

[1 – Introduction](#)[2 – pbsSoftLogic Installation](#)[3 – Basic concepts](#)[4 – Function Block programming Language](#)[5 – Quick Startup and logic Simulation](#)[6 – Runtime Kernel for Linux , transferring License to Controller and Working with Linux](#)[7 – Project Settings facilities](#)[8 – AMS-R3010 RTU Configuration](#)[9 – Modbus Master Configuration and integration with remote I/O Modules](#)[10- Modbus Slave Configuration](#)[11 – DNP3 Slave Configuration](#)[12 – IEC870-5 Slave \(101-104\) Configuration](#)[13 – S7 Communication Driver Configuration](#)

[14 – SQLite Configuration, RTU local data Archiving and Automatic synchronization with MS SQL Server](#)

[15 – OPC UA Server configuration for RTU](#)

[16 – Mobile Networking and GSP Client Driver Configuration](#)

[17 – IEC870-5-103 Master Driver Configuration](#)

[18 – OPC Client Driver Configuration for Win32 Target](#)

[19 - User defined function block by Lua Scripting and C Language](#)

[20 – pbsSDK :User defined Communication Driver Development](#)

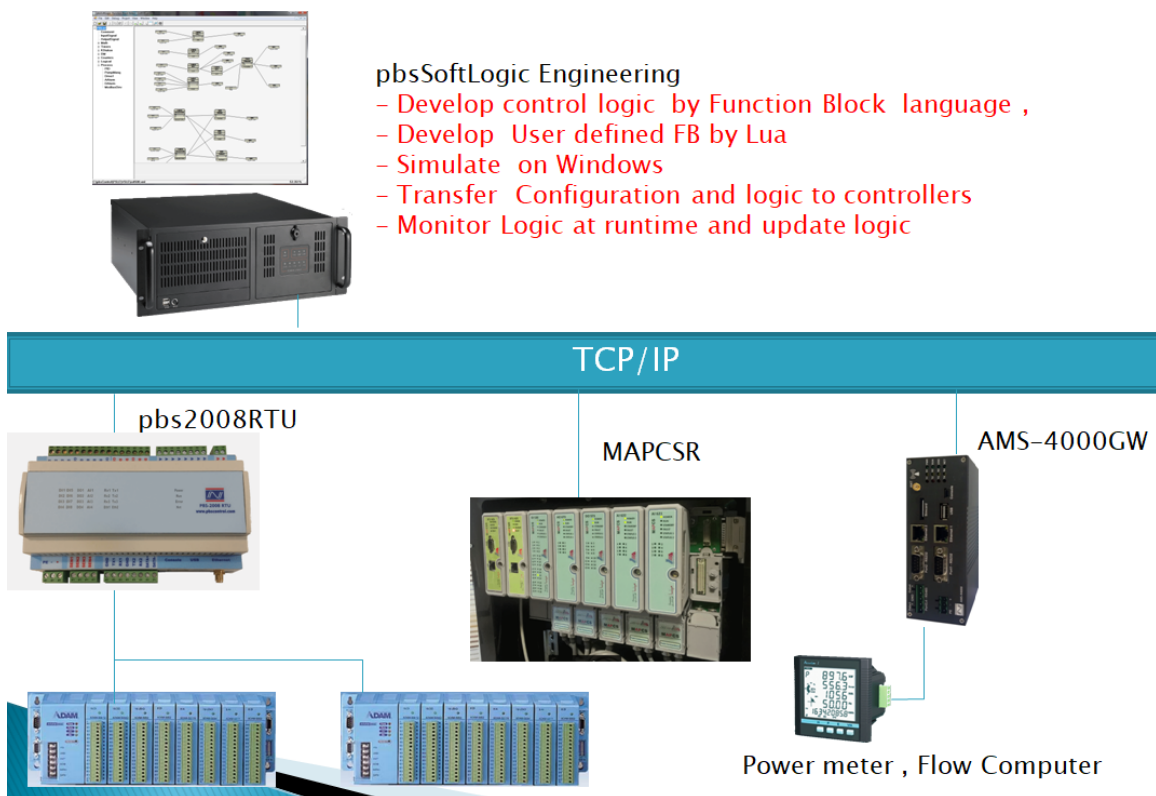
[21 – Standard Function Blocks Definition](#)

1 – Introduction

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

pbsSoftLogic has following specifications :

- Standard Function Block programming Environment
- Lua (scripting Language) is used for user defined Function Blocks development
- Developed application can be run on Embedded Windows , WinCE , QNX and Embedded linux OS
- Offline simulation of developed application on windows
- More than 300 Ready and tested Function block for easy programming.



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

Current Version: 3.0 RC2

Date: Sep 2019

2 – pbsSoftLogic installation

pbsSoftLogic Engineering is running on any Operating systems which support Dot Net Frame 4.7.1.

You need to install Dot Net Frame 4.7.1 on your machine for proper operation of pbsSoftLogic .

You can download latest pbsSoftLogic from <http://www.pbscontrol.com>

Simply unzip file and run VSFBEditor.exe. No need for any installation process.

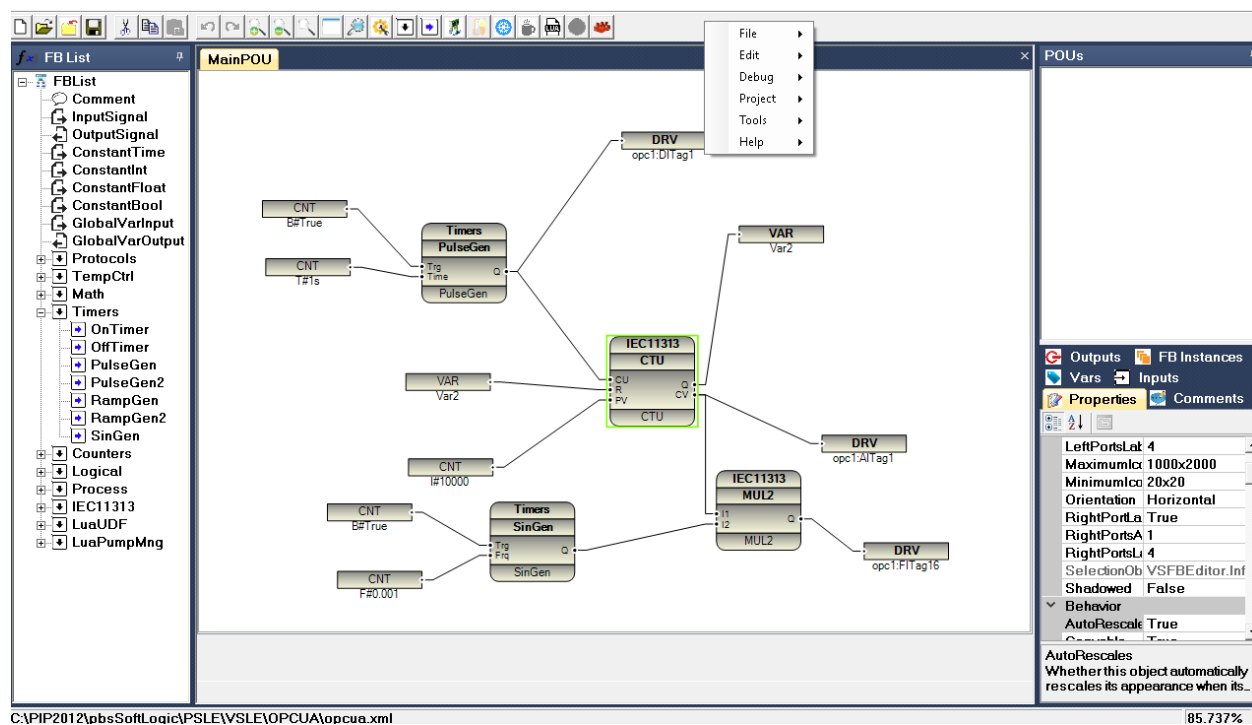
pbssoftLogic files and directories :

