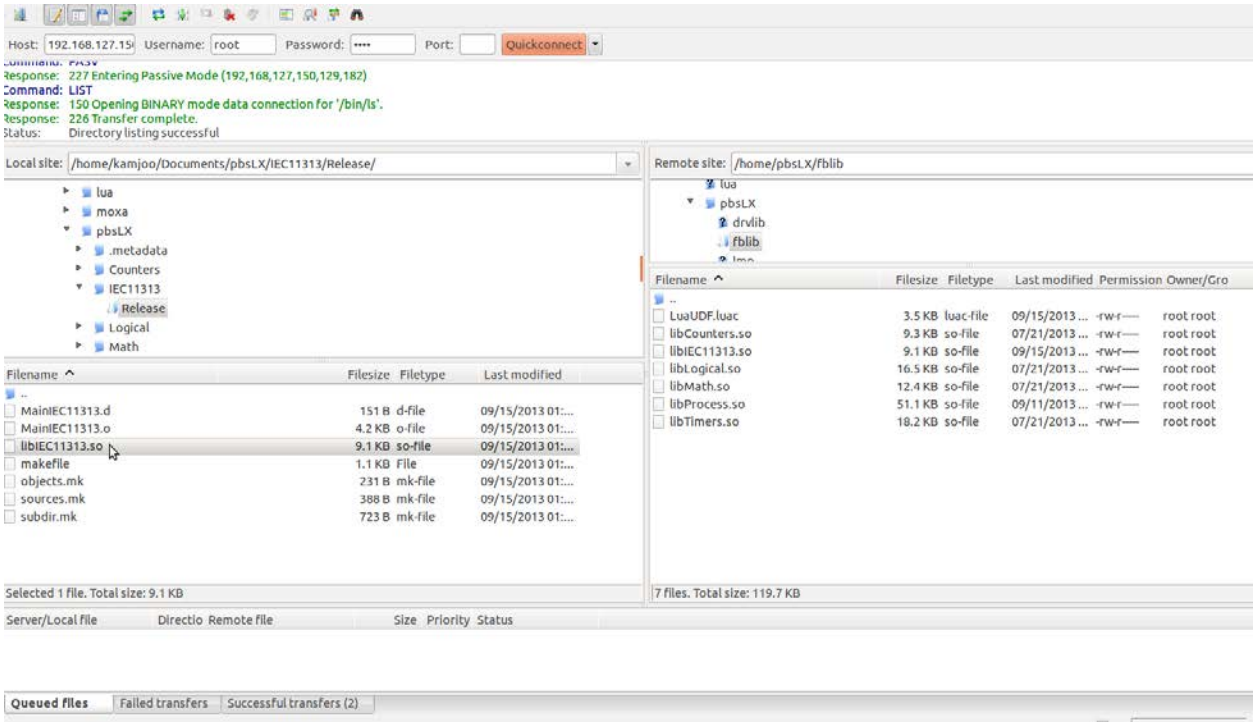
```
// Save Static data
        m_db = fopen(TmpStaticDataPath, "wb");



        strcpy(db_elem.name,"SOld");
        sprintf(db_elem.value,"%d",SOld);
        fwrite(&db_elem, sizeof(db_elem), 1, m_db);


        strcpy(db_elem.name,"R1Old");
        sprintf(db_elem.value,"%d",R1Old);
        fwrite(&db_elem, sizeof(db_elem), 1, m_db);


        strcpy(db_elem.name,"Q1");
        sprintf(db_elem.value,"%d",Q1);
        fwrite(&db_elem, sizeof(db_elem), 1, m_db);


        fclose(m_db);
```

In this stage you can compile FB. Eclipse will make libIEC11313.so file at release directory.

You should copy this file to controller. /home/pbsLX/fblib directory .

You can use filezilla for transferring libIEC11313.so file to controller.

Please notice that transfer mode must be Binary in Filzilla .



10 – 2 Lua UDF Development

Lua scripting language is developed at 1993 by Roberto Ierusalimschy , Walder Celes and Luiz Henrique at university of PUC-Rio Brazil .(http://www.lua.org/authors.html) . For detail information about Lua , please refer to www.lua.org .

For quick Lua introduction, please visit http://www.inf.puc-rio.br/~roberto/talks/ppl-2012.pdf

In last 20 years Lua is used in many projects and devices:

TVs (Samsung), routers (Cisco), keyboards (Logitech), printers (Olivetti), set-top boxes (Verizon), M2M devices (Sierra Wireless),calculators (TI-Nspire),Wireshark, Snort, Nmap, VLC Media Player, LuaTeX

Adobe Lightroom One million lines of Lua code

Slashdot: News for nerds, Feb 1, 2012:

"Wikipedia Chooses Lua as its new template language "

Lua is used in many game development environments as programming framework:

Corona SDK - http://www.coronalabs.com/products/corona-sdk/
Gideros Studio - http://www.giderosmobile.com/
Moai – http://www.getmoai.com/
Love -https://love2d.org/
Codea - http://twolivesleft.com/

Lua is fast, small and very reliable. Lua is an active project and worldwide accepted as scripting language. So we selected Lua instead of ST as pbsSoftLogic scripting language for developing user defined Function blocks.
Lua Virtual machine is integrated to pbssoftlogic Linux Runtime kernel Version 1.5 and logic simulator. We didn't include Lua in pbsSoftLogic windows Runtime because you can develop UDF by C# and no need for Lua . We will use Lua for developing UDF for Linux based controllers and logic simulator.

When you use Lua for developing UDF, you don't need to use Linux cross compiler.
For developing Lua UDF you need to do following steps:

1 - Defining FB Input / Output structure – define UDF body. This step is same as C# /C UDF development.

2 – Write UDF script by pbsSoftLogic Lua Editor.

3 – Compile Lua source code for checking programming errors.

4 – Test Lua UDF by Logic simulator.

5 – Transfer Lua source code to controller.

We will compile Lua source code just for checking programming errors. We do not transfer compiler code to linux controller. When you transfer Lua UDF to controller, it will transfer Lua UDF source code.

pbsSoftLogic Linux controller , compiles Lua UDF source code when it load UDF.

10 – 2 Lua Language basics

Lua is dynamically typed language. There are eight basic type in Lua :

- Nil – no value , default value of a variable before initialization
- Boolean : has value false and true
- Number :double precision floating point
- String: sequence of characters. like “pbsSoftlogic”
- userdata (not used in pbsSoftlogic)
- thread (not used in pbsSoftlogic)
- table (will use for passing FB input outputs to Lua)

Tables are the main data structure in Lua . Look at following samples:

```
a = {} -- create a table and store its reference in 'a'
k = "x"
a[k] = 10 -- new entry, with key="x" and value=10
a[20] = "great" -- new entry, with key=20 and value="great"
print(a["x"]) --> 10
k = 20
print(a[k]) --> "great"
a["x"] = a["x"] + 1 -- increments entry "x"
```

In pbsSoftLogic we pass FB input output values by Table. In following figure you can see very simple pbsSoftLogic Lua function . You should follow same structure for your UDF:

```
function  fun3(Obji )

-- TmpPath , PID  , SRAMPath,SDPath are same for all FB . Do not delete them

    local Objo = {}

    TmpPATH =  Obji["1"]
    TmpPID = Obji["2"]
    TmpLogic = Obji["3"]
    TmpSRam = Obji["4"]
    TmpSD = Obji["5"]

     --read inputs
     in1 = tonumber(Obji["6"])
     in2 = tonumber(Obji["7"])

    -- define  output signals
     local out1 = 0
     local out2 = 0

     -- read Static data

     -- Solev logic

     --save static data

     -- write outputs
      Objo["1"] = tostring(out1)
      Objo["2"] = tostring(out2)

    return Objo
end
```

Obji = input table to FB. It contains all FB inputs. The first fifth element is used by pbsSoftlogic Linux kernel to pass following data to any UDF:

Obji[“1”] = path of RAMDisk Drive in Linux Controller for saving static data . for example it is like “/mnt/ramdisk/” it include “/” .

Obji[“2”] =unique Identifier of UDF .

Obji[“3”] = name of program. In Linux Kernel it is always “logic”

Obji[“4”] = SRAM address in controller . It is RAM with battery backup. It include “/”

Obji[“5”] =SD address . It is External flash SD card address for data logging. It include “/”

Points:

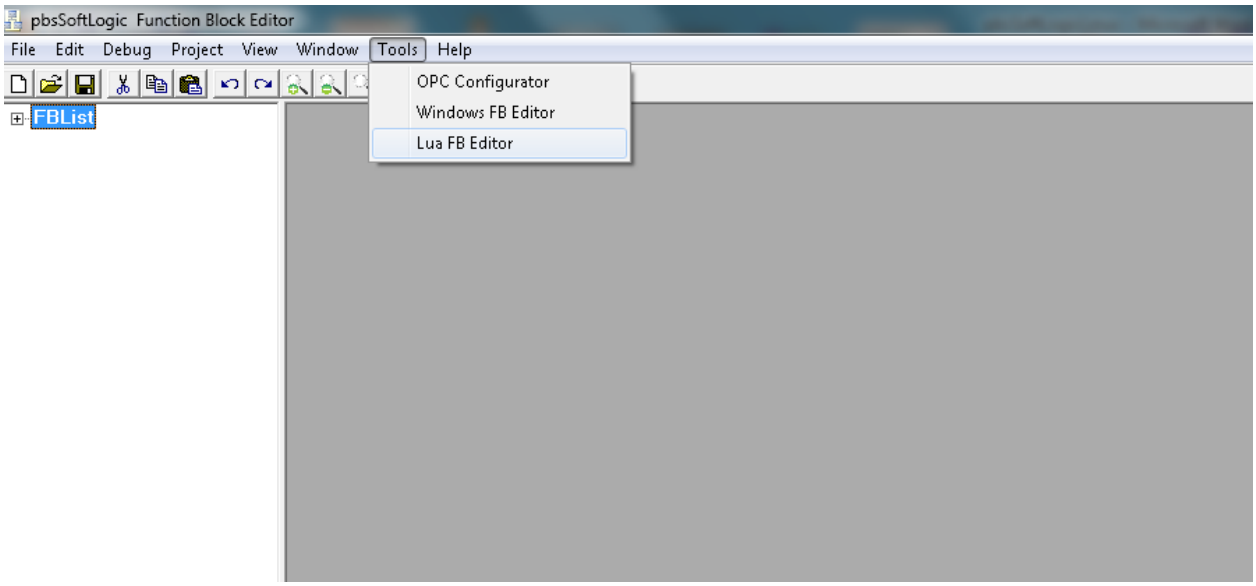
- UDF inputs start from key “6”.
- All key value should be as string number: “1” ,”2” ,”3”,…
- All inputs are pass as string to Lua . So you should change its type to number by tonumber function. Example in1 = tonumber(obji[“6”]) . This is value of first UDF input.

objo is return table from Lua.

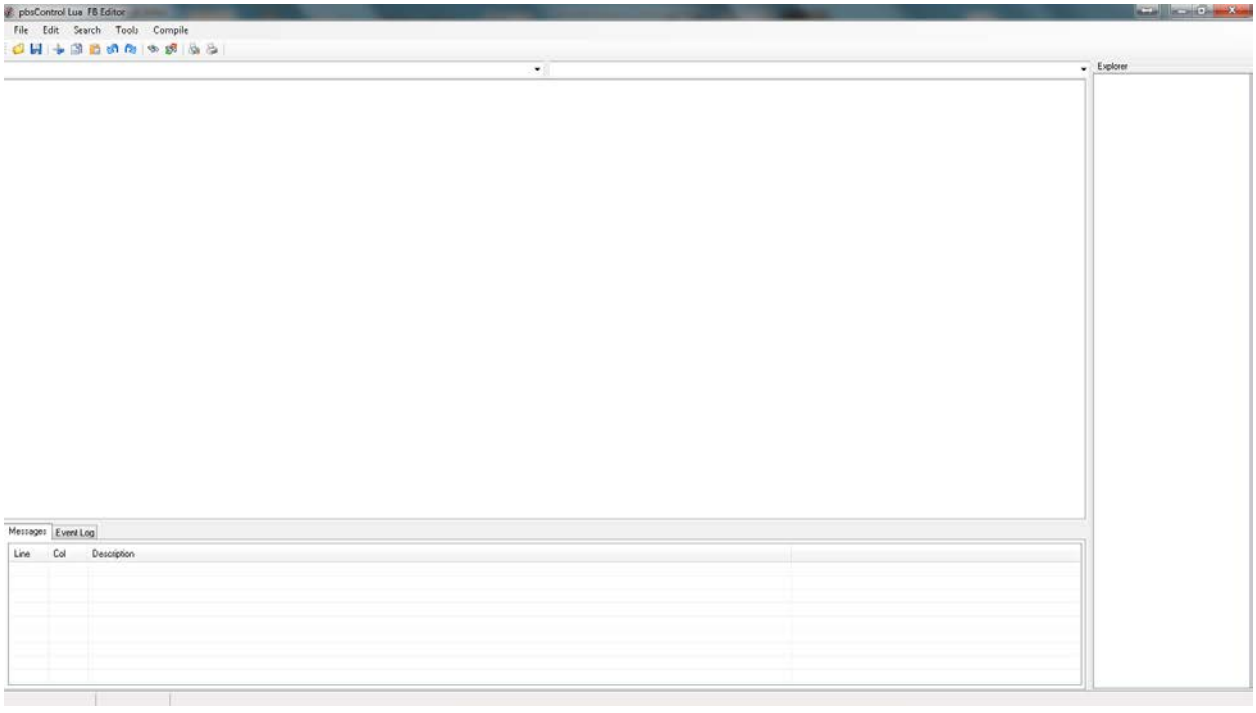
Points:

- objo key start from”1”.
- objo[“1”] = first UDF output
- objo[“2”] = second UDF output
- objo[“n”] = n’th UDF output n<32
- All values will return to pbsSoftlogic linux kernel by string format by tostring function .
- objo[“1”] = tostring(out1)
- Last statement in Lua should be return objo .

pbsSoftLogic included Lua editor . Open Lus editor from tools menu.



Run Lua FB editor. You can see following environment:



- Source code of Lua UDF is at \PSLE\LuaSrc directory.

You can define Lua functions and Lua Function Blocks in pbsSoftLogic .

Lua Function: Function Without static data.

Lua Function Block: Function With Static data.

In Lua FB Editor , execute “New Lua Function” from File menu . It will make NewLuaFun.Lua file at \PSLE\LuaSrc directory .

Lua FB Editor will make NewLua_Fun function as template for Lua Functions:

```
function  NewLua_Fun(Obji)
    -- Lua Function = Function without static data . There is no memory in function code.
    local Objo = {}
    --Define Output Variables
    local out1=0
    local out2=0
    --add more output variables  here
    --Define Intrnal Variables
    --Read Input variables
   Input1 = tonumber(Obji["6"])
   Input2 = tonumber(Obji["7"])
    --add more input variables here
    --Solve Logic
    --Write your logic here
    out1 = Input1 + Input2
    out2 = Input1 - Input2
    --Write outputs
    Objo["1"] = tostring(out1)
    Objo["2"] = tostring(out2)
    return Objo
end
```

There is no memory in Lua Functions. Input signals will pass to function and output values will calculate based on current value of inputs.

At following figure, we calculate (x^2 + Y^2)^0.5

```
function fun3(Obji)
    -- Lua Function = Function without static data . There is no memory in function code.
    local Objo = {}
    --Define Output Variables
    local L=0
    --add more output variables  here
    --Define Intrnal Variables
    --Read Input variables
   X = tonumber(Obji["6"])
   Y = tonumber(Obji["7"])
    --add more input variables here
   --Solve Logic
   --Write your logic here
   L = (X ^2 + Y^2)^0.5
   --Write outputs
   Objo["1"] = tostring(L)
   return Objo
end
```

In Lua FB Editor, execute “New Lua Function Block” from File menu. It will make NewLuaFB.Lua file at \PSLE\LuaSrc directory.