- VSFBEditor.exe Main Application for developing Projects .
- FBLuaEditor.exe User defined Function block editor With Lua Scripting language
- OPCExplorer.exe OPC Configuration file for connecting to OPC servers with Windows Target
- pbsLMP.dll Logic Monitoring Protocol . will use for Logic monitoring in Controller .
- options.xml basic options of pbsSoftlogic .
- cfg Directory : Body definition of Function blocks and Lua Source code .
- doc Directory : user manual of pbsSoftlogic
- LinuxCSrc 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 : Compiled Lua Script will move in this directory for transferring to controller
- PSLERT Win32 Runtime kernel
- SQLite : SQLite database for local RTU data archiving
- Win32Simulation : Logic simulation kernel
- DNPS.exe , IECSlave.exe , MDL1000.exe , MDL2000.exe , ModbusSlave.exe , SQLiteEditor.exe ,MapCsR1cfg.exe : protocol and Modular RTU Editor . You can use these editors or directly edit xml configuration file for each Modular RTU or protocol.
- Lua52.exe , Luac52.exe Lua Compiler and utility

Other files are system files and should be at PSLE directory .

cfg	IECSlave.exe	QWhale.Syntax.Schemes.dll	QWhale.Common.xml
de	lua52.exe	WtOPCSvr.dll	QWhale.Editor.xml
doc	luac52.exe	DNPS.exe.config	QWhale.Syntax.Parsers.xml
en	MapCsR1Cfg.exe	FBCSEditor.exe.config	QWhale.Syntax.Schemes.xml
es	MDL1000.exe	FBCSEditor.vshost.exe.config	QWhale.Syntax.xml
fr	ModbusSlave.exe	FBLuaEditor.exe.config	
License	OPCEXplorer.exe	IECSlave.exe.config	
linux	SQLiteEditor.exe	MapCsR1Cfg.exe.config	
LinuxCSrc	VSFBEditor.exe	MapCsR1Cfg.vshost.exe.config	
OPC	VSFBEditor.vshost.exe	MDL1000.exe.config	
openvpn	VSOPCSimu.exe	ModbusSlave.exe.config	
PSLERT	Janus.Data.v4.dll	OPCEXplorer.exe.config	
qnx	Janus.Windows.CalendarCombo.v4.dll	pbsLogicSimulator.exe.config	
ru	Janus.Windows.Common.v4.dll	SQLiteEditor.exe.config	
Schemes	Janus.Windows.FilterEditor.v4.dll	VSFBEditor.exe.config	
SDK	Janus.Windows.GridEX.v4.dll	VSFBEditor.vshost.exe.config	
Sqlite	Janus.Windows.UI.v4.dll	VSLE.exe.config	
target	lua51.dll	VSOPCCClient.exe.config	
Temp	LuaInterface.dll	VSOPCSimu.exe.config	
Timezone	Northwoods.Go.dll	VSStartup.exe.config	
uk	Northwoods.Go.Draw.dll	GlobalSettings.xml	
Utility	Northwoods.Go.Svg.dll	Janus.Windows.CalendarCombo.v4.xml	
VSLE	Northwoods.Go.Xml.dll	Janus.Windows.Common.v4.xml	
VSLELib	pbsLMP.dll	Janus.Windows.FilterEditor.v4.xml	
VSLESrc	pbsOPCCAPI.dll	Janus.Windows.GridEX.v4.xml	
Win32simulation	PBSOPCCClient.dll	Janus.Windows.UI.v4.xml	
Win32Src	pbsOPCSrvAPI.dll	Northwoods.Go.Draw.xml	
WinCeSrc	QWhale.Common.dll	Northwoods.Go.Svg.xml	
DNPS.exe	QWhale.Editor.dll	Northwoods.Go.xml	
FBCSEditor.vshost.exe	QWhale.Syntax.dll	Northwoods.Go.Xml.xml	
FBLuaEditor.exe	QWhale.Syntax.Parsers.dll	options.xml	

In following figure you can see pbsSoftLogic Engineering Environment:



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 20 years minimum.

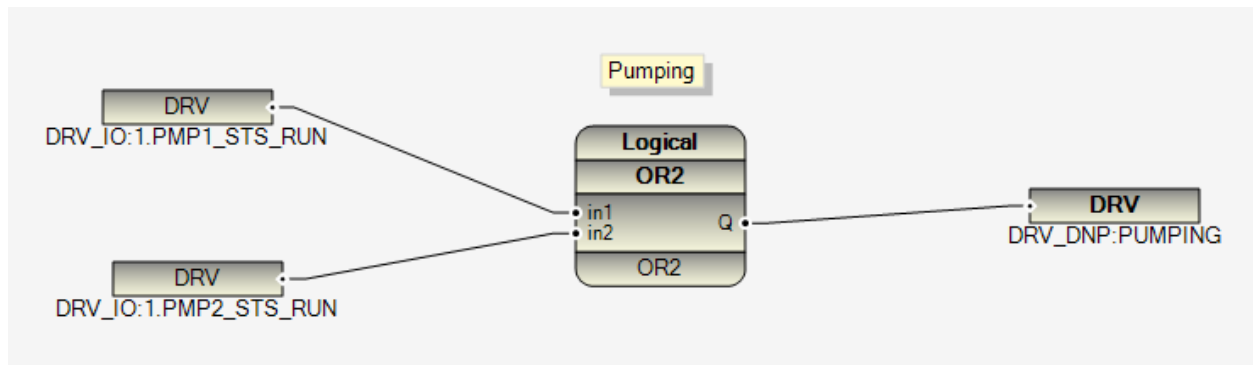
pbssoftLogic is an IDE for developing Function Blocks programs , Simulate , and downloading to Linux /QNX/Win32 based controllers . You can use Lua Scripting language for developing new FB by yourself.

All FB source code of pbsSoftLogic are open source and are located at \psle\LinuxCSrc and \psle\QNXCSrc and \psle\Win32CSrc

4 – Function Block Programming Language

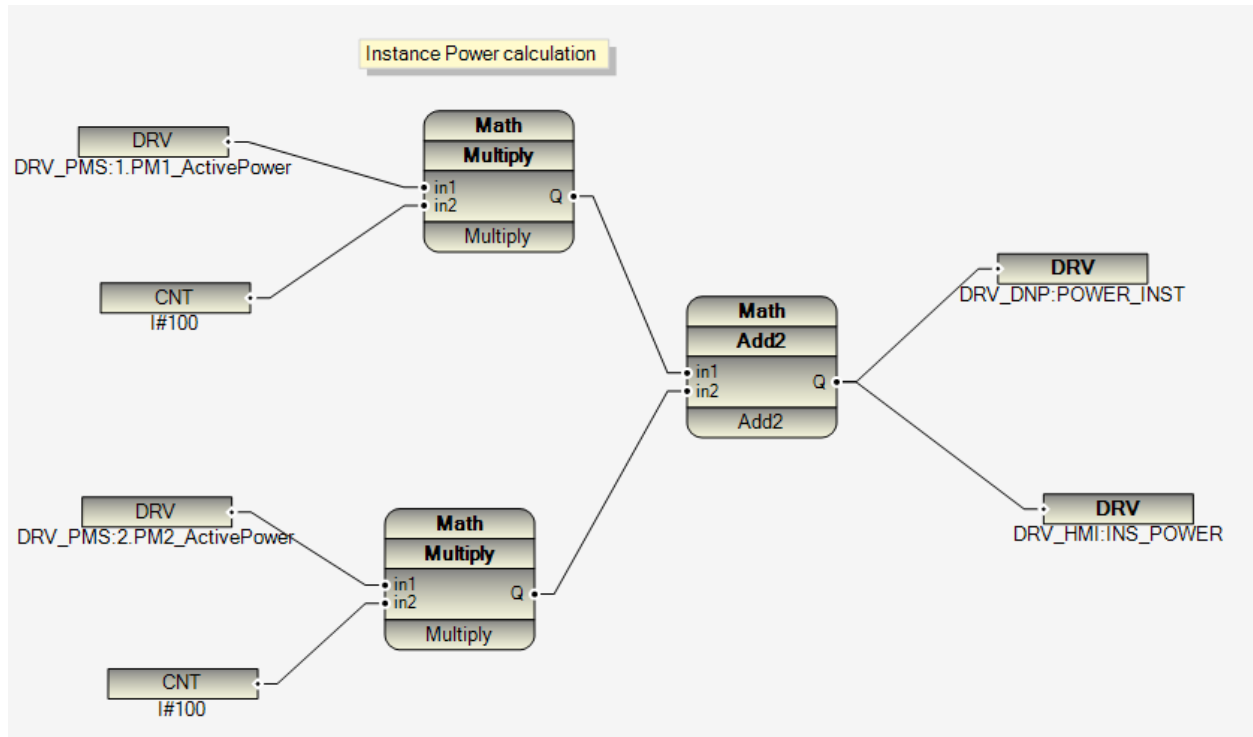
Main element of a Function Block program is FB (Function Block). In Following you can see a few simple examples.

Example1:



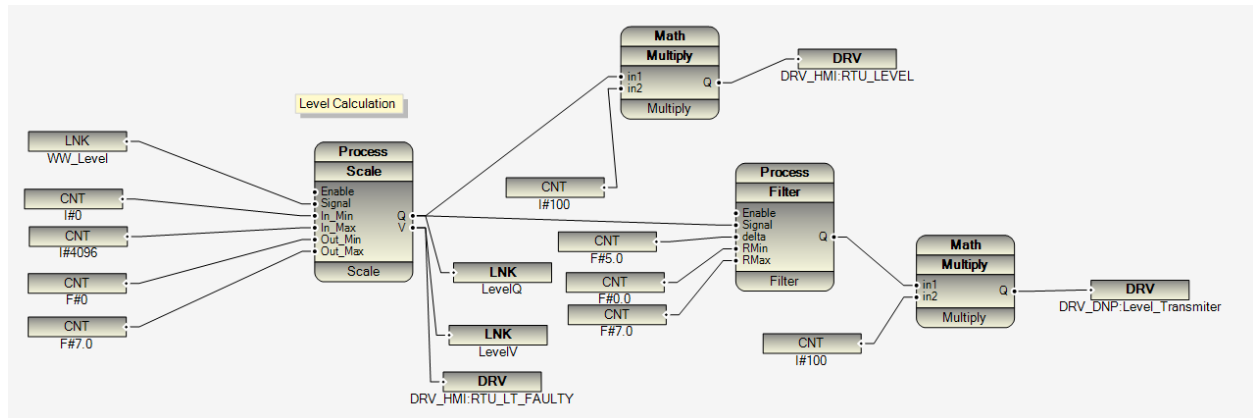
In this logic, two signals PMP1_STS_RUN and PMP2_STS_RUN are input to OR FB and Output will write to PUMPING Signal.

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)

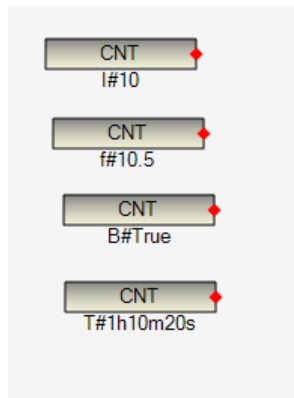
Example 3:



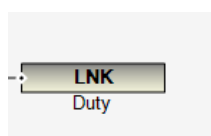
Main Element of a Function Block Program:

- 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)

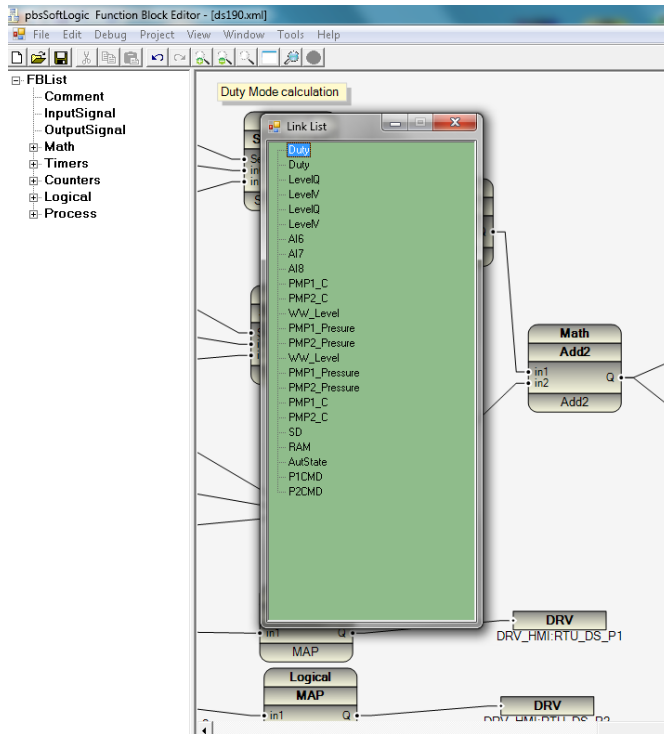
Constant Signal Format: Type # Value.



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. LNK and VAR have same usage.



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

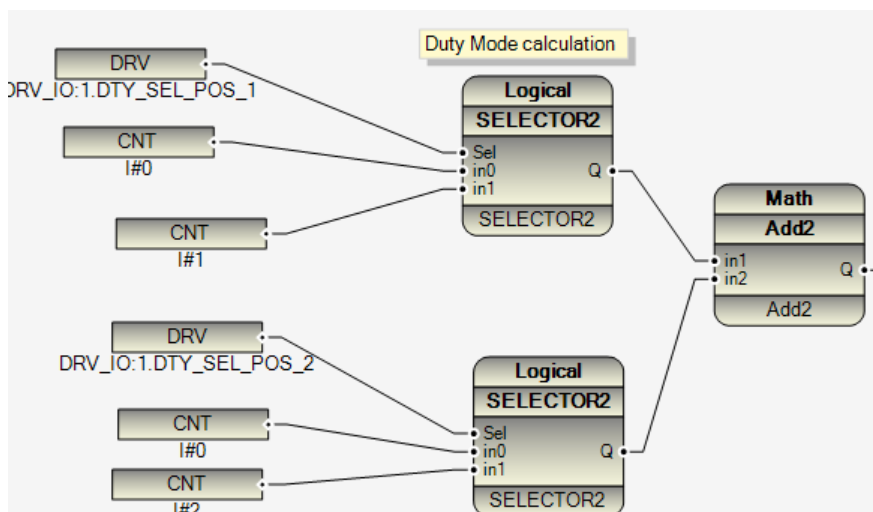


By double click on each link item; Logic will focus on Link/VAR Signal. So you can easily browse and check all link/VAR signals.

LNK/VAR is internal global variable in your project.