Lua FB Editor will make NewLua_FB function as template for Lua Function block:

```lua
function  NewLua_FB(Obji)
    -- Lua Function Block = Function   with static data . There is  memory in function Block code.
    local Objo = {}
    -- TmpPath , PID ,VSLEName , SRAMPath,SDPath are same for all FB . Do not delete them
   TmpPath  = Obji["1"]
   PID      = Obji["2"]
   VSLEName = Obji["3"]
   SRAMPath = Obji["4"]
   SDPath   = Obji["5"]
    --Define Output Variables
    local out1=0
    local out2=0
    --add more output variables  here
    --Define Intrnal Variables
    local state=0 -- this is static signal , so we will keep its last value in FB memory block
    local  dt=0
    local  input1old = 0
    --Read Input variables
   Input1 = tonumber(Obji["6"])
   Input2 = tonumber(Obji["7"])
    --add more input variables here
    --Read Static data
```

In this Lua FB sample, we consider following variables:

- Two output signal – ou1 , out2
- Three static signal – state , dt , input1old
    - State shows current state of FB.
    - dt is date time signal . In Lua os.time() function returns seconds from 1/1/1970 . When you compare current time with dt , it shows seconds passed from dt .
    - input1old is used for detecting rising edge of input1 signal .

```lua
    --Read Static data
   local fr  = io.open(TmpPATH .. "NewLua_FB_" .. TmpPID  .. "_" .. TmpLogic ..  ".dat" ,"r")
   if fr ~=nil then
       while true do
           line = fr:read()
               if line ~= nil then
                   i ,j =string.find(line,"=")
                   n = string.sub(line,1,j-1)
                   if n == "state" then
                       state = tonumber(string.sub(line,j+1))
                   end
                   if n == "dt" then
                       dt = tonumber(string.sub(line,j+1))
                   end
                   if n == "input1old" then
                       input1old = tonumber(string.sub(line,j+1))
                   end
                   -- add more static variables here
                   -- if n == "varstatic1" then
                   --            varstatic1 = tonumber(string.sub(line,j+1))
                   -- end
                   if n == "out1" then
                       out1 = tonumber(string.sub(line,j+1))
                   end
                   if n == "out2" then
                       out2 = tonumber(string.sub(line,j+1))
                   end
                   -- All Output Variables should be Static
                   -- if n == "out3" then
                   --            out3 = tonumber(string.sub(line,j+1))
                   -- end
               else
                   break
               end
       end
   fr:close()
   end
```

In pbsSoftlogic static data is simulated by a data file In Controller ram disk.

If Logic scan time is set to 500 msec , then every second , whole logic will execute for two times .

For each function block we have one static data file which is located on ram disk.

Because static data files are located on ram disk, so continues read /write of static data files will not make damage on controller and we will not lose system performance.

Static data file name is generated from function Block name, function block unique ID and logic name.

Function Block Unique ID – TmpPID - and Logic name – TmpLogic - are passed by pbsSoftLogic Linux kernel to function block. In Static data file name, you need to change function block name to your UDF name. For above FB, Static data file is as following figure:

```
state=0
dt=1379658085
input1old=0
out1=0
out2=0
```

We read static data file, line by line and find value of static signals and initialize static data tags at beginning of FB.

Always consider output signals as static and save their values in static data file. Normally output signals are not calculated in function block at each cycle, so you need to use old value of output signals in current cycle.

```
59      --Solve Logic
60      --Write your logic here
61      if Input1 ==1 and input1old ==0 then
62          dt = os.time()      -- save start time here
63          state = 1
64      end
65      if Input1 ==0 and input1old ==1 then
66          state = 0
67      end
68      if state ==1 then
69          if os.time() -dt< 10  then  -- compare current time with start time ,
70              out1 = 1
71              out2 = Input2
72          else
73              out1 = 0
74              out2 = 0
75          end
76      else
77          out1 = 0
78          out2 = 0
79      end
80      input1old = Input1
81      --Save Static data
82      local fw  = io.open(TmpPATH .. "NewLua_FB_" .. TmpPID  .. "_" .. TmpLogic ..  ".dat" ,"w")
83      fw:write("state=" .. tostring(state) .. "\n")
84      fw:write("dt=" .. tostring(dt) .. "\n")
85      fw:write("input1old=" .. tostring(input1old) .. "\n")
86      fw:write("out1=" .. tostring(out1) .. "\n")
87      fw:write("out2=" .. tostring(out2) .. "\n")
88      fw:close()
89      --Write outputs
90      Objo["1"] = tostring(out1)
91      Objo["2"] = tostring(out2)
92      return Objo
93  end
```

After reading input signals and static tags, you need to solve your logic.

Always remember that your logic is executing many times in a second.

For detecting rising edge or falling edge of a signal, you need to compare current value of signal with value of signal at last cycle.

```
if Input1 ==1 and input1old ==0 then
    dt = os.time()     -- save start time here
    state = 1
end
```

Input1 is current value of signal and input1Old is last value of signal.

At end of function block, always you need to map current value of signal to old value.

```
input1old = Input1
```

Normally function block is in a specific state at each cycle. So you need to define state static tag and set its value by input signal changes or internally in the function block. In above example when there is rising edge at Input1 signal, we will set state to 1 and will save time by os.time() function . os.time() returns current time from 1/1/1970 in seconds .

In following code, when Inut1 signal has falling edge, FB will go to state zero.

```
if Input1 ==0 and input1old ==1 then
    state = 0
end
```

In following code, when FB is in state one, it will map Input2 to out2 and sets out1 to 1 for 10 seconds. If before 10 seconds, falling edge detecting for Input1 signal, FB goes to state 0.

```
if state ==1 then
    if os.time() -dt< 10  then  -- compare current time with start time ,
        out1 = 1
        out2 = Input2
    else
        out1 = 0
        out2 = 0
    end
else
    out1 = 0
    out2 = 0
end
```

For calculating elapsed time always use above technique.

After solving your logic, you need to save static data and write output signals.

Lua expression from programming in Lua 3ed written by Roberto ierusalimschy :

Lua supports the usual arithmetic operators: the binary '+' (addition), '-' (subtraction),'*' (multiplication), '/' (division), '^' (exponentiation), '%' (modulo), and the unary '-' (negation). All of them operate on real numbers. For instance, x^0.5 computes the square root of x, while x^(-1/3) computes the inverse of its cubic root.
The following rule defines the modulo operator:
a % b == a - math.floor(a/b)*b For integer operands, it has the usual meaning, with the result always having the same sign as the second argument. For real operands, it has some extra uses. For instance, x%1 is the fractional part of x, and so x-x%1 is its integer part. Similarly, x-x%0.01 is x with exactly two decimal digits:

```
x = math.pi
print(x - x%0.01) --> 3.14
```

Lua provides the following relational operators:
< > <= >= == ~=
All these operators always produce a boolean value.

The == operator tests for equality; the ~= operator is the negation of equality. We can apply both operators to any two values. If the values have different types, Lua considers them not equal. Otherwise, Lua compares them according to their types. Specifically, nil is equal only to itself.

The logical operators are **and**, **or**, and **not**. Like control structures, all logical operators consider both the boolean **false** and nil as false, and anything else as true. The **and** operator returns its first argument if it is false; otherwise, it returns its second argument. The **or** operator returns its first argument if it is not false; otherwise, it returns its second argument:

```
print(4 and 5) --> 5
print(nil and 13) --> nil
print(false and 13) --> false
print(4 or 5) --> 4
print(false or 5) --> 5
```

Both **and** and **or** use short-cut evaluation, that is, they evaluate their second operand only when necessary. Short-cut evaluation ensures that expressions like (type(v)=="table"and v.tag=="h1") do not cause run-time errors: Lua will not try to evaluate v.tag when v is not a table.

A useful Lua idiom is x=x or v, which is equivalent to if not x then x = v end That is, it sets x to a default value v when x is not set (provided that x is not set to **false**).

Another useful idiom is (a and b)or c, or simply a and b or c, because **and** has a higher precedence than **or**. It is equivalent to the C expression a?b:c, provided that b is not false. For instance, we can select the maximum of two numbers x and y with a statement like max = (x > y) and x or y When x>y, the first expression of the **and** is true, so the **and** results in its second expression (x), which is always true (because it is a number), and then the **or** expression results in the value of its first expression, x. When x>y is false, the **and** expression is false and so the **or** results in its second expression, y.
The **not** operator always returns a boolean value:

```
print(not nil) --> true
print(not false) --> true
print(not 0) --> false
print(not not 1) --> true
print(not not nil) --> false
```

Lua denotes the string concatenation operator by .. (two dots). If any operand is a number, Lua converts this number to a string.

Operator precedence in Lua follows the table below, from the higher to the lower priority:
^
not # - (unary)
* / %
+ -
..
< > <= >= ~= ==
and
or

All binary operators are left associative, except for '^' (exponentiation) and '..' (concatenation), which are right associative. Therefore, the following expressions on the left are equivalent to those on the right:
a+i < b/2+1 <--> (a+i) < ((b/2)+1)
5+x^2*8 <--> 5+((x^2)*8)
a < y and y <= z <--> (a < y) and (y <= z)
-x^2 <--> -(x^2)
x^y^z <--> x^(y^z)

Assignment is the basic means of changing the value of a variable or a table field:
a = "hello" .. "world"
t.n = t.n + 1

Lua allows multiple assignment, which assigns a list of values to a list of variables in one step. Both lists have their elements separated by commas. For instance, in the assignment
a, b = 10, 2*x

if then else

An **if** statement tests its condition and executes its then-part or its else-part accordingly. The else-part is optional.

```
if a < 0 then a = 0 end
if a < b then return a else return b end
if line > MAXLINES then
  showpage()
  line = 0
end
```

while

As the name implies, a **while** loop repeats its body while a condition is true. As usual, Lua first tests the **while** condition; if the condition is false, then the loop ends; otherwise, Lua executes the body of the loop and repeats the process.

```
local i = 1
while a[i] do
 print(a[i])
 i = i + 1
end
```

Numeric for

The **for** statement has two variants: the numeric **for** and the generic **for**.
A numeric **for** has the following syntax:

```
for var = exp1, exp2, exp3 do
        <something>
end
```

This loop will execute something for each value of var from exp1 to exp2, using exp3 as the step to increment var. This third expression is optional; when absent, Lua assumes 1 as the step value. As typical examples of such loops, we have

```
for i = 1, f(x) do print(i) end

for i = 10, 1, -1 do print(i) end
```

If you want a loop without an upper limit, you can use the constant math.huge:

```
for i = 1, math.huge do
        if (0.3*i^3 - 20*i^2 - 500 >= 0) then
                print(i)
                break
        end
end
```

11 – Runtime Kernel for Linux and transferring License to Controller

pbsSoftLogic has two parts :

1 - Engineering station. Running on windows Operating system

2 – Runtime Engine. Running on Embedded Linux 2.6.x on controllers (IA240, UC7112Plus, W406 …) and runtime version for WinCE controllers(ECU-1911 ,UNO-1019 and UC-7122)

In this section we will talk about Linux Runtime engine.

You can download latest Linux runtime engine for different MOXA controllers from http://www.pbscontrol.com/download.html page.

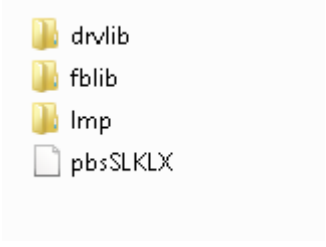
pbsSoftLogic Runtime Engine for Linux has following format :

- It locates at /home/pbsLX directory
- /home/pbsLX/pbsSLKLX file is main runtime module. It is an executable Linux file.
- /home/pbsLX/lmp/libpbsLMP.so logic monitoring protocol implementation for linux .
- /home/pbsLX/fblib/libCounters.so , libLogic.so , libMath.so , libProcess.so , libTimers.so linux implementation of pbsSoftlogic internal Function blocks . For each FB group there is one linux dynamic library .
- /home/pbsLX/drvlib/mmix/libpbsModbusMLx.so pbsSoftLogic Modbus Master(RTU/TCP) implementation for linux .
- /home/pbsLX/drvlib/msix/libpbsModbusSLX.so pbsSoftLogic Modbus Slave implementation for linux .

uc7112.rar : Runtime kernel for IA240-IA241-W341-W321-UC7112Plus-IA3341

w406.rar : Runtime kernel for IA260-IA261-IA262-EM2260

When you unzip uc7112.rar and w406.rar you can see following directories:



For transferring pbsLX directory to controller do following tasks :

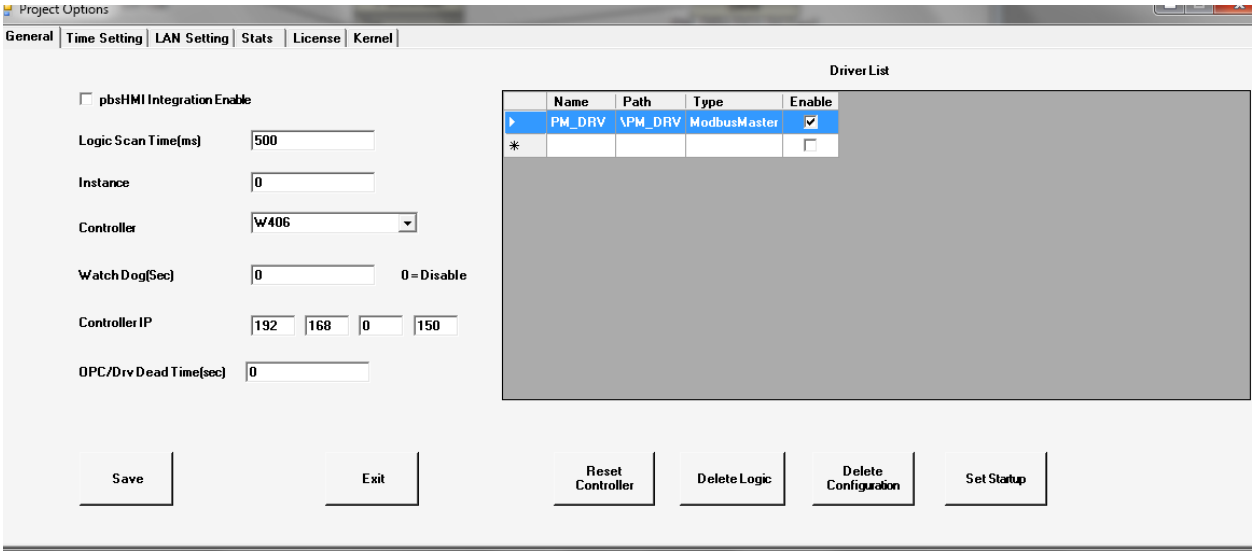
1 – Open project setting page and click on kernel Tab. To be sure that Controller IP address is correct on General Tab.

2 – Click on Browse Button and select pbsLX Directory that you want to transfer to controller. To be sure that you select correct runtime Kernel for your controller.

3 – If controller has old Runtime Kernel, first Shutdown RTU Kernel.

4 – Click on Transfer To Controller Button. It will transfer all files and directories to controller but not changing logic and configuration.

5 – If it is new controller without any kernel, in General Tab click on Set Startup Button to put all necessary modules in controller startup path.



6 – From general tab Restart Controller.

For each controller, you need to have license file for life time operation. Without License, it will work for 30 Min and you need to restart Controller.

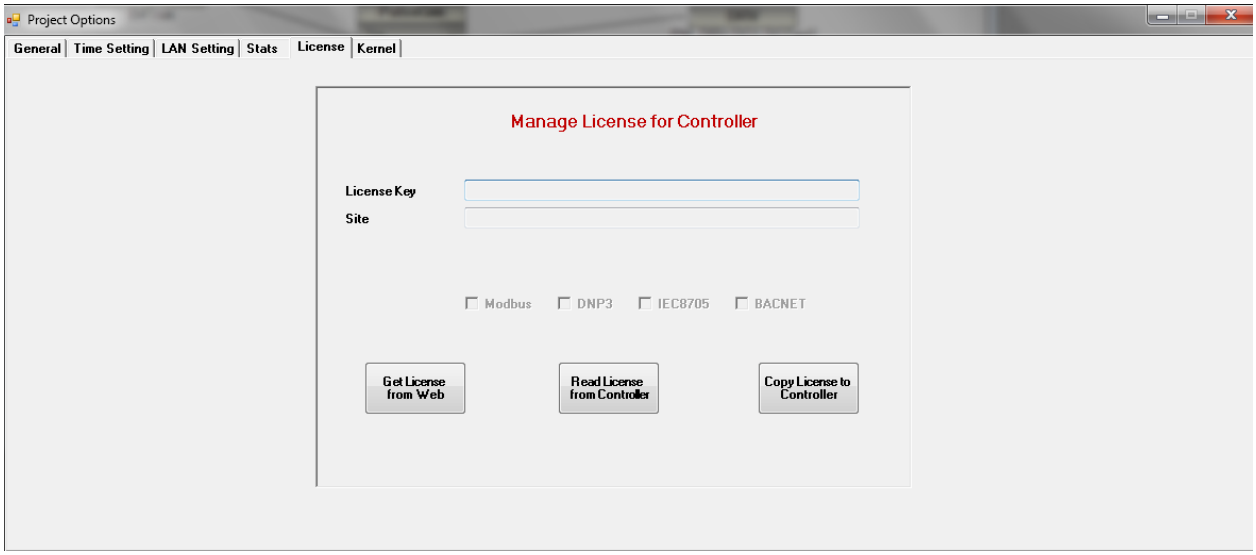
We have following license for controller runtime:

- RTU/PLC functionality and Modbus Master/Slave protocol. This is basic license for each controller.
- DNP3 Slave License.
- IEC870-5-101/104 Slave License
- BACNET License

You need to purchase each license separately from your supplier or directly through www.pbscontrol.com web site. You can purchase basic license and purchase other license for your controller. But your license key is same for each controller.

When you purchase pbsSoftLogic License, you will receive a license key. For activating license do following steps:

1 – Open project setting page and select License Tab.



2 – To be sure that your PC is connected to Internet and Controller In the same time.

3 – Copy and Paste Controller license Key to License key text Box .

4 – Write some description about your project, Project name, country,

5 – Click on get License from Web Site It will connect to pbscontrol web site and get all purchase licenses

6 – Modbus , DNP3 , IEC and BACNet check boxs will be checked based on your purchase

7 – Click on Copy License to Controller. It will move license file to controller.

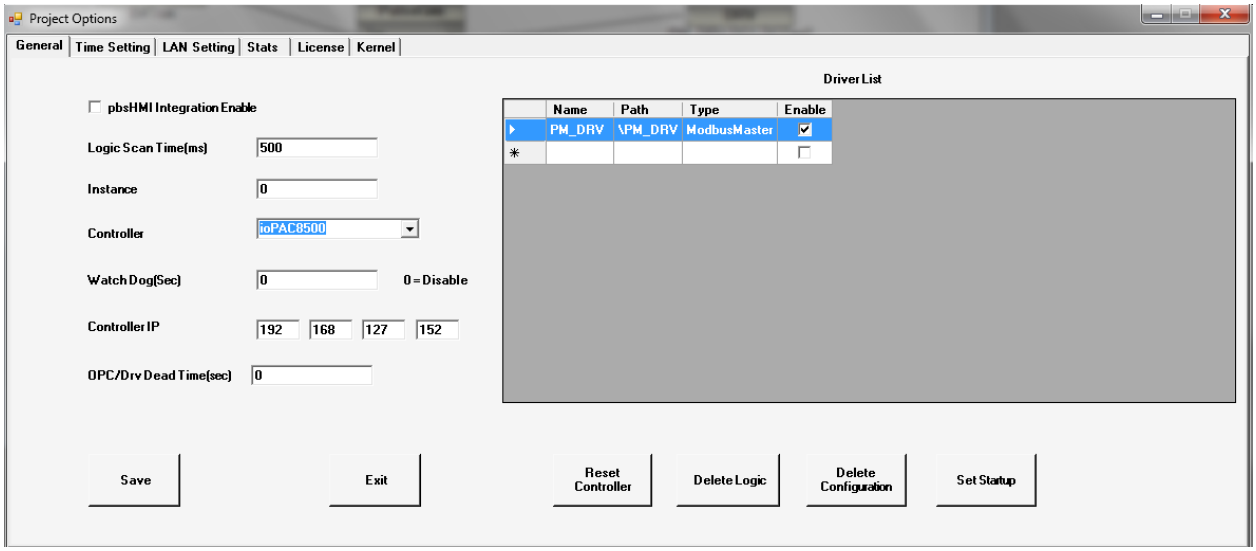
8 – From General tab, restart Controller.

If you have a controller and want to check its license, click on Read License from Controller.

12 – Project Settings facilities

There are many facilities in setting page in pbsSoftLogic Editor.

Open Setting Page you can see following tabs:



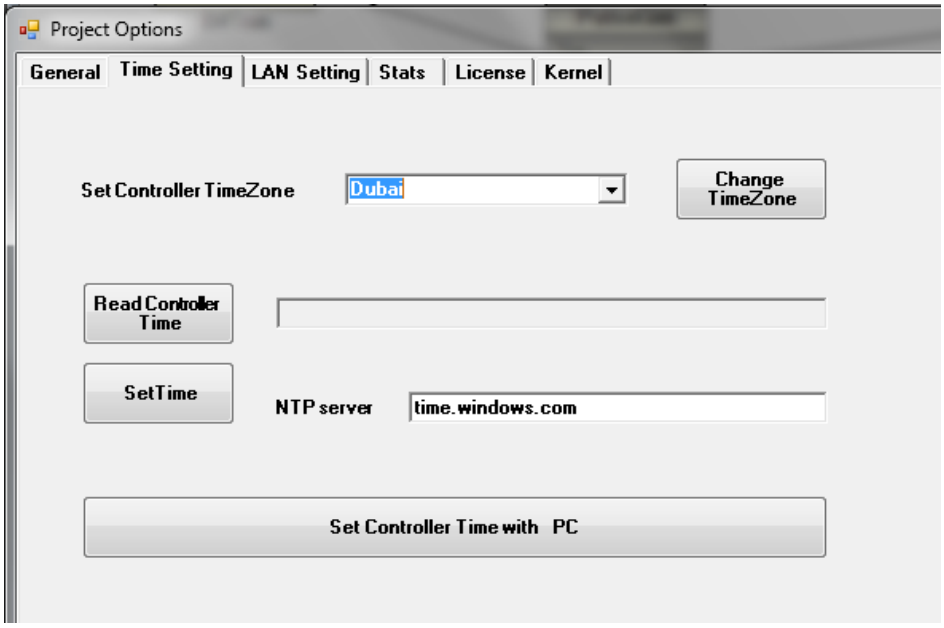
- General
- Time Setting
- LAN Setting
- Stats
- License
- Kernel

General Tab: In This page you can set following parameters:

- Logic Scan time (Msec)
- Controller Type

- Watch Dog Value in sec , if Value is 0 , DWT is disabled
- Controller IP address
- Communication Drivers
- Restart Controller
- Delete Logic
- Delete Configuration
- Set Startup: will set all necessary modules in Startup path of controller. For a new controller before running any Commands in setting page, you need to set Startup and restart controller manually.
-

Time Setting:



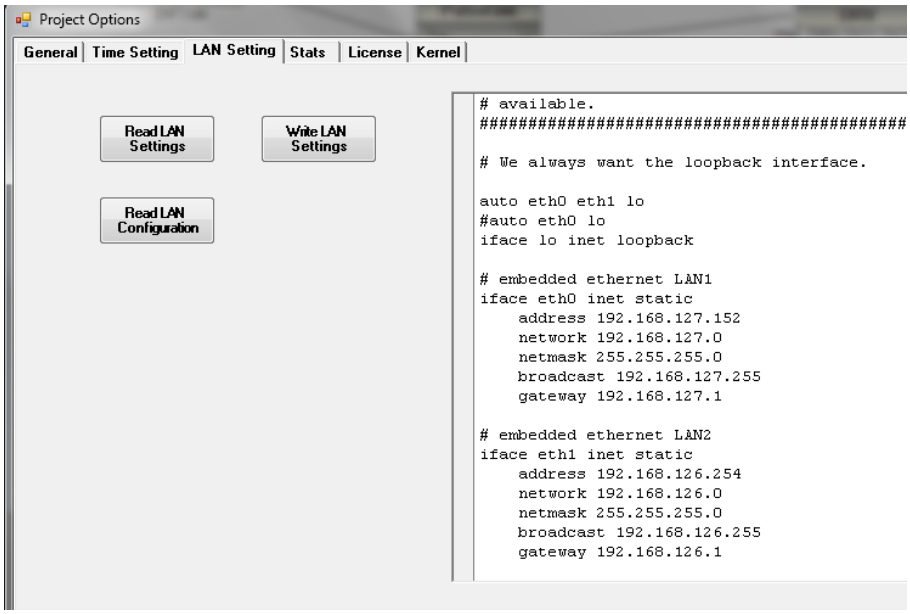
Set Controller Time Zone: Select your location from list box, and click on Change TimeZone .

Read Controller Time : Will read current Date time and time Zone of controller .

Set Time : will set Controleller time from NTP Server , it can be a computer on the network or any Time web site . But controller should connect to Internet.

Set Controller Time with PC : It will set Controller time from PC that is running pbsSoftLogic .

LAN Settings:



Read LAN Setting: It will read current LAN Setting from Controller.

Write LAN Settings: it will Write LAN settings to controller

Read LAN Configuration: it will read current ALN configuration fro controller.

For changing controller IP address:

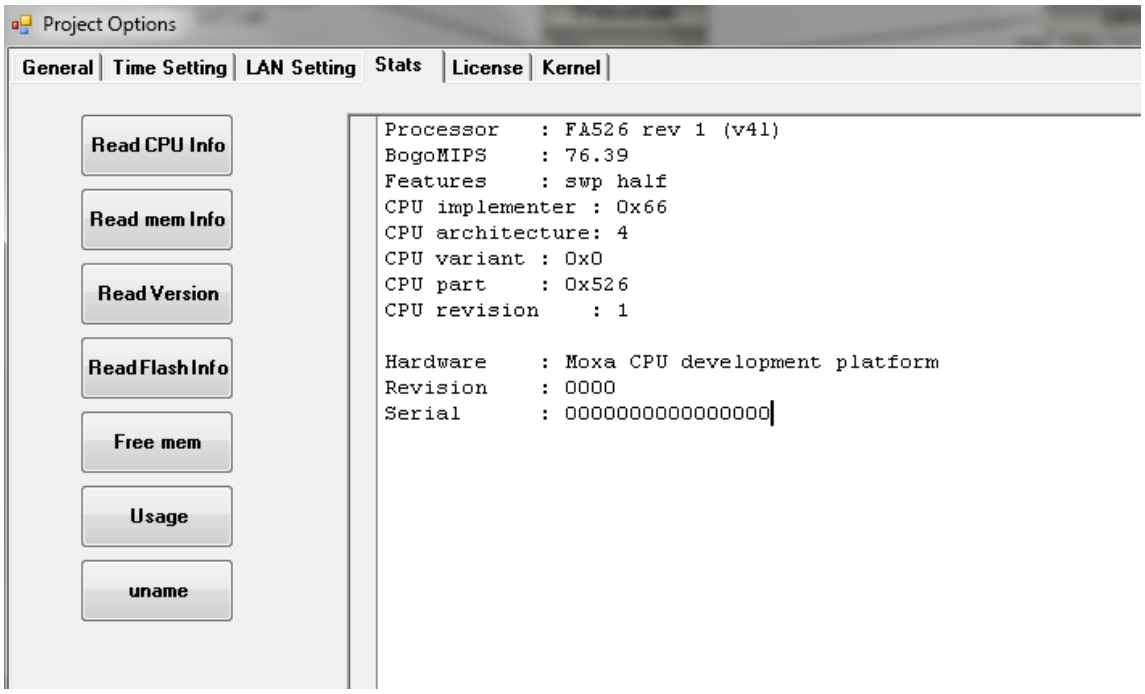
1 – Read LAN Settings

2 – Change IP address for each LAN port and other settings

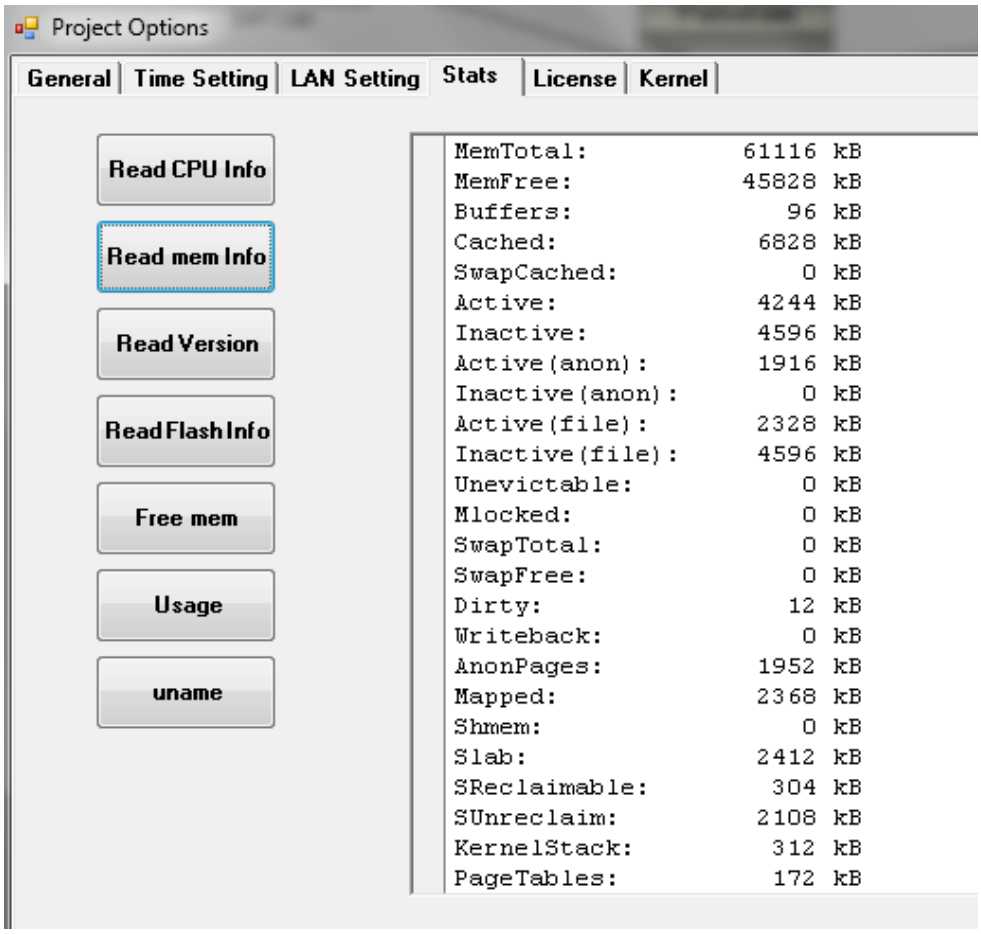
3 – Write New Settings to Controller.

Controller Stat tab:

Read CPU Information: It will Read Hardware Information from controller

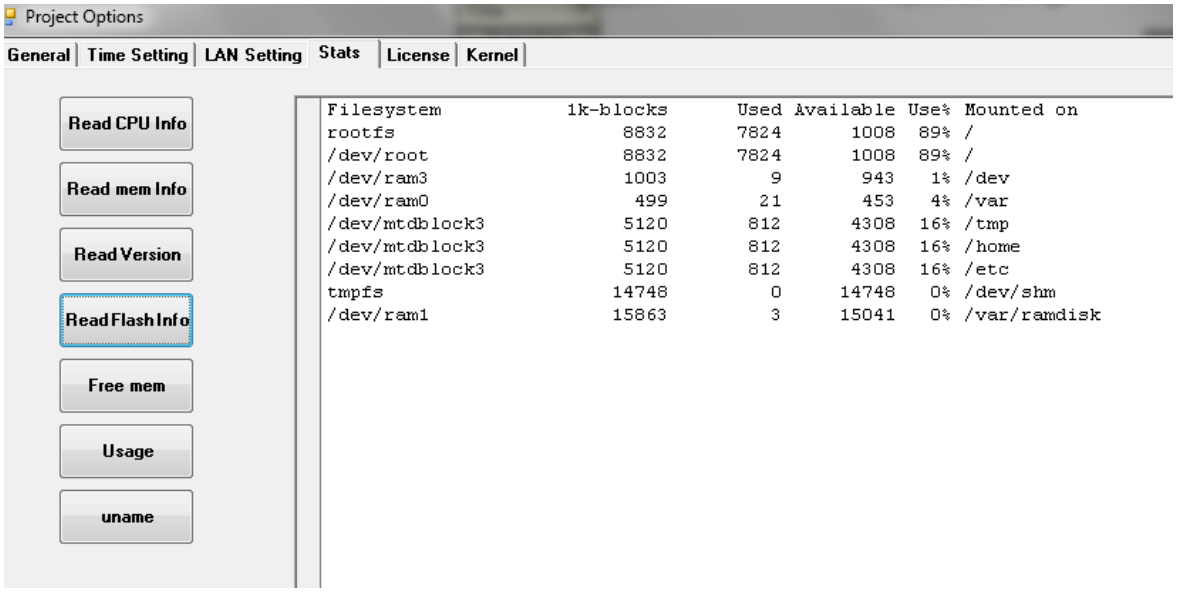


Read memory information: shows detail of memory usage of controller

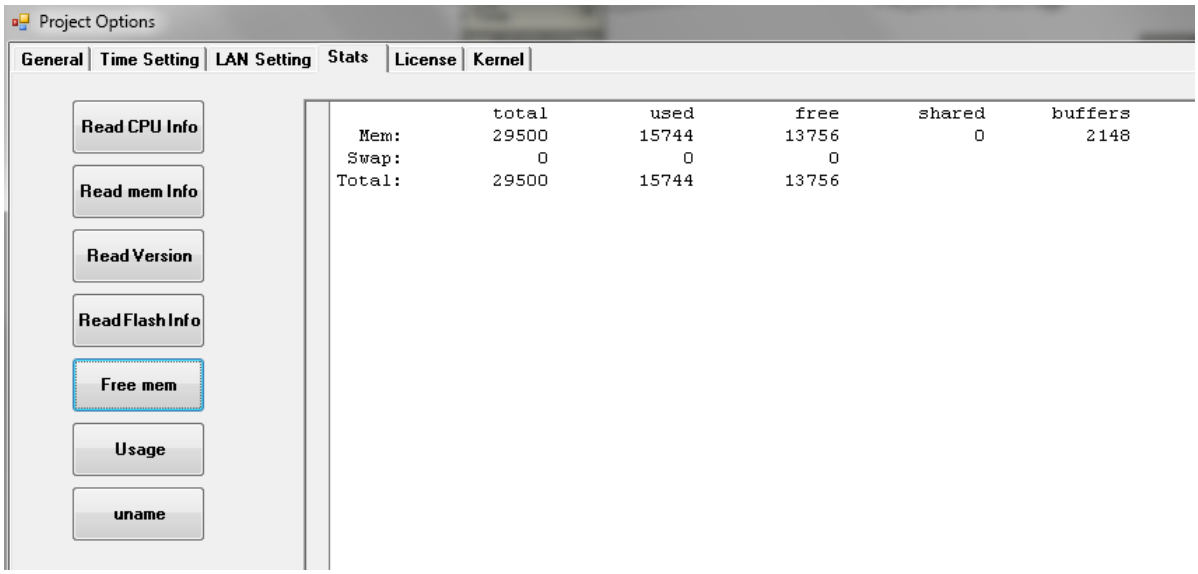


Read Version: Read Controller Linux Version, GCC compiler version

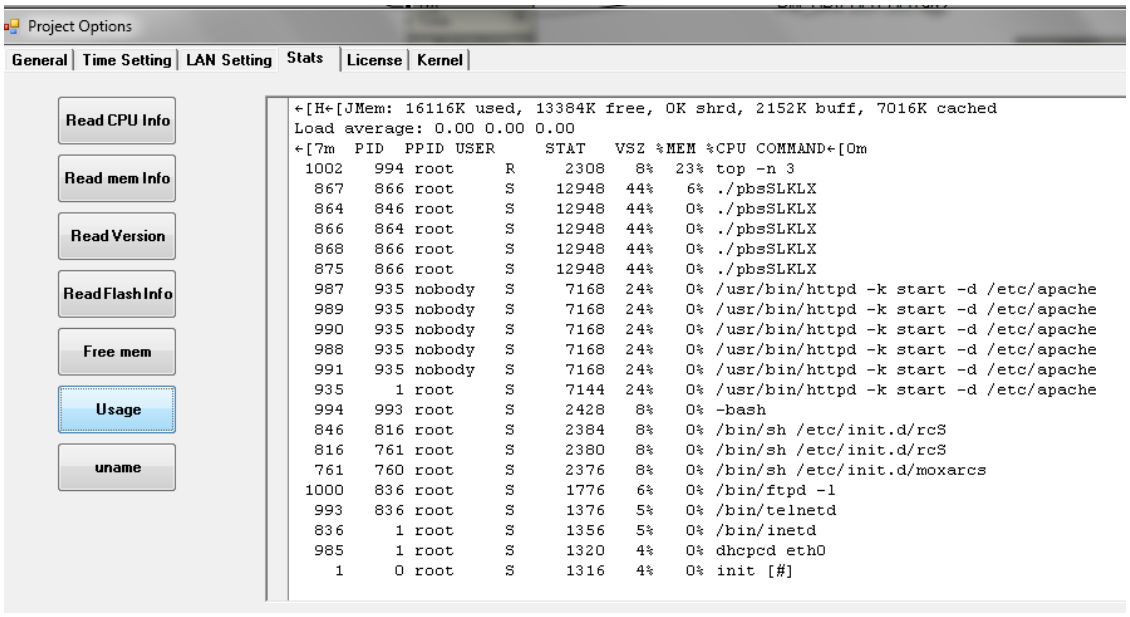
Read Flash Information:



Read Free Memory:



Usage: this is equal to top command in linux .



13 – ECU-1911 Local I/O Definition

ECU-1911 is one of our main controllers for SCADA projects. ECU-1911 has following specifications:

General

☐ **Power Consumption:** <10 W (Typical)

☐ **Power Requirements:** 24 VDC (Typical) (10 Min ~ 30 Max VDC)

☐ **OS Support:** Windows CE 5.0

System Hardware

☐ **CPU:** Xscale @ PXA-270 520MHz

☐ **Memory:** Onboard 64 MB SDRAM/ 32 MB Flash

☐ **Storage:** 1 x type I/II Compact Flash slot (Support FAT16 and UP TO 2 GB)

Digital Input

☐ **Channels:** 32

☐ **I/O Type:** Sink

☐ **Wet Contact:**

– Logic 0: 0 ~ 10 V
– Logic 1: 19 ~ 30 V

☐ **Isolation:** 3000 VDC

☐ **Connector:** Terminal Block (#14 ~ 22 AWG)

Digital Output

☐ **Channels:** 32

☐ **I/O Type:** Power Relay Form A

☐ **Contact Rating:**

– AC: 5A @ 250 V;
– DC: 5 A @ 30 V (Resistive Load)

☐ **Isolation:** 500 VDC

☐ **Connector:** Terminal Block (#14 ~ 22 AWG)

Analog Input

☐ **Channels:** 8 differential

☐ **Resolution:** 16 bits

☐ **Sampling rate:** 10 Hz/sec (total)

☐ **Input Impedance:** Voltage: 20 MΩ Current: 120 Ω (Build-in 120 Ω. for Current)

☐ **Input Range:** 0 ~ 150 mV, 0 ~ 500 mV, 0 ~ 1 V, 0 ~ 5 V, 0 ~ 10 V, 0 ~ 15 V, ±150 mV, ±500 mV, ±1 V, ±5 V, ±10 V, ±15 V, ±20 mA, 4 ~ 20 mA

Environment

☐ **Humidity:** 5 ~ 95% @ 40°C (non-condensing)

☐ **Operating Temperature:** -20 ~ 70°C (-4 ~158°F) @ 5 ~ 85% RH

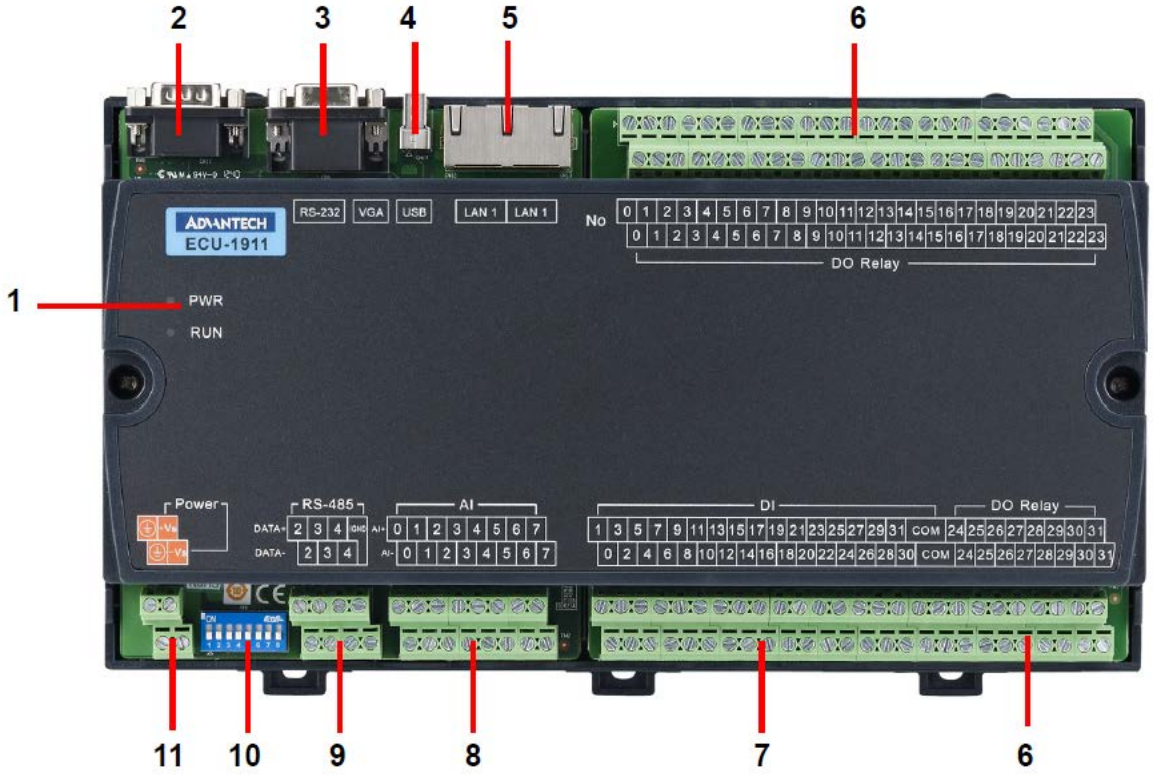
☐ **Storage Temperature:** -40 ~ 80°C (-40 ~176°F)

I/O Interface

☐ **Serial Ports:** 1 x RS-232 with DB9 (RTS,CTS,TX,RX); 3 x RS-485 with Terminal Block connector, Automatic RS-485 data flow

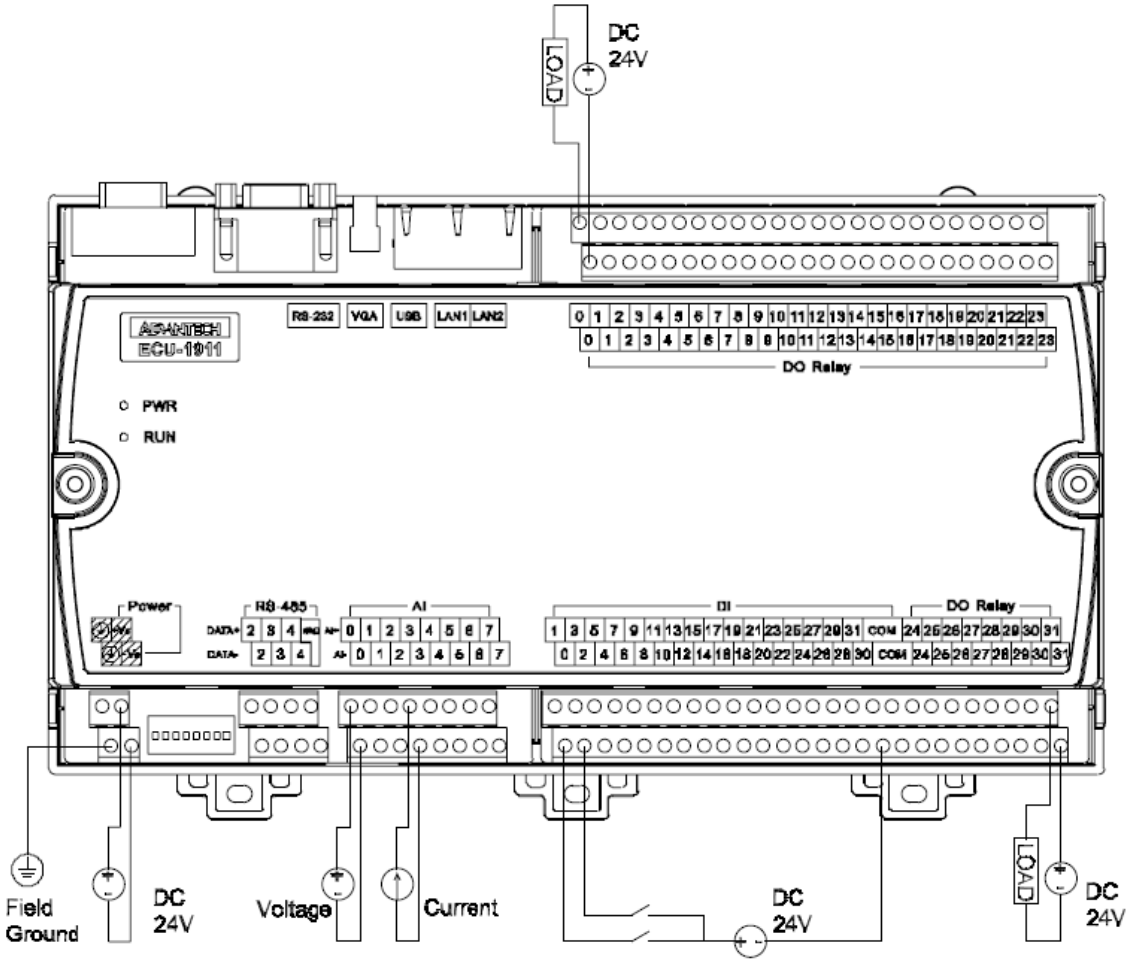
☐ **LAN:** 2 x 10/100Base-T RJ-45 ports

☐ **USB Port:** 1 x USB, OpenHCI, Rev. 1.1 compliant



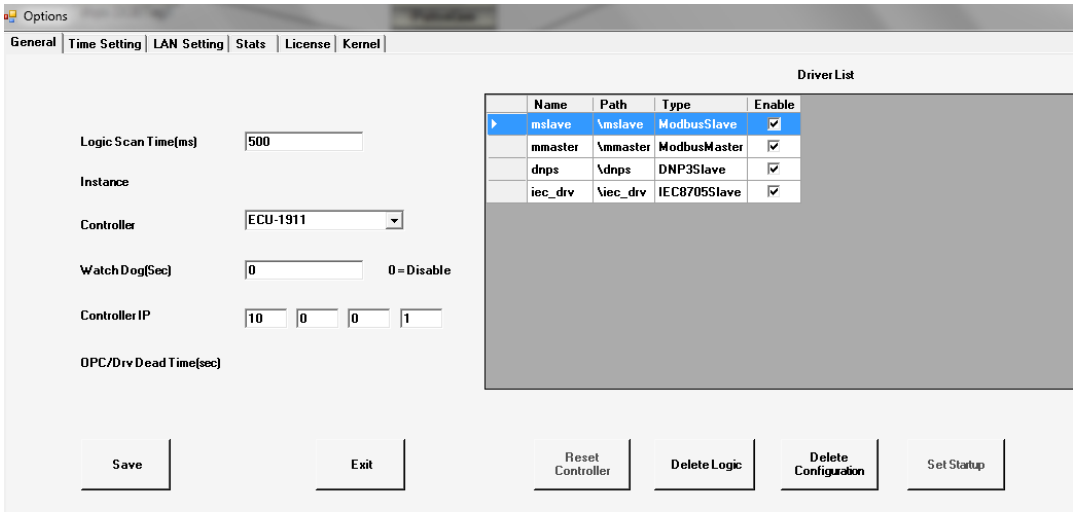
Item	Description
1	LED
2	RS-232 Serial Port
3	VGA port
4	USB port
5	Networking port
6	Digital output
7	Digital input
8	Analog input
9	RS-485 Serial Port
10	Switch
11	Power

ECU-1911 Field wiring

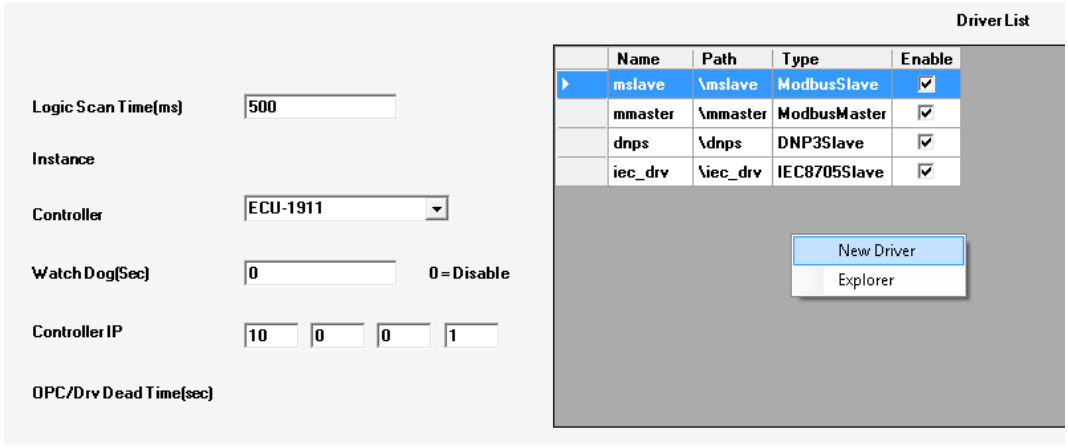


Defining ECU-1911 Local I/O in pbsSoftLogic

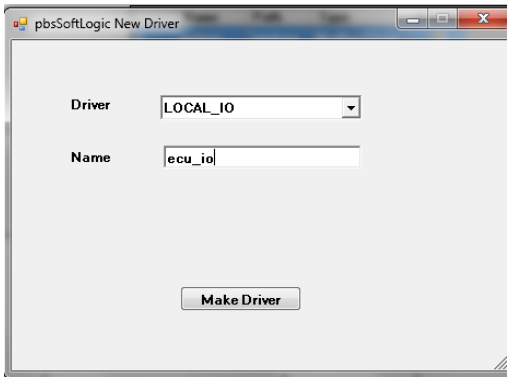
1 – At project setting select ECU-1911 as controller type.