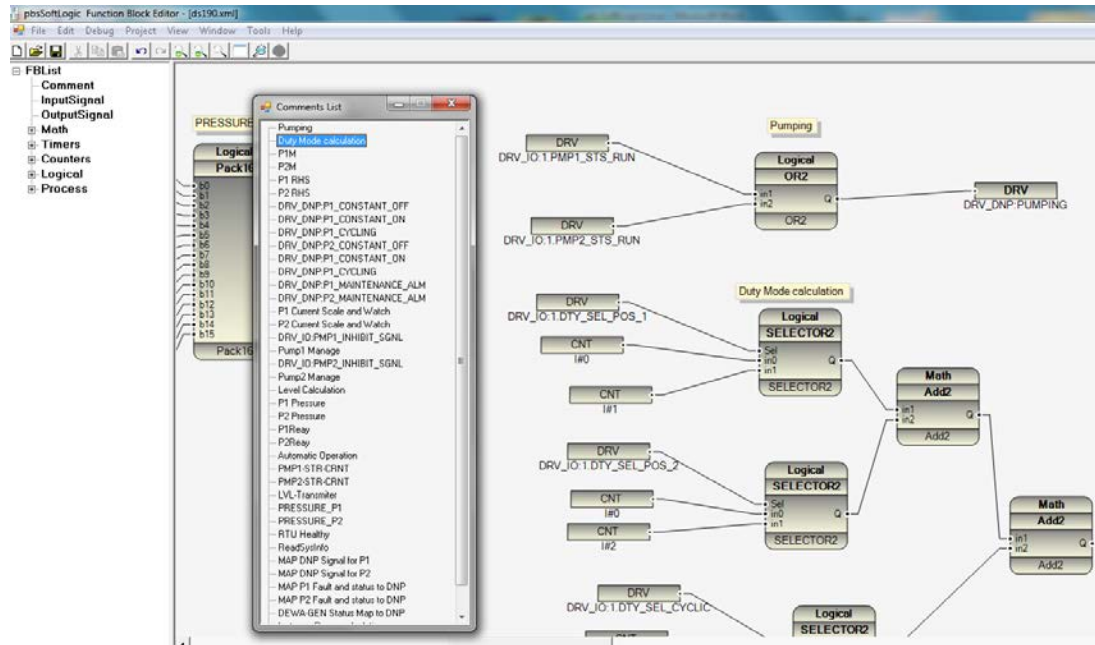
6 – Comments: you can put comment everywhere in logic. Drag a Comment element from FBList and

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



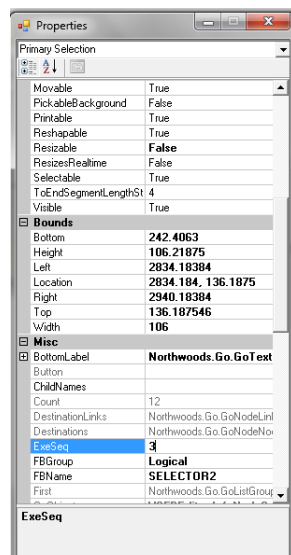
By selecting comments items from Debug menu, you can see list of all Comments in the logic.

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



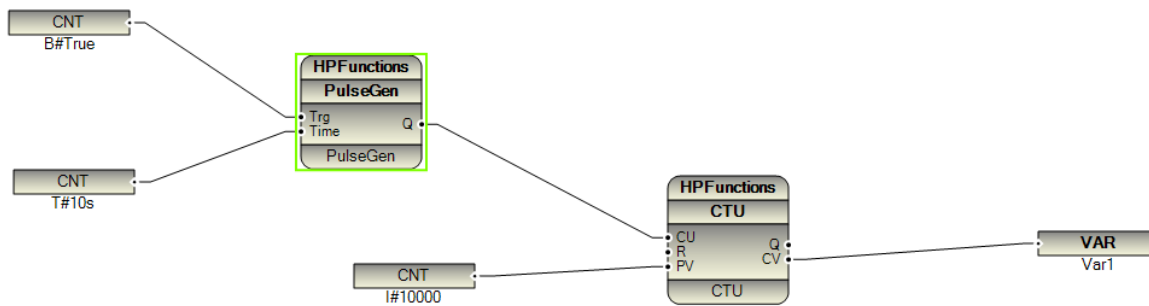
Function Block Programming Rules:

- 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 .

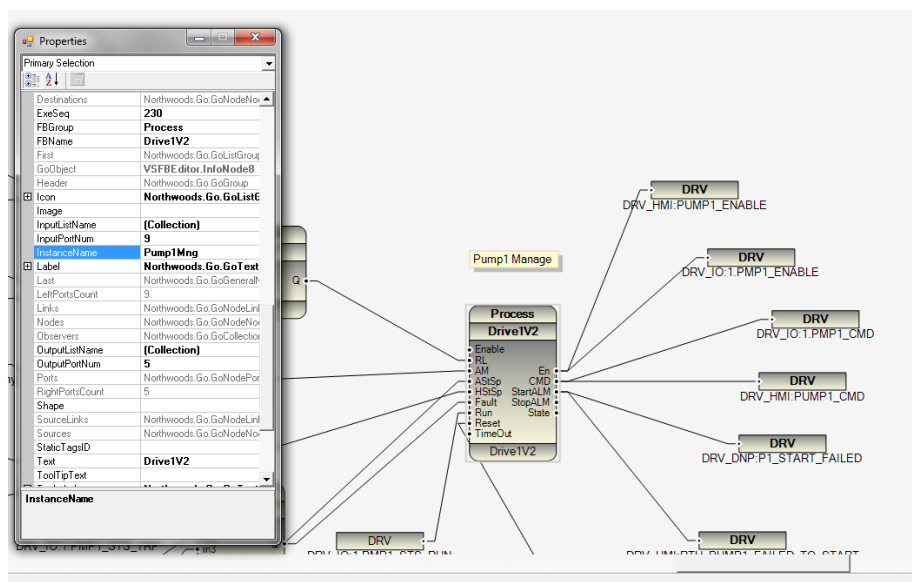


When you start to develop logic , FBEditor will increase ExeSeq number for each FB that you use automatically . but you can change its sequence and by this way , you can control execution sequence of logic . We advise to set all ExeSeq numbers manually , because when you copy paste some part of logic , 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 .

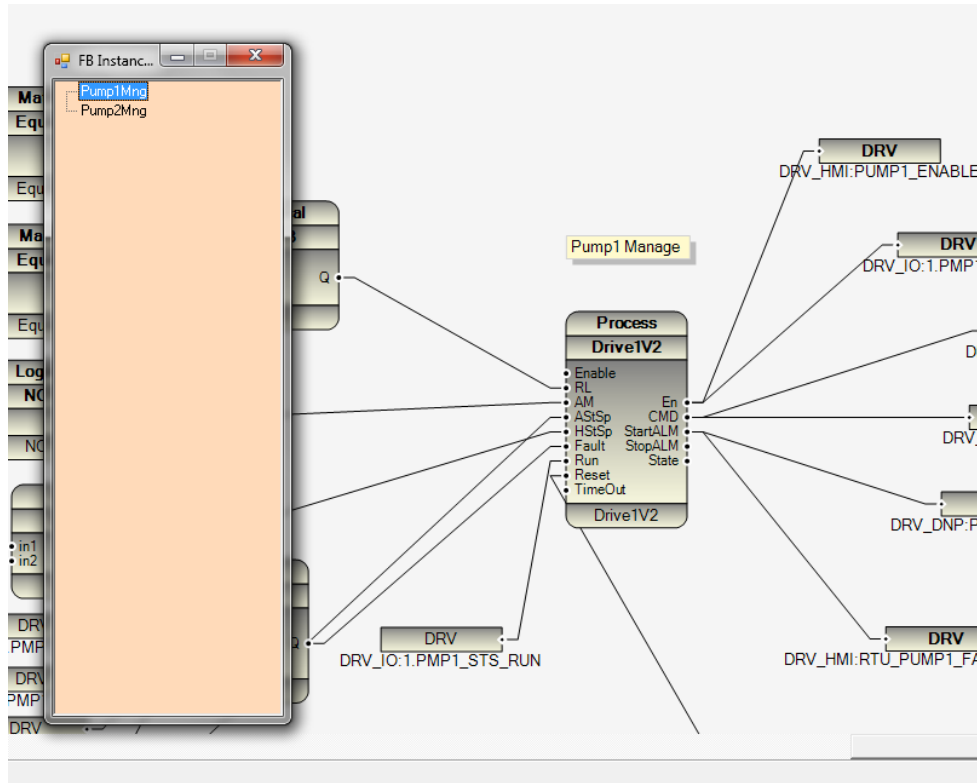
In following sample ExeSeq Number of PulseGen is Bigger than CTU , but Logic will solve without any problem and only in one RTU Cycle there is no output for CTU FB .