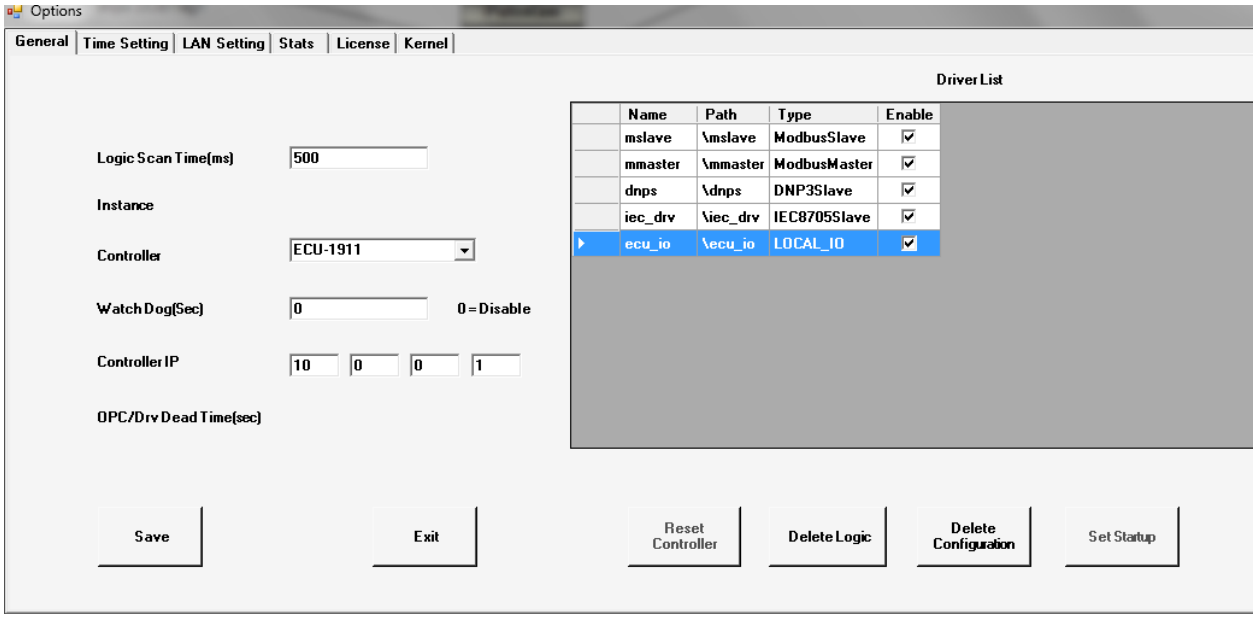
2 – Right click at driver list and select new Driver



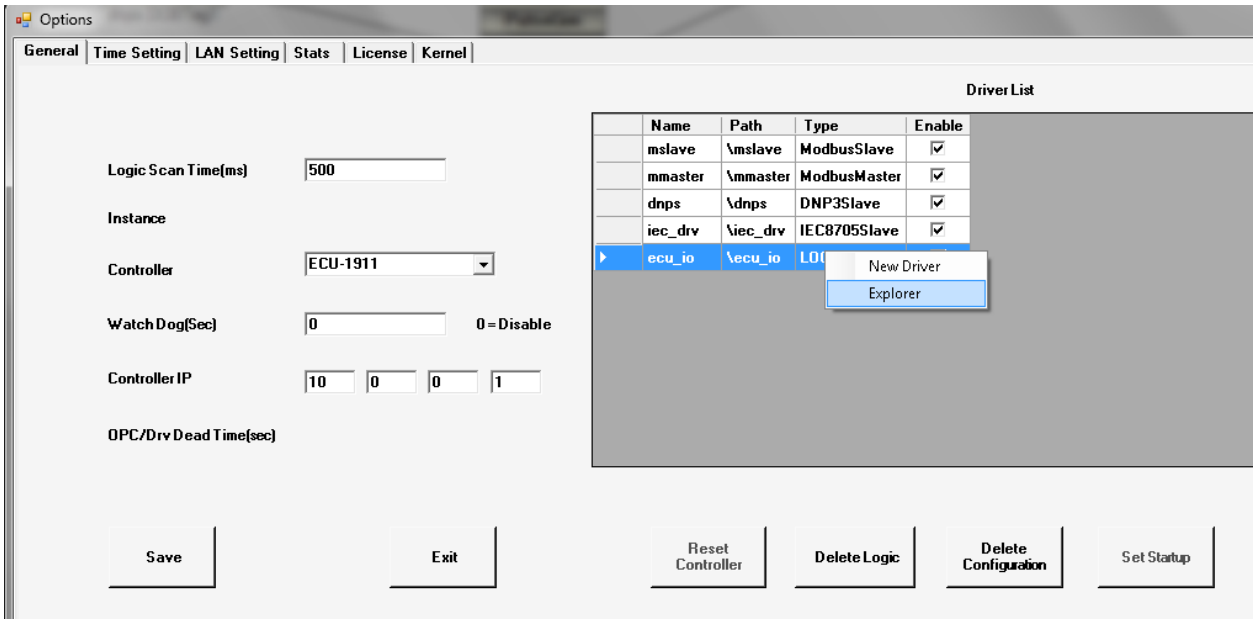
3 – Select New Driver and select LOCAL_IO from list. Write a Unique name for Driver. Click on Make Driver.



4 – New Local I/O driver is adding to list of drivers.



5 – Click on LOCAL_IO driver and right click on it.



6 – Select explorer. Edit Local_IO.xml file. You can see a xml file for all ECU-1911 I/Os. You can change Tag Name based on your logic and project.

Digital Signal Definition:

```
<Tag Name="DITag0" Type="DI" Init="0" Address="0" />
<Tag Name="DITag1" Type="DI" Init="0" Address="1" />
<Tag Name="DITag2" Type="DI" Init="0" Address="2" />
<Tag Name="DITag3" Type="DI" Init="0" Address="3" />
<Tag Name="DITag4" Type="DI" Init="0" Address="4" />
<Tag Name="DITag5" Type="DI" Init="0" Address="5" />
<Tag Name="DITag6" Type="DI" Init="0" Address="6" />
<Tag Name="DITag7" Type="DI" Init="0" Address="7" />
<Tag Name="DITag8" Type="DI" Init="0" Address="8" />
<Tag Name="DITag9" Type="DI" Init="0" Address="9" />
<Tag Name="DITag10" Type="DI" Init="0" Address="10" />
<Tag Name="DITag11" Type="DI" Init="0" Address="11" />
<Tag Name="DITag12" Type="DI" Init="0" Address="12" />
<Tag Name="DITag13" Type="DI" Init="0" Address="13" />
<Tag Name="DITag14" Type="DI" Init="0" Address="14" />
<Tag Name="DITag15" Type="DI" Init="0" Address="15" />
<Tag Name="DITag16" Type="DI" Init="0" Address="16" />
<Tag Name="DITag17" Type="DI" Init="0" Address="17" />
<Tag Name="DITag18" Type="DI" Init="0" Address="18" />
<Tag Name="DITag19" Type="DI" Init="0" Address="19" />
<Tag Name="DITag20" Type="DI" Init="0" Address="20" />
<Tag Name="DITag21" Type="DI" Init="0" Address="21" />
<Tag Name="DITag22" Type="DI" Init="0" Address="22" />
<Tag Name="DITag23" Type="DI" Init="0" Address="23" />
<Tag Name="DITag24" Type="DI" Init="0" Address="24" />
<Tag Name="DITag25" Type="DI" Init="0" Address="25" />
<Tag Name="DITag26" Type="DI" Init="0" Address="26" />
<Tag Name="DITag27" Type="DI" Init="0" Address="27" />
<Tag Name="DITag28" Type="DI" Init="0" Address="28" />
<Tag Name="DITag29" Type="DI" Init="0" Address="29" />
<Tag Name="DITag30" Type="DI" Init="0" Address="30" />
<Tag Name="DITag31" Type="DI" Init="0" Address="31" />
```

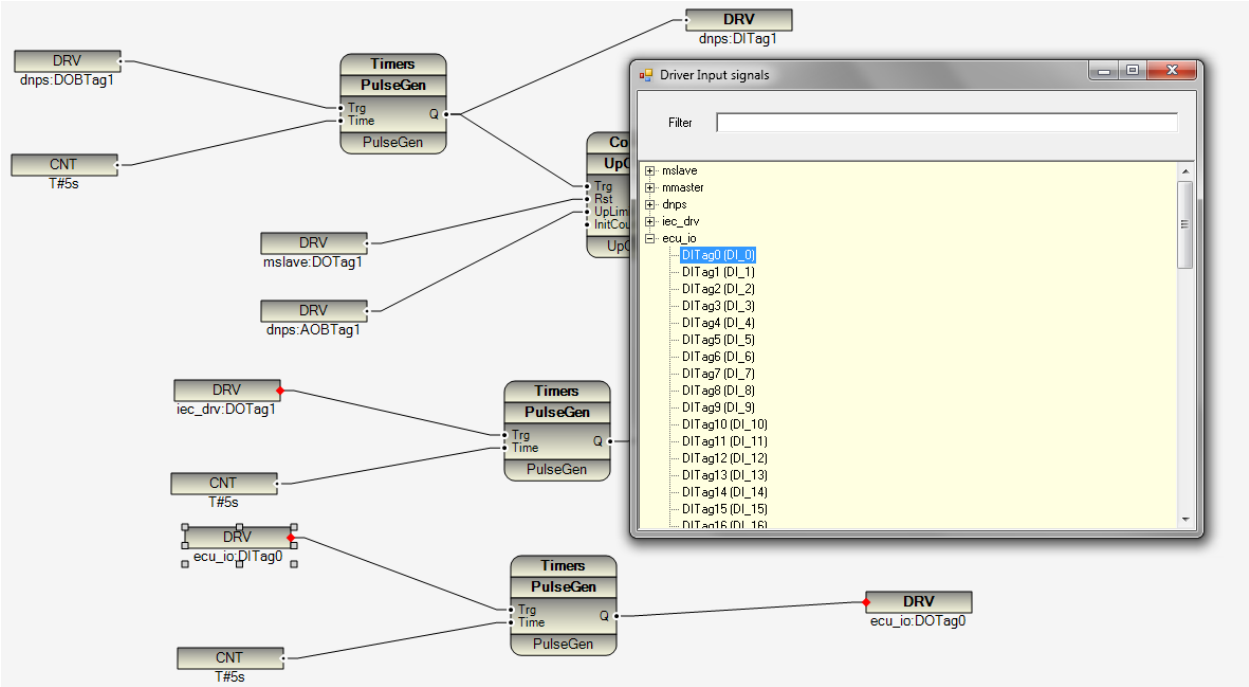
Analog Input Signal definition :

```
<Tag Name="AITag0" Type="AI" Init="0" Address="0" />
<Tag Name="AITag1" Type="AI" Init="0" Address="1" />
<Tag Name="AITag2" Type="AI" Init="0" Address="2" />
<Tag Name="AITag3" Type="AI" Init="0" Address="3" />
<Tag Name="AITag4" Type="AI" Init="0" Address="4" />
<Tag Name="AITag5" Type="AI" Init="0" Address="5" />
<Tag Name="AITag6" Type="AI" Init="0" Address="6" />
<Tag Name="AITag7" Type="AI" Init="0" Address="7" />
```

Digital Output Tag Definition :

```
<Tag Name="DOTag0" Type="DO" Init="0" Address="0" />
<Tag Name="DOTag1" Type="DO" Init="0" Address="1" />
<Tag Name="DOTag2" Type="DO" Init="0" Address="2" />
<Tag Name="DOTag3" Type="DO" Init="0" Address="3" />
<Tag Name="DOTag4" Type="DO" Init="0" Address="4" />
<Tag Name="DOTag5" Type="DO" Init="0" Address="5" />
<Tag Name="DOTag6" Type="DO" Init="0" Address="6" />
<Tag Name="DOTag7" Type="DO" Init="0" Address="7" />
<Tag Name="DOTag8" Type="DO" Init="0" Address="8" />
<Tag Name="DOTag9" Type="DO" Init="0" Address="9" />
<Tag Name="DOTag10" Type="DO" Init="0" Address="10" />
<Tag Name="DOTag11" Type="DO" Init="0" Address="11" />
<Tag Name="DOTag12" Type="DO" Init="0" Address="12" />
<Tag Name="DOTag13" Type="DO" Init="0" Address="13" />
<Tag Name="DOTag14" Type="DO" Init="0" Address="14" />
<Tag Name="DOTag15" Type="DO" Init="0" Address="15" />
<Tag Name="DOTag16" Type="DO" Init="0" Address="16" />
<Tag Name="DOTag17" Type="DO" Init="0" Address="17" />
<Tag Name="DOTag18" Type="DO" Init="0" Address="18" />
<Tag Name="DOTag19" Type="DO" Init="0" Address="19" />
<Tag Name="DOTag20" Type="DO" Init="0" Address="20" />
<Tag Name="DOTag21" Type="DO" Init="0" Address="21" />
<Tag Name="DOTag22" Type="DO" Init="0" Address="22" />
<Tag Name="DOTag23" Type="DO" Init="0" Address="23" />
<Tag Name="DOTag24" Type="DO" Init="0" Address="24" />
<Tag Name="DOTag25" Type="DO" Init="0" Address="25" />
<Tag Name="DOTag26" Type="DO" Init="0" Address="26" />
<Tag Name="DOTag27" Type="DO" Init="0" Address="27" />
<Tag Name="DOTag28" Type="DO" Init="0" Address="28" />
<Tag Name="DOTag29" Type="DO" Init="0" Address="29" />
<Tag Name="DOTag30" Type="DO" Init="0" Address="30" />
<Tag Name="DOTag31" Type="DO" Init="0" Address="31" />
```

7 – Use Local I/O in your logic by opening driver list and select signals.



14 – OPC Client Driver Configuration for Win32 Target.

pbsSoftlogic Version 1.7 supports Windows 32 Target the same way as Linux and wince target .

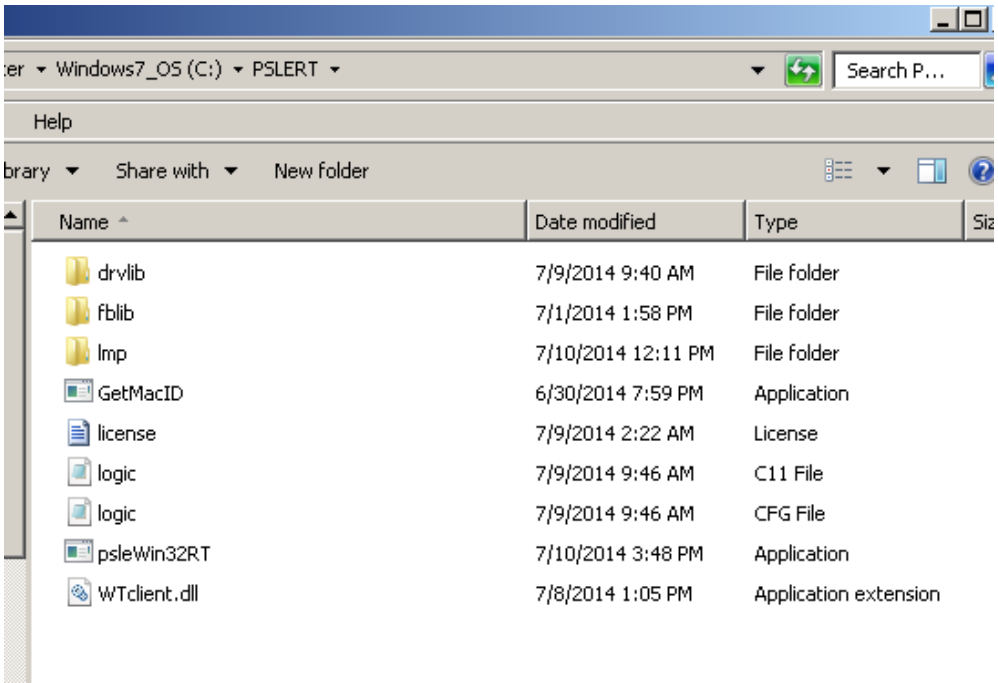
There are two windows32 runtime kernels for pbsSofLogic:

- Runtime kernel that is based on OPC standard. (VSLE.exe) we named this kernel PCWIN32 in project setting. This is pure Dot Net Kernel and is developed by C# .VSLE .exe is mostly used for Subsystem integration based on OPC technology.

- Runtime kernel that is compiled from Linux and wince kernel c source code for win32. This is high performance kernel and can be used as PLC/RTU applications on embedded Win32 controller. We named this kernel WIN32 in project setting. This part is talking about Win32 runtime and how we can use it. Win32 Kernel is just based on driver concepts and it has following drivers built in :

    - Modbus RTU/TCP master /Slave
    - DNP3 Master/Slave
    - IEC870-5-101/104 Master/Slave
    - IEC870-5-103 master
    - OPC client Driver
    - OPC server Driver
    - Open API Driver for C interfacing with runtime kernel.

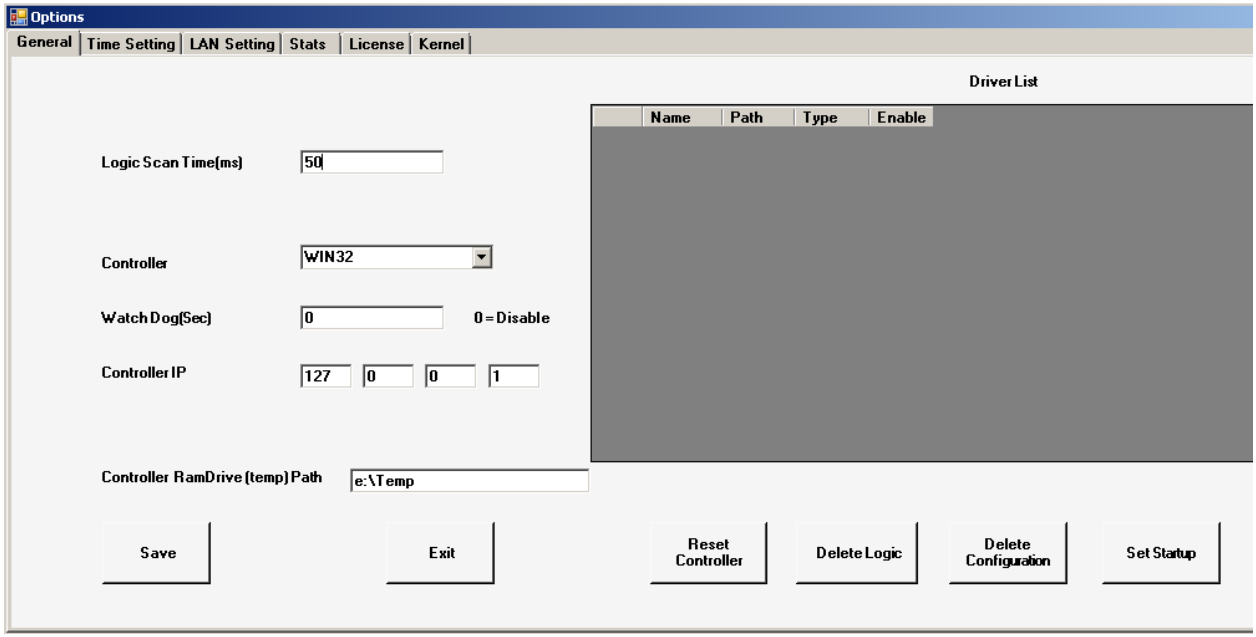
Download latest Win32 target from www.pbscontrol.com . Unzip it on any drive in your controller. Suppose we unzipped kernel on C:\PSLERT Directory.



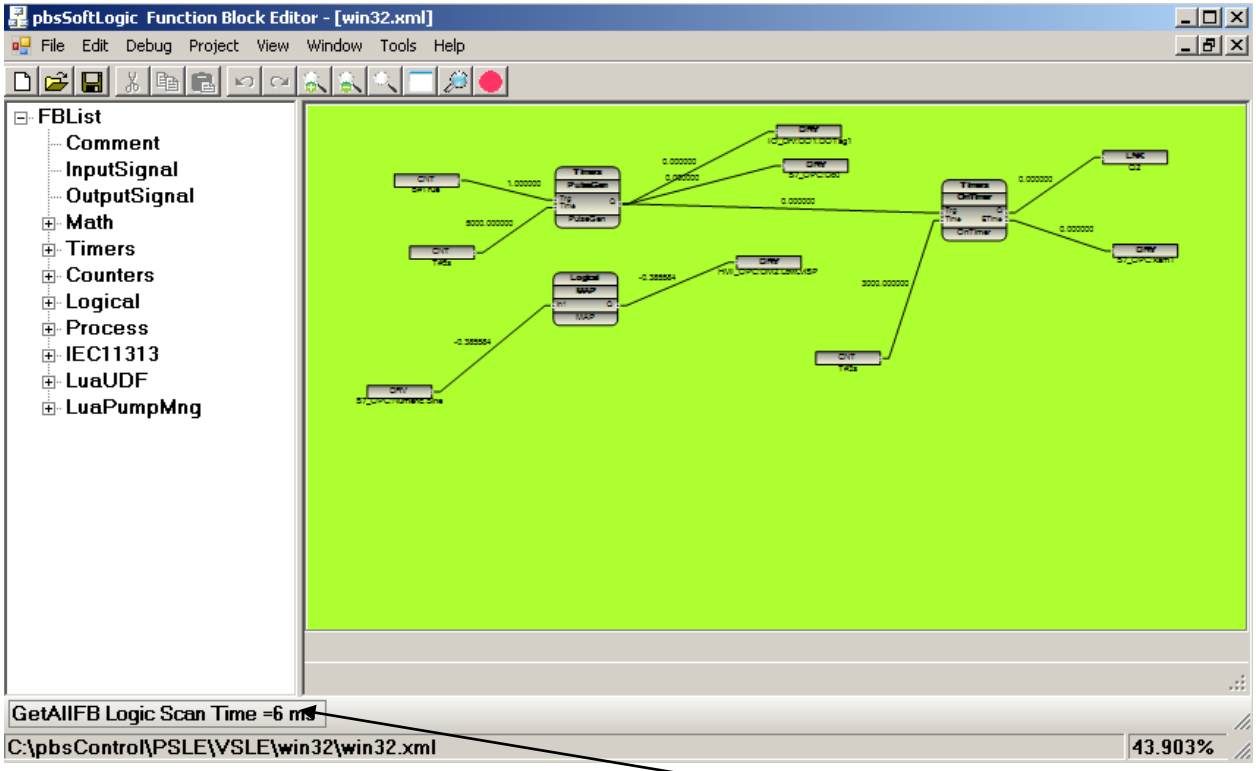
- psleWin32RT.exe is main application for kernel. It should be in Windows Auto start routine.
- Logic.c11 compiled pbsSoftLogic Logic file. Transferred by pbsSoftlogic Eng
- Logic.cfg compiled pbsSoftlogic Configuration file. Transferred by pbsSoftlogic Eng
- License.lic license file that is linked to MACID of Controller. psleWin32RT.exe will works for 30 min without License file .
- GetMacID.exe utility program for making license file. You need to run getMacID.exe and send MacID to supplier for getting permanent license file.
- Drvlib : communication driver library
- Fblib : Function block implementation library (c and Lua)
- Lmp : Logic monitoring protocol library
- WtClient.dll main dll file for OPC DA2.0 client driver.

pbsSoftlogic is using FTP for transferring logic and configuration file to Controller . So you need to install FTP server on target controller with Windows32 OS. Install FileZilla server or use internal windows FTP Server services and define “root” user with “root” password.
Set C:\PSLERT as default path of FTP server for “root” user. "root” user should has write/read access to c:\PSLERT directory.

Make a new project and set project setting as following:



Controller Type is Win32.
Logic Scan time (ms): period for reading all inputs, running logic and writing all outputs. We name this time logic scan time. When you connect and monitor logic you can see real value for logic scan time.
Logic Scan Time in settings = Real Logic Scan Time + sleep Time
Suppose you set Logic Scan time in Setting page to 50 ms, but real logic scan time is 20 ms . So kernel will sleep for 30 ms at each cycle.



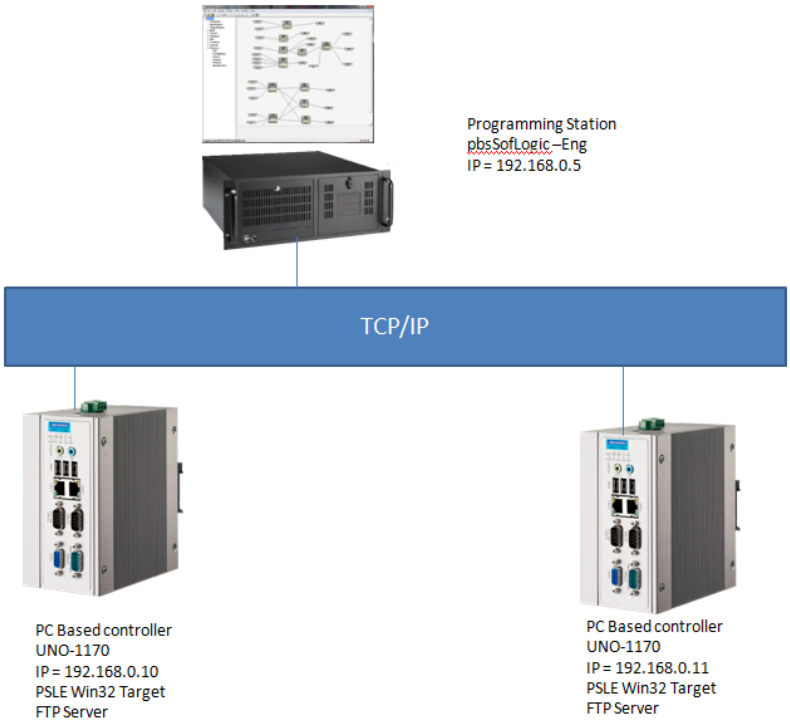
You can see real logic scan time at bottom side of logic monitoring page. In above sample, real logic scan time is 6 ms.

When you are using drivers like modbus , you need to add Modbus scan time to logic scan time to calculate real scan time of whole IO and logic .

Controller RAM Driver (Temp) Path: pbsSoftLogic runtime kernel is using files for keeping static data of Function blocks. Because at each scan runtime kernel is open, read and write static data to files, so it is too much better to use ram drive for saving static data files.

You can download very professional and free RAM Disk Driver from http://memory.dataram.com/products-and-services/software/ramdisk Web Site. We tested Data ram disk in many projects and it is 100% compatible with pbsSoftLogic .

Controller IP: you can use PC based controllers like UNO-1150 , UNO-1170 and use separate laptop for programming . Then you need to set PC Based controller IP here. When programming PC and controller PC are same, then you can use 127.0.0.1 as Controller IP.



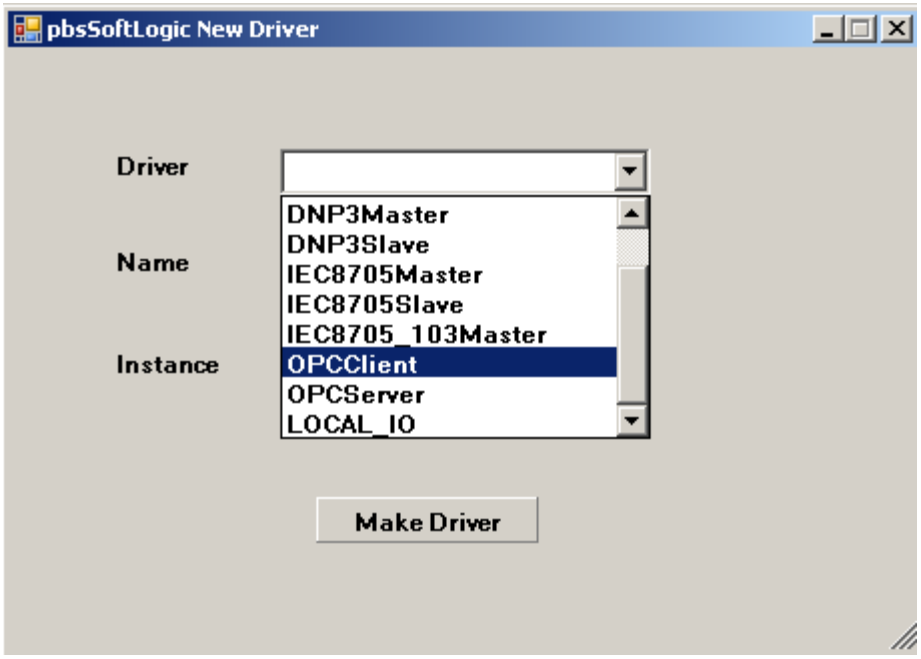
In above sample, we used two UNO-1170 as controllers and one station as programming station.

You need to make two separate project for each UNO-1170.

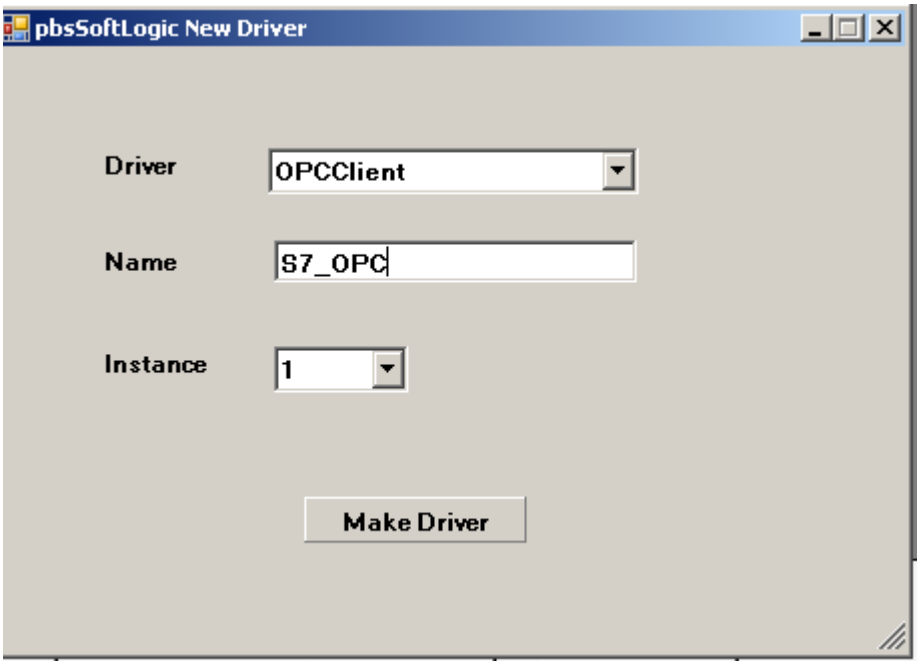
If you need to pass data between controllers, then you can define modbas-TCP master on one controller and Modbus-TCP slave on another controller. You can also use DNP3 over TCP and IEC870-5-104 for communicating between two controllers.

Defining OPC client Driver

Open project setting page and right click on Driver list, then select OPCClient Driver.



Select a unique name for driver and select driver instance. You can connect to 8 OPC server on each controller in the same time. Each OPC server connection should have unique Instace ID.

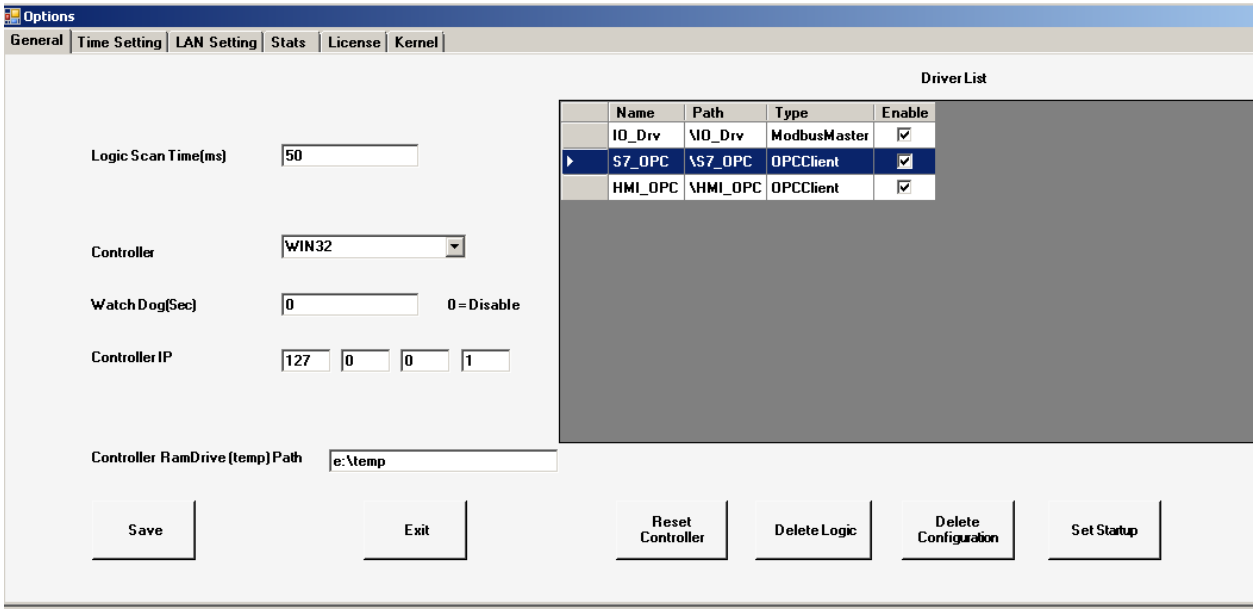


Click on “Make Driver” Button. pbsSoftLogic will make basic definition in your project .

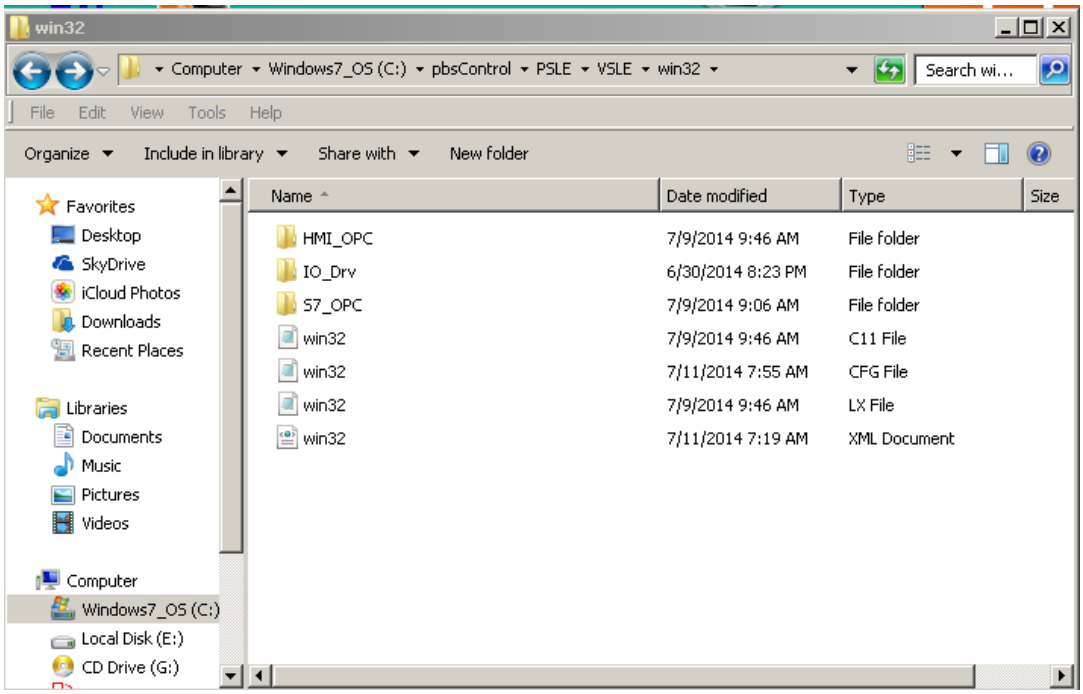
OPC client Driver : pbsSoftLogic runtime kernel will connect to other OPC servers DA2.0

OPC Server Driver : pbsSoftLogic runtime kernel will act as OPC Server and other client can connect to it.