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.



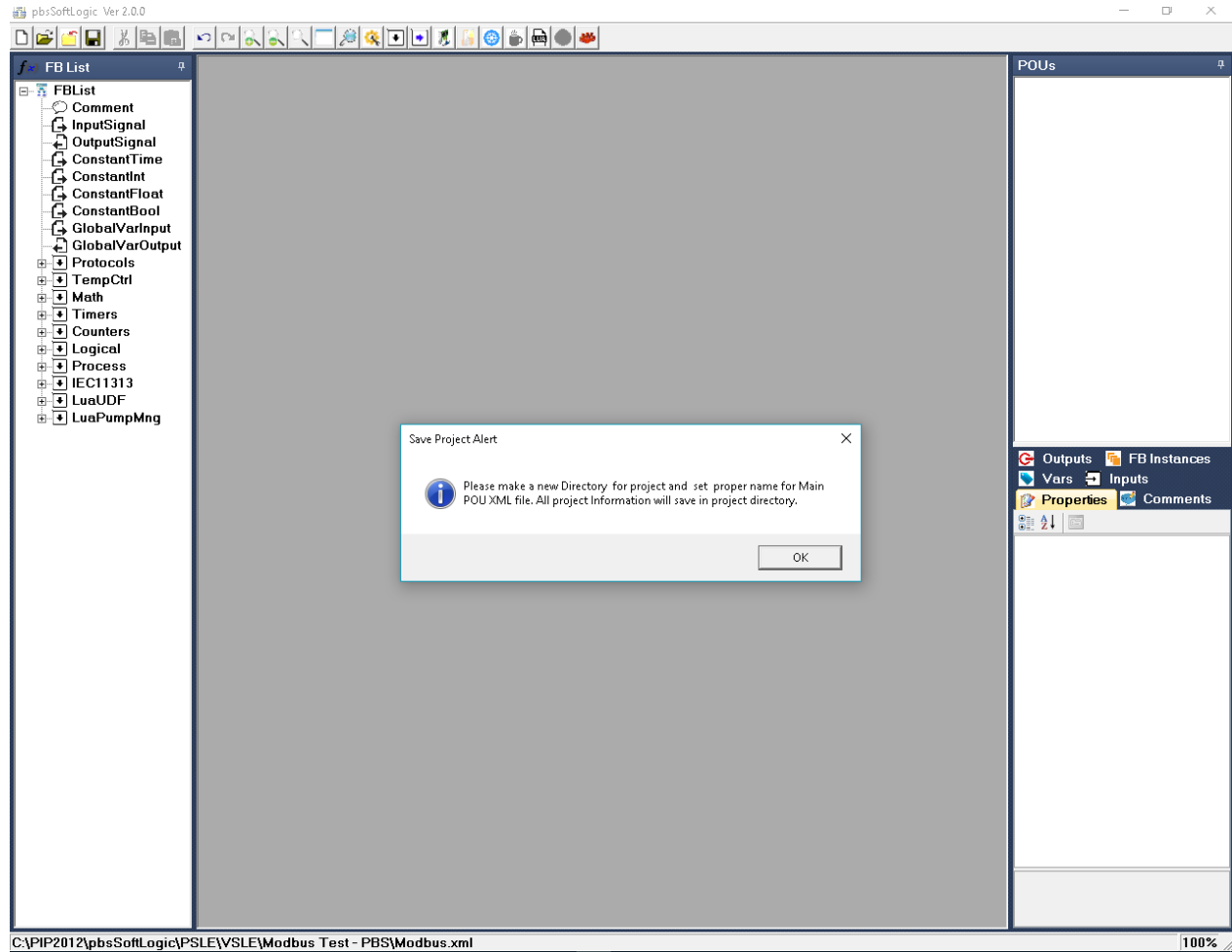
Note : if you want to have 100% warm logic update you should use instance name for critical function blocks .

6 – you can write your logic in multiple Function Block pages . Always First Function Block which is made by PSLE is main POU (Program Organization Block)

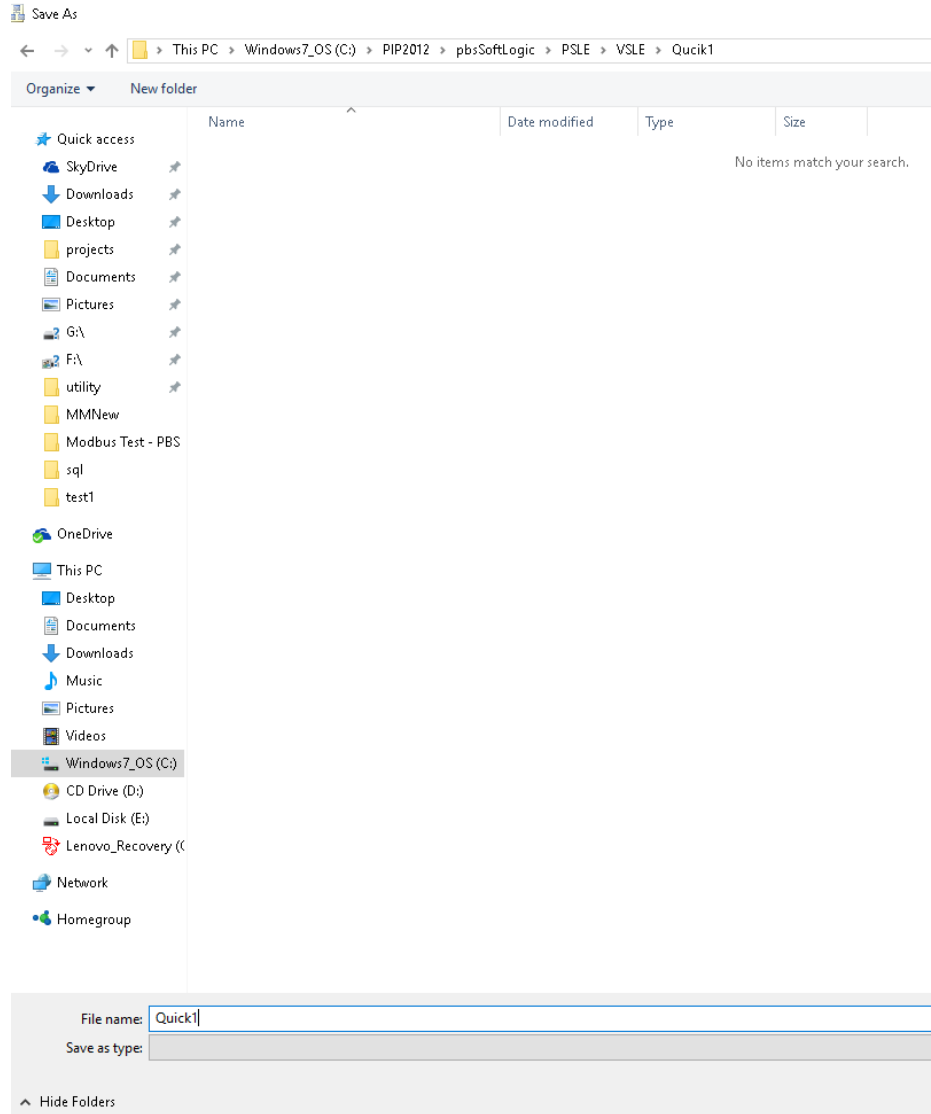
5 – Quick Startup and Logic Simulation

In this part, we 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.

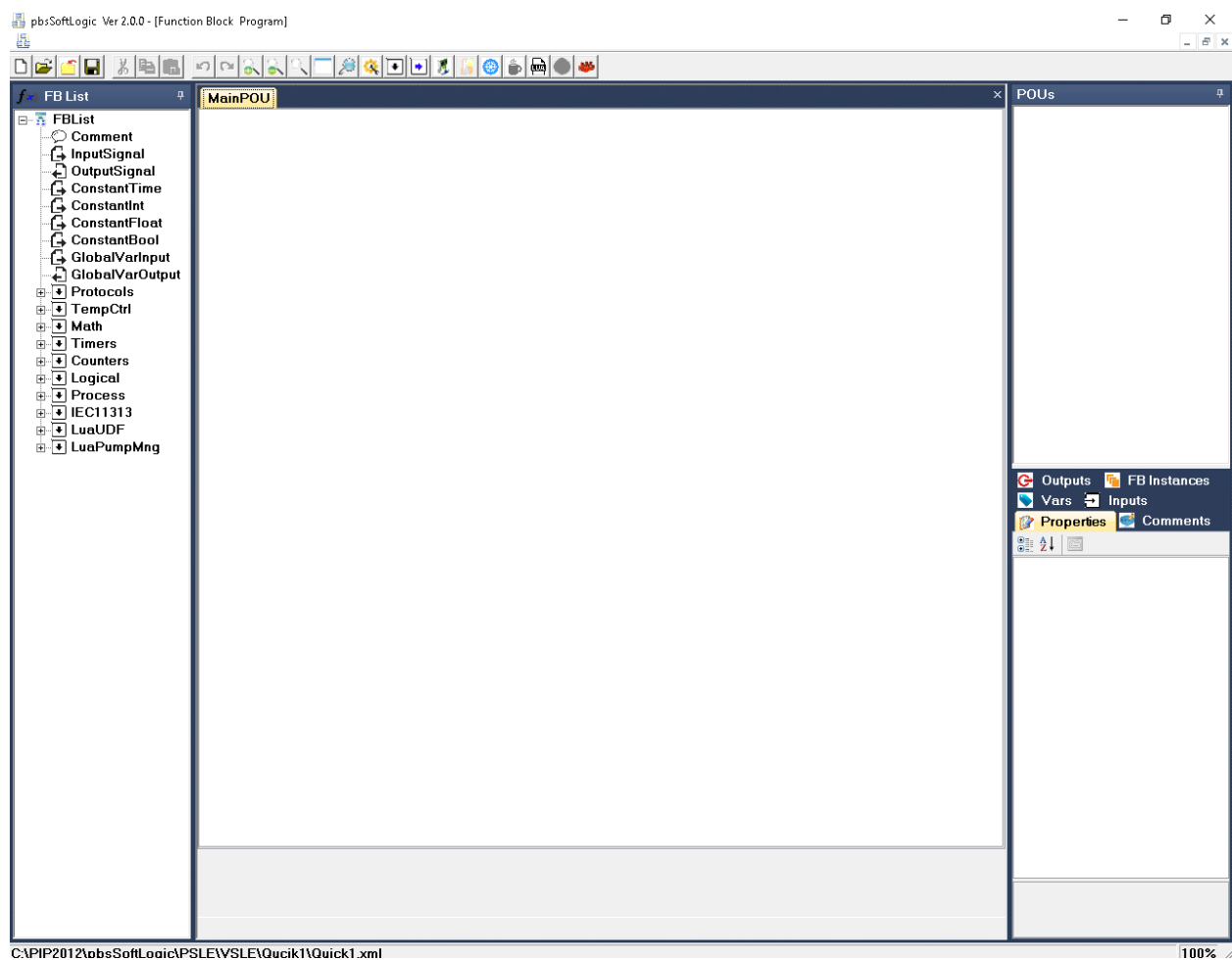


At first step you should make a directory for your project. You can make anywhere in your system with proper name related to your project. Suppose we will make Quick1 Directory in pbsSoftLogic VSLE Path.



Select a proper name for your MainPOU , I will name MainPOU as Quick1 too .

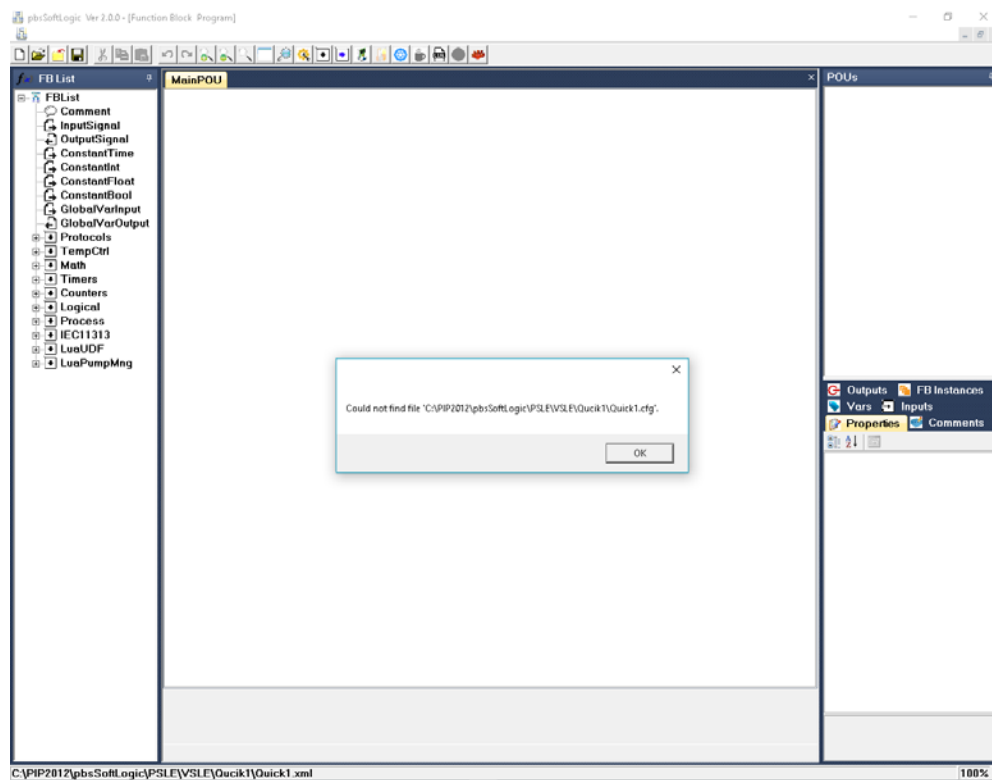
Click on save button, it will close save form automatically and you are ready for configuration and programming. You couldn't close MainPOU Program and it is always open when your project is open.