In this part, we will talk about OPC Client Driver.

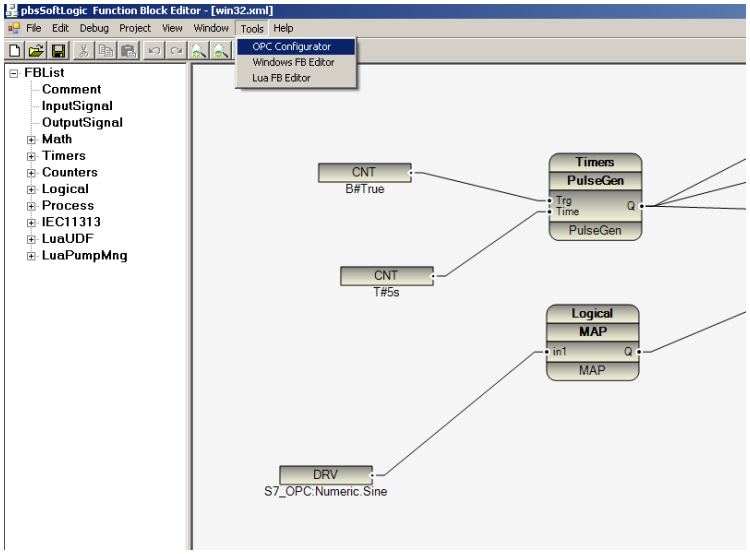


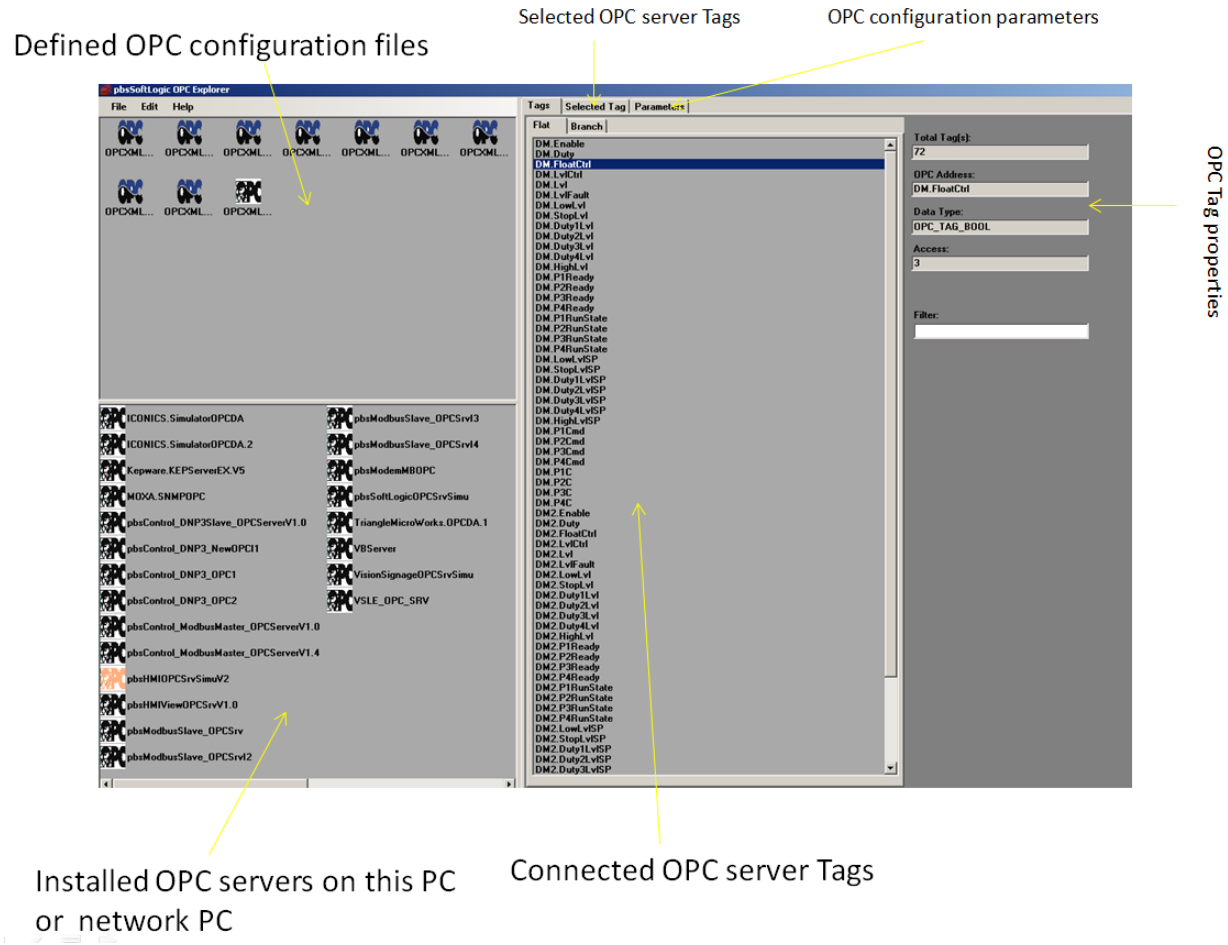
pbsSoftLogic will make a new folder in project directory with same driver name .



Inside S7_OPC directory , you can see OPCTags.xml file . we will keep all parameters and tags inside OPCTags.xml file .

For making OPCTags.xml you should use pbsSoftlogic OPC configurator utility at Tools menu .

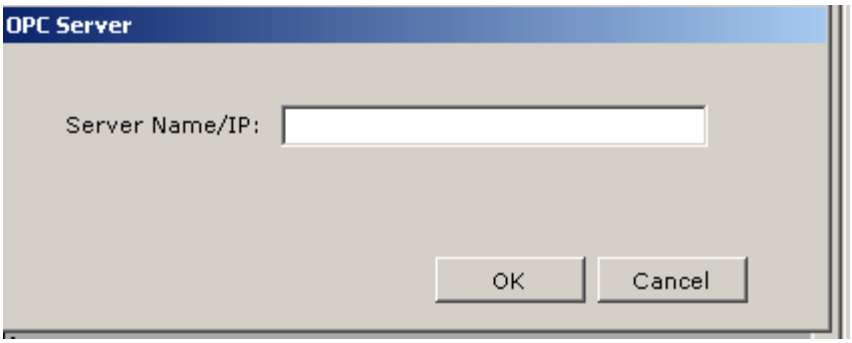




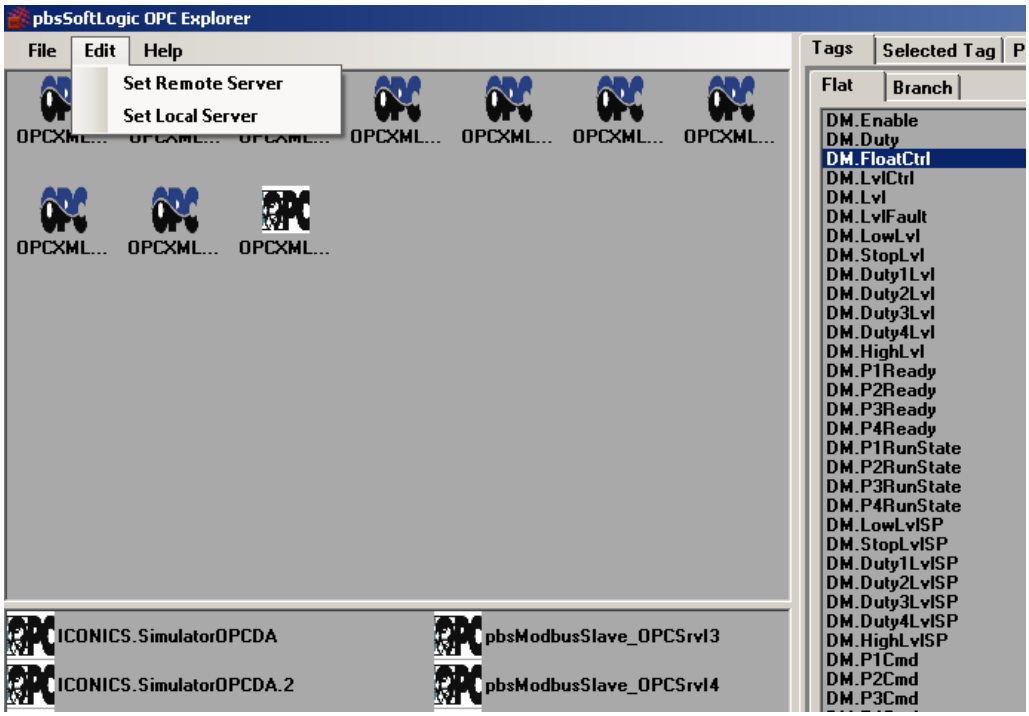
Defined OPC configuration files: At Top left panel you can see all defined OPC configuration files. These files are located at \PSLE\OPC folder.

Installed OPC Servers: At Bottom left panel you can see all installed OPC servers on this machine or remote PC. For browsing OPC servers on remote machine you need to do all setting for OPC on network for both PC . OPC network operation is dependent too much on Operating systems and it is out of scope for this document.

For connecting to remote PC , at Edit menu select “Set remote Server” then type Server Name of IP address for getting all installed OPC servers on that Machine .

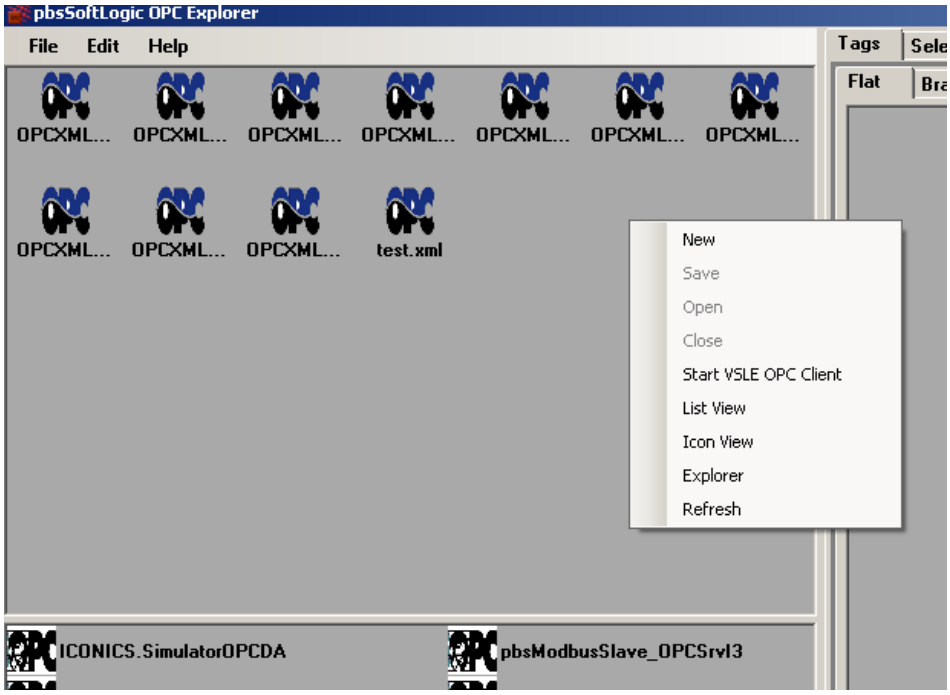


For changing to local Machine , from Edit menu select “Set Local Server”.



For defining new OPC Configuration file , right click on “Defined OPC Configuration” panel .

And select “new” menu.

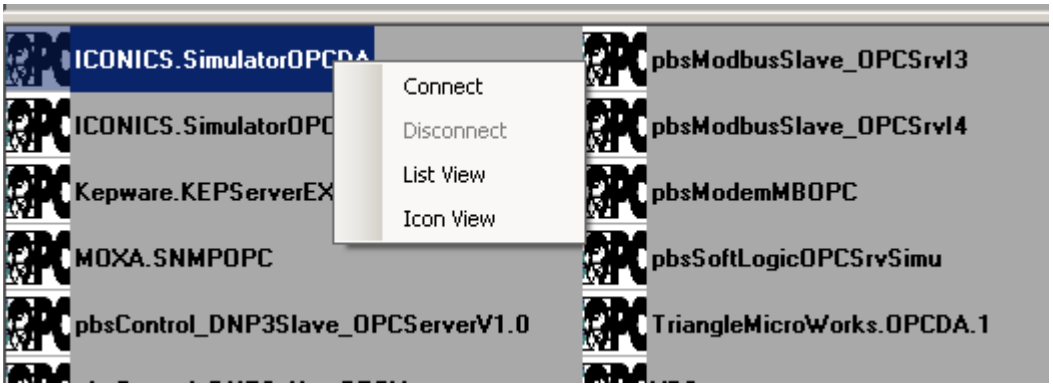


OPC explorer will look at \psle\OPC directory and find all files with OPCXMLn.xml name format and will make a new file with OPCXML{n+1}.xml name when n is max number in the OPC directory .

You can rename OPC configuration file by running Explorer menu and rename file by windows utilities.

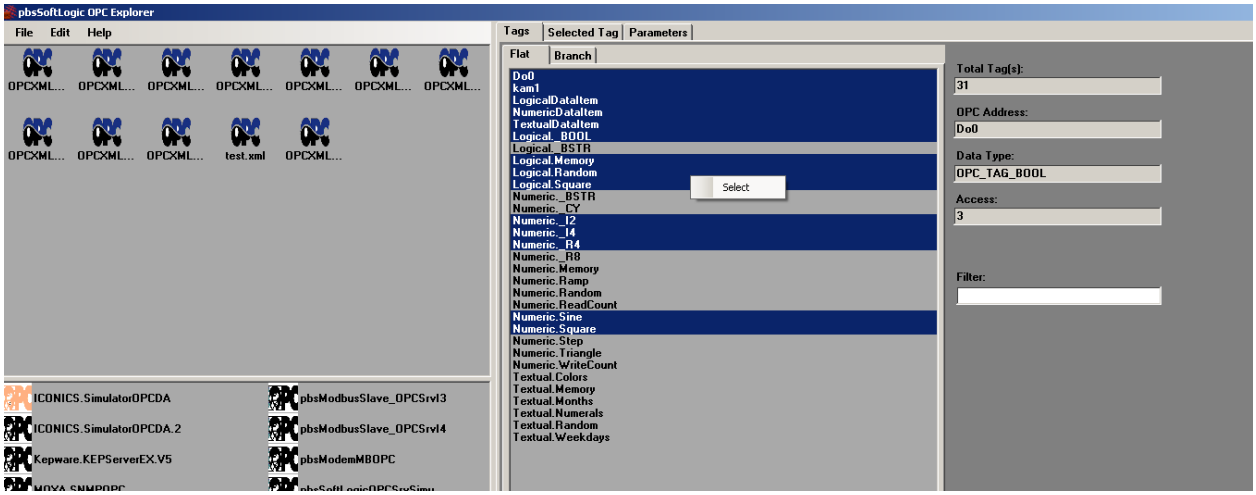
By running refresh Menu, Defined OPC configuration file panel will be refreshed with new names.

After you make a new OPC configuration file, select OPC server at Installed OPC server panel and connect to OPC server by right click menu.



From OPC server tags Panel, select all tags that you want to add in configuration. You can use Filter at right side to find OPC tags. You can press and hold Ctrl Key and select multiple items by left click.

Right click on selected Tags and run “Select” Menu.

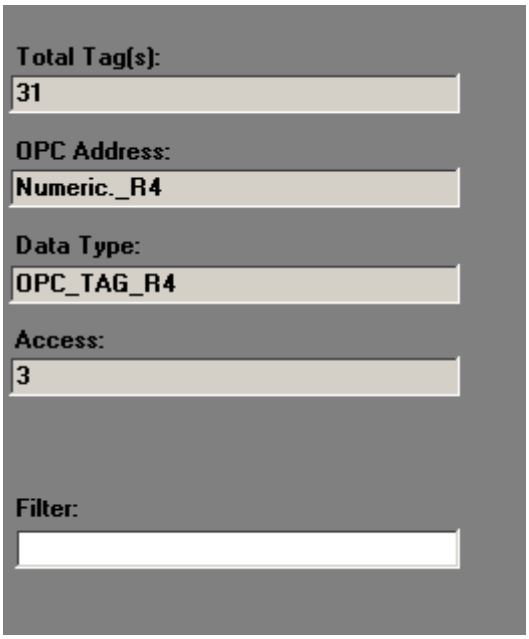


String and date data types are not supported in pbsSoftLogic for OPC client Driver.

Following Data Types are supported:

VT_I2	2 byte signed int
VT_I4	4 byte signed int
VT_R4	4 byte real
VT_R8	8 byte real
VT_BOOL	True=1, False=0
VT_I1	signed char
VT_UI1	unsigned char
VT_UI2	unsigned short
VT_UI4	unsigned long
VT_I8	signed 64-bit int
VT_UI8	unsigned 64-bit int
VT_INT	signed machine int
VT_UINT	unsigned machine int

At right Panel you can see OPC tag properties:



Tag Access :

OPC server tag has Read Access by client = 1

OPC server tag has write Access by client = 2

OPC server tag has Read/write Access by client = 3

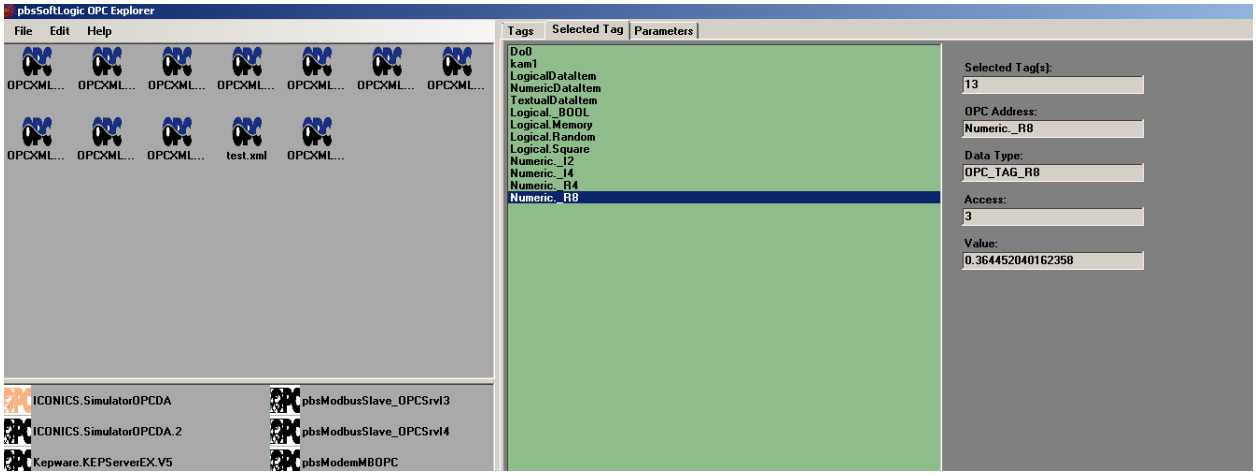
VT_BOOL has different definition in OPC:

```
VT_BOOL                    True=-1, False=0
```

But in pbsSoftLogic Runtime kernel, it is mapped as following:

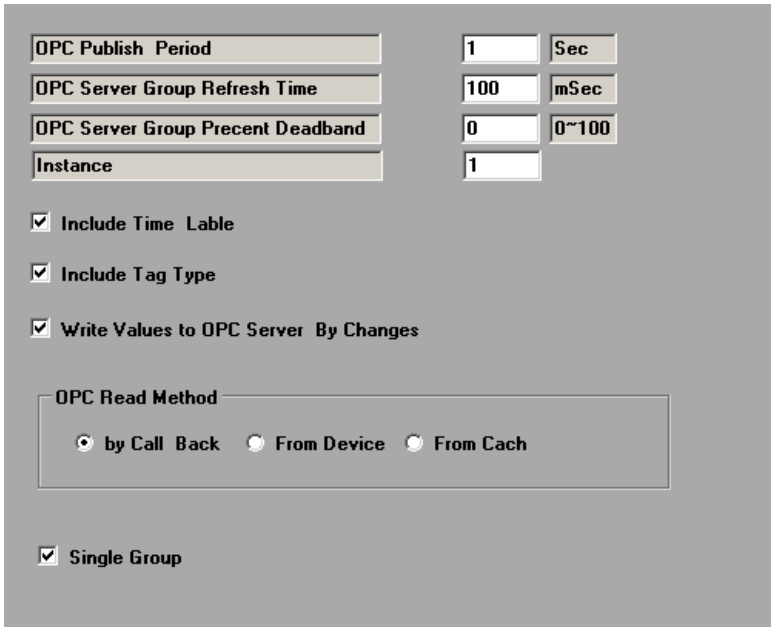
```
VT_BOOL                    True=1, False=0
```

After selecting OPC tags, click on “Selected Tags” Tab. you can see list of selected tags and at right side Tag properties with Tag Value.



When you click on each tag, Tag properties will be update at right panel.

Click on “Parameters” tab. you can see OPC connection parameter page.



OPC driver uses following parameters:

- OPC Server Group refresh Time
- OPC Server Group percent Dead band
- Instance

Other parameters are for PCWIN32 target and not used in Win32 Target.

OPC Server Group percent Dead band definition from OPC standard:

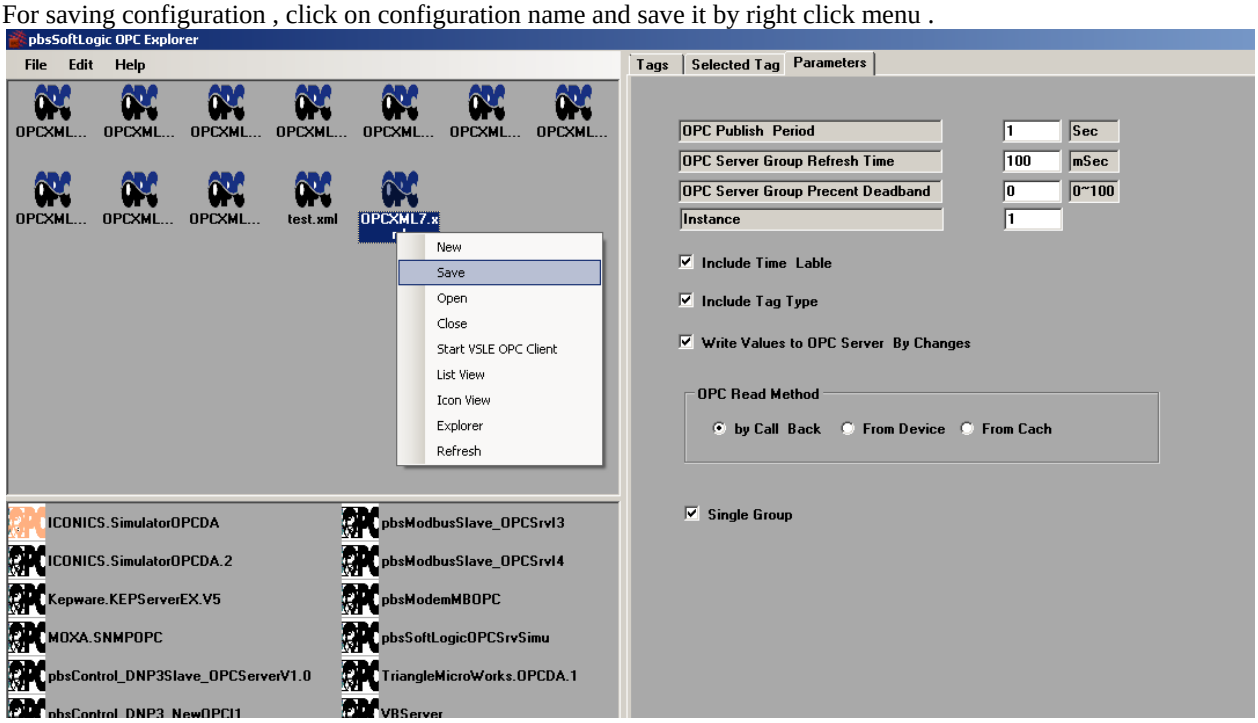
The percent change in an item value that will cause a subscription callback for that value to a client. This parameter only applies to items in the group that have Analog signals.

OPC Server Group refresh Time definition from OPC standard:

The fastest rate at which data changes may be sent to client for items in this group.

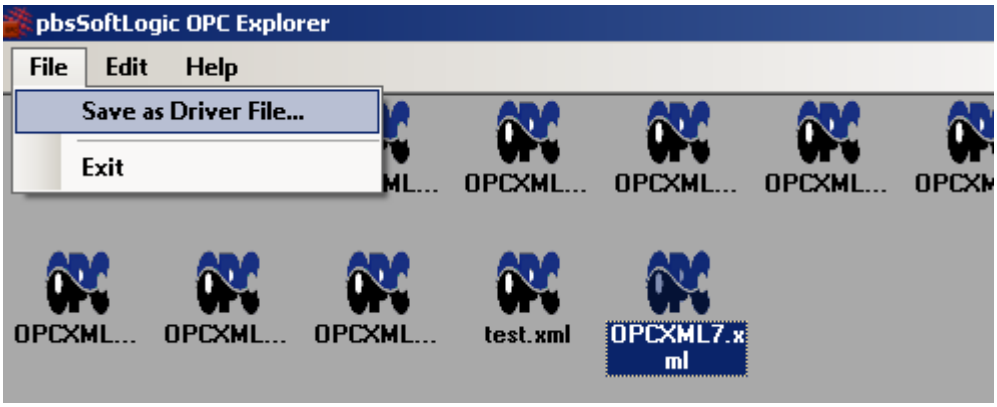
Instance:
Instance number is same as driver instance number.

For saving configuration , click on configuration name and save it by right click menu .



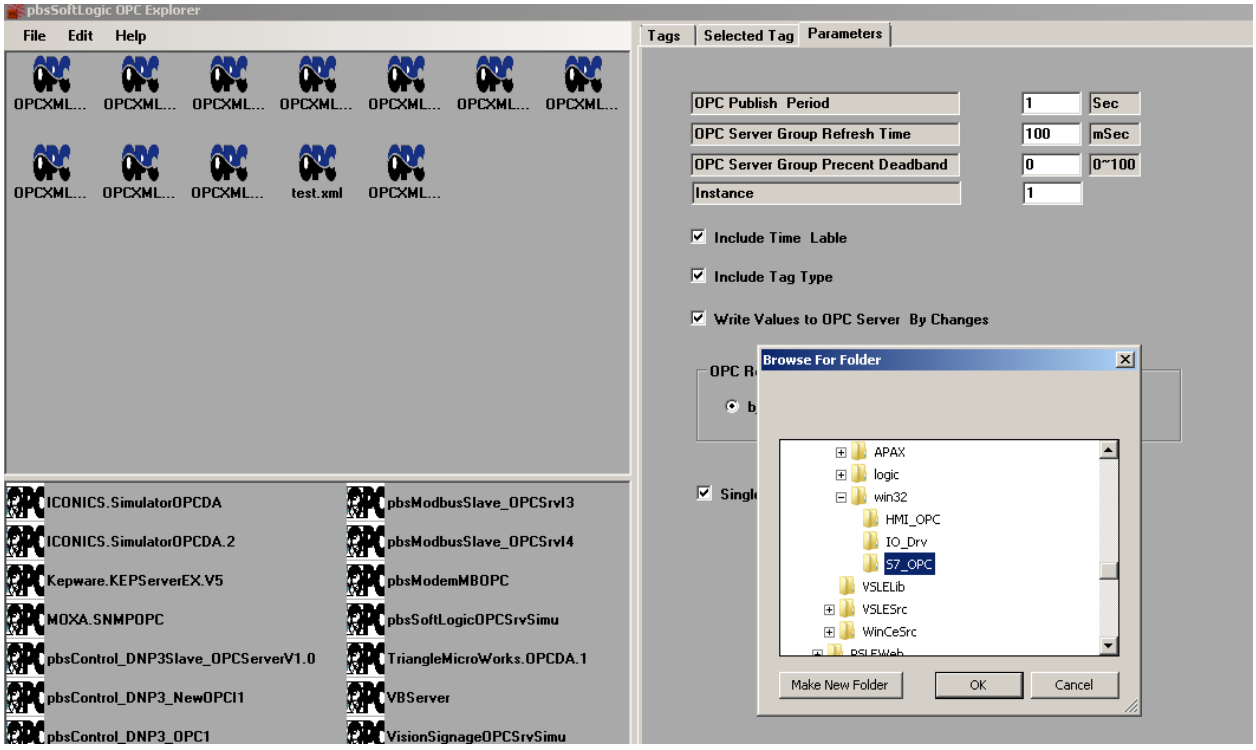
OPC configuration file will be saved at \psle\OPC directory. This file is same file that is used in OPC client driver configuration.

For saving OPC configuration as OPC driver configuration, at “file” menu, select “Save as Driver File”



Save and close configuration file then run “Save as Driver File…” menu and select Driver path.

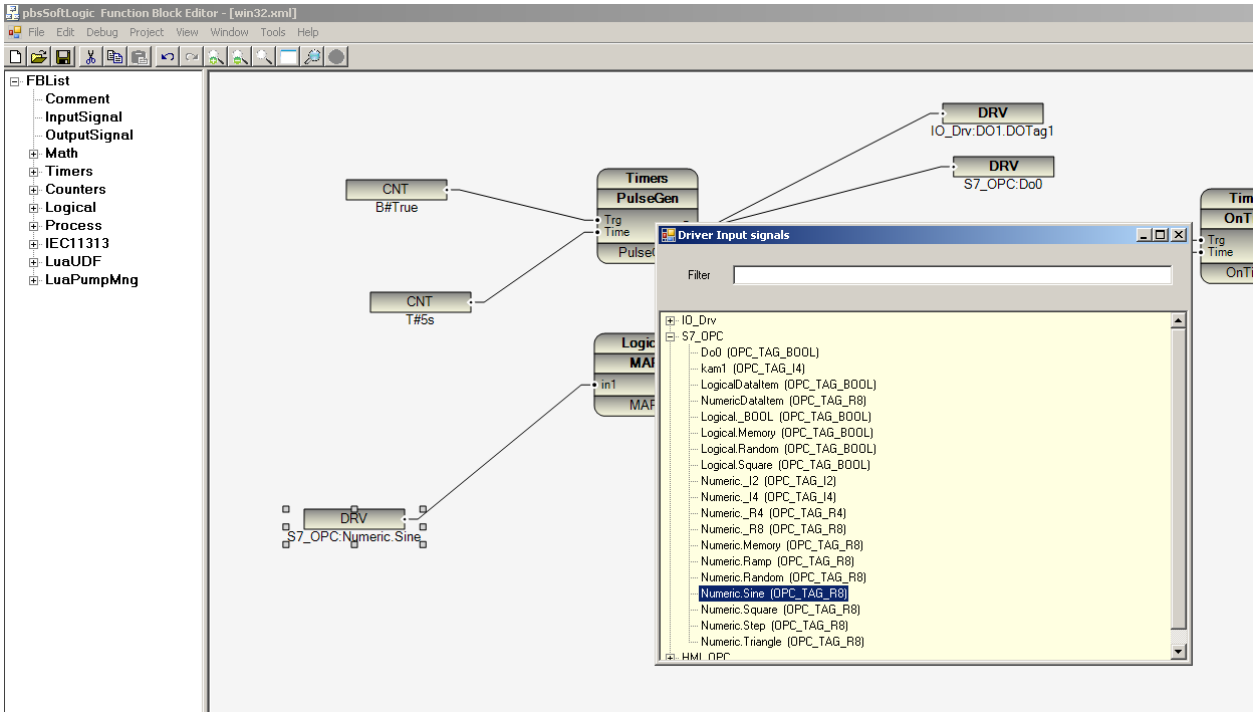
You should select same folder that is made by pbssoftlogic when you define OPC Client driver in project setting page.



It will copy OPC configuration file at driver folder with OPCTags.xml name.

Using OPC Tags in your logic :

- use InputSignal or Outputsignal Elements in your logic
- Right click on Inputsignal or Outputsignal elements
- Select “DRV Signals”
- Select OPC signal from Driver list signals.



- After finish logic , compile logic from “project/compile” menu.
- Transfer configuration file to Controller by “Project/Transfer Configuration” menu.
- Transfer Logic file to controller by “Project/Transfer Logic” menu.
- Restart runtime kernel. (psleWin32RT.exe)

When you transfer Logic and configuration to controller, pbssoftLogic will use following files and change their names as following: (suppose project name is win32)

Win32.lx : it is compiled configuration file will copy to controller and its name changed to logic.cfg

Win32.c11 : it is compiled logic file will copy to controller and its name changed to logic.c11

OPC client Driver runtime specifications:

- Remote Server name: 128 characters

- OPC server name: 256 characters

- OPC Item name: 128 characters

- OPC Server DA 2.0

- Selected OPC Items will read one time and Driver start time, after that OPC server should write Changes by call back to OPC client Driver.

- Maximum Number of OPC Tags for each instance: 1024

- Maximum Number of OPC instance: 8