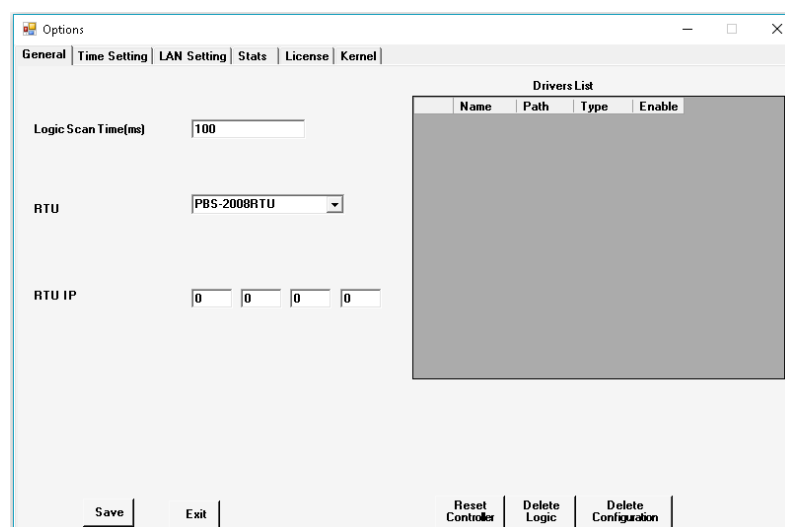
Step2 : Select your RTU Type . Click on project setting button



pbsSoftLogic will prompt you that there is no any configuration file for this project . Simply Click on OK Button to open project setting page.



When Project setting page is opened, it will show by default AMS-R3010 RTU as controller.



Open RTU Combo Box you can see different RTU that is supported by pbsSoftLogic . For now use same AMS-R3010 RTU.

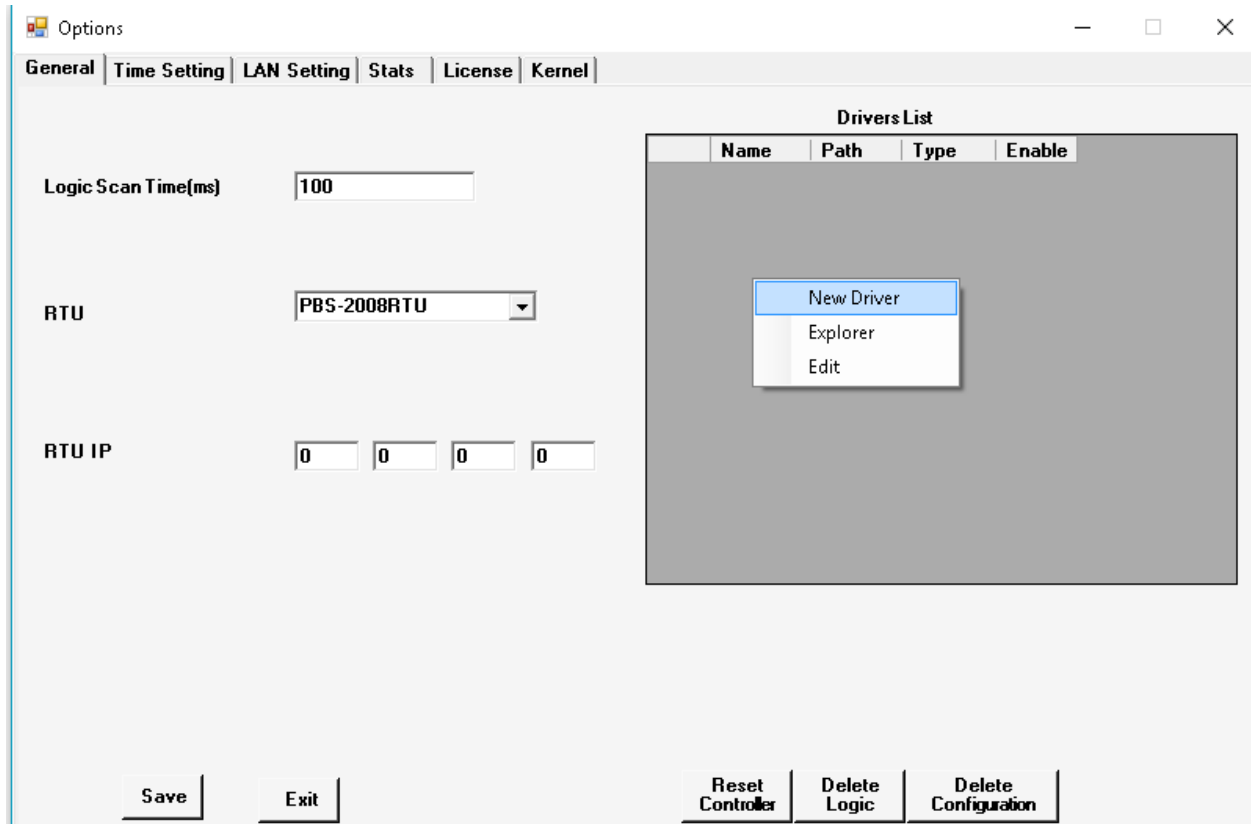
Type RTU IP for example type 192.168.1.213

Keep Logic Scan Time as 100msec. This is loop time for reading All Driver Inputs, Solve Logic and write Driver Outputs. Every 100 msec above sequence will, but may be whole this sequence only get 2 msec to finish and 98 msec CPU is sleeping.

Click on save button to save configuration.

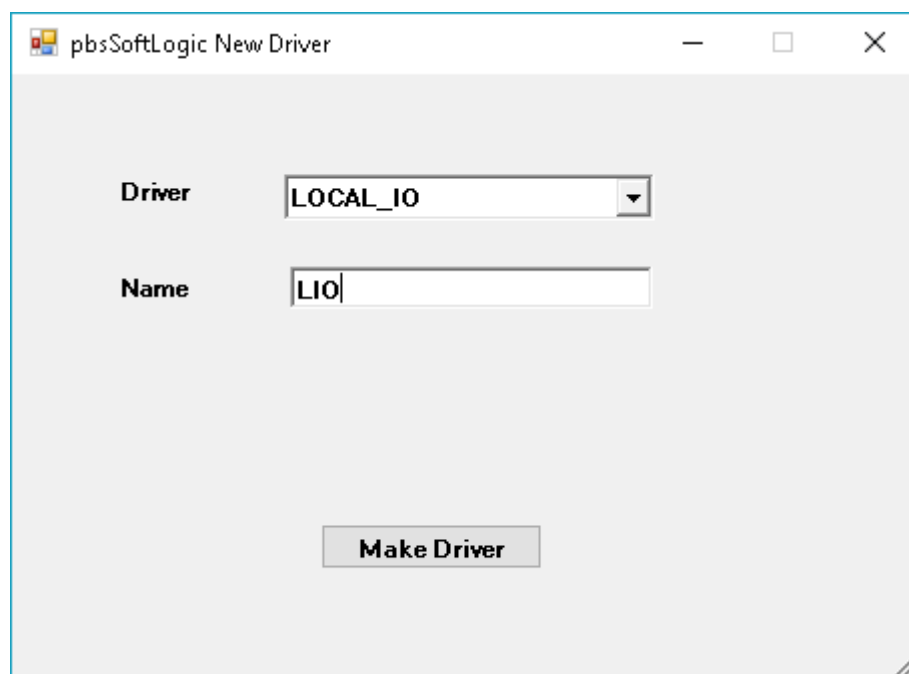
Step3: Define Local IO for RTU

Right click In Driver List part and select New Driver Command:

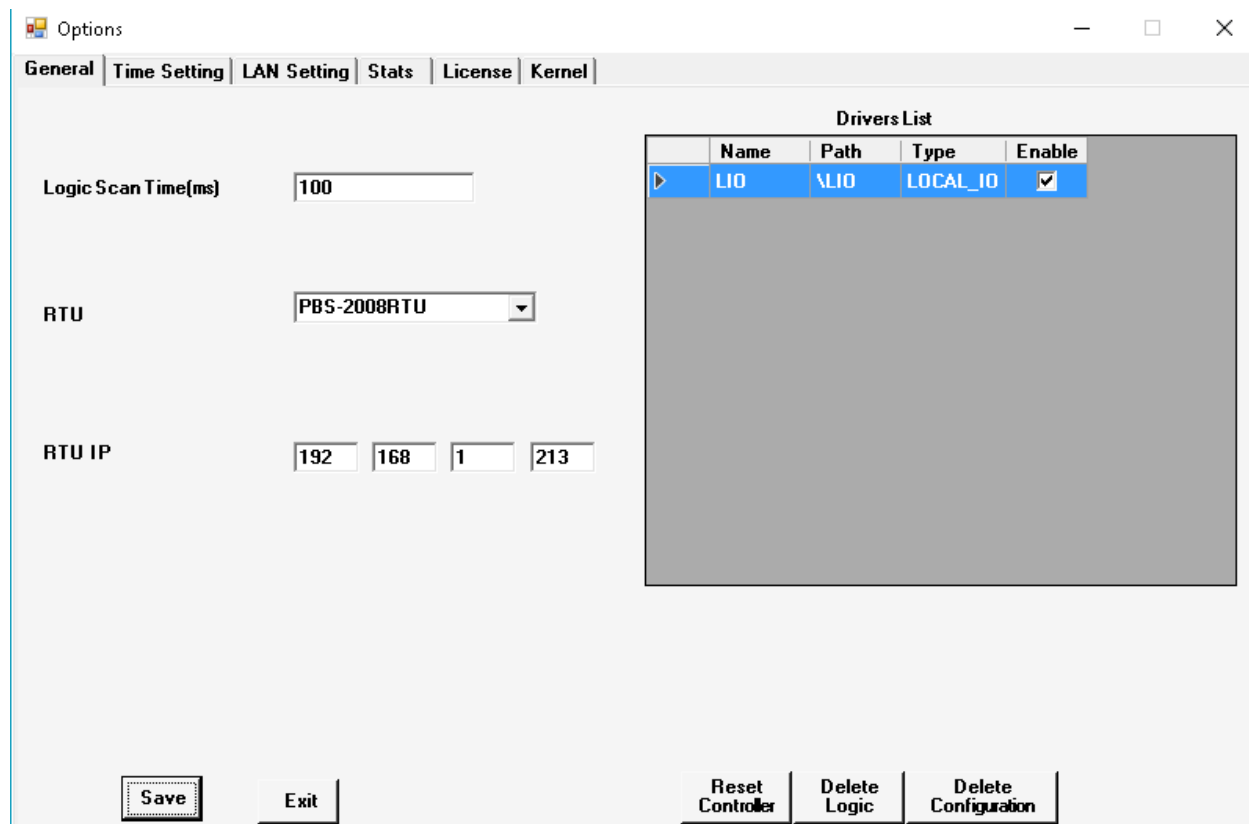


It will show list of Supported Drivers of pbsSoftLogic :

Select Local _IO and select an unique name for driver . (for example LIO)

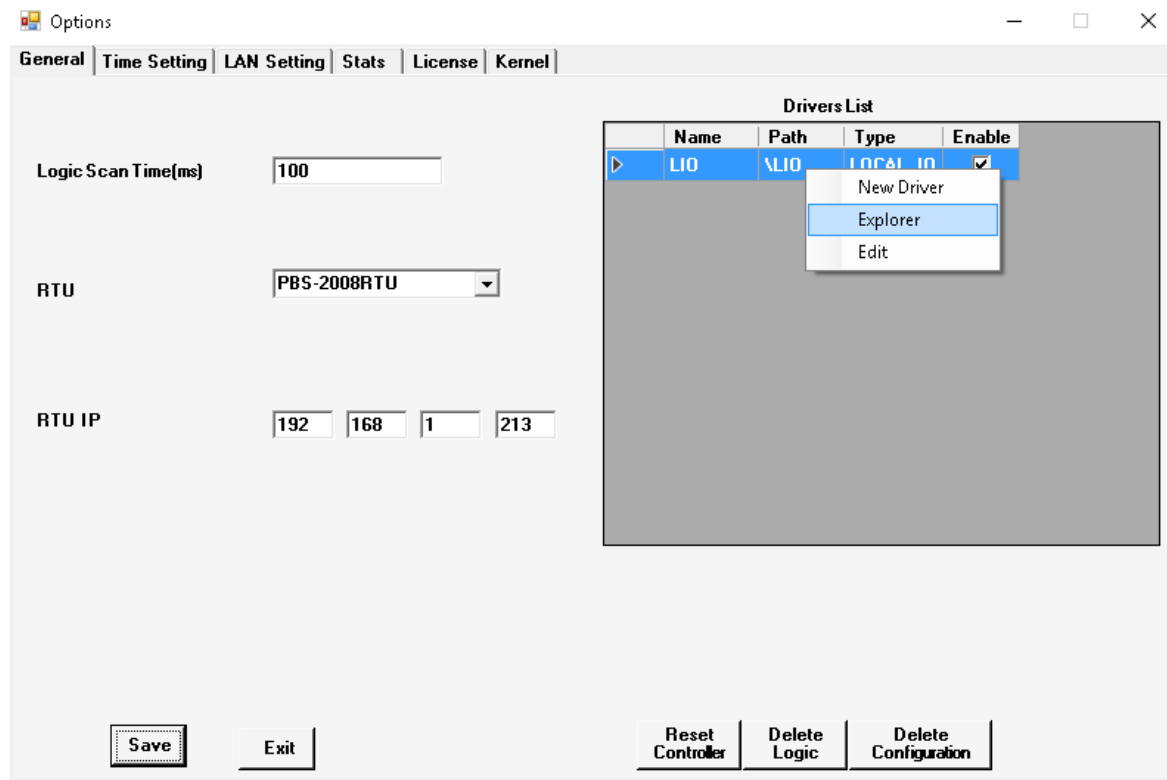


Click on Make Driver Button. pbsSoftLogic will include AMS-R3010RTU configuration file to your project.

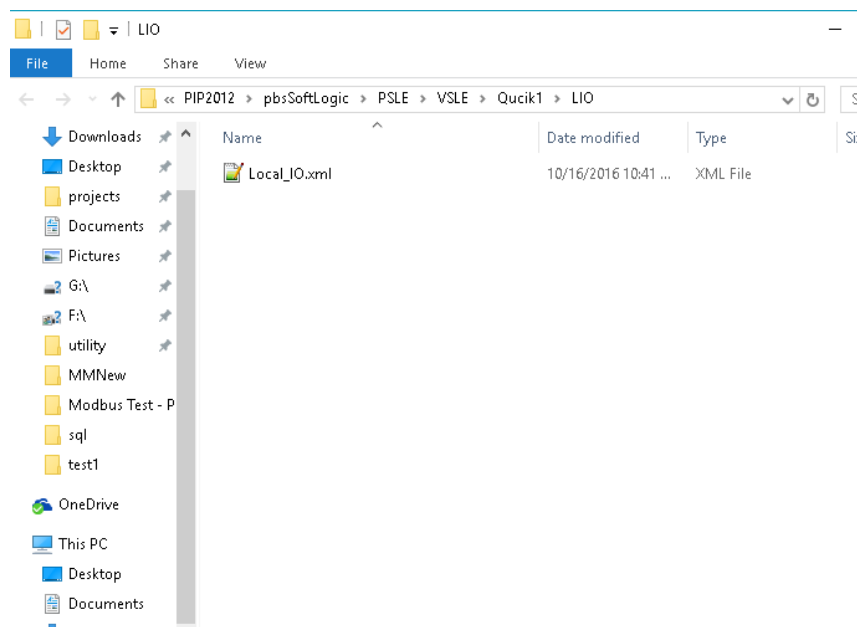


LOCAL_IO is a general driver for modeling hardware functionality of RTU. pbsSoftLogic will make different Local_IO configuration file based on RTU type .

Right click on LIO Driver and Select Explorer option.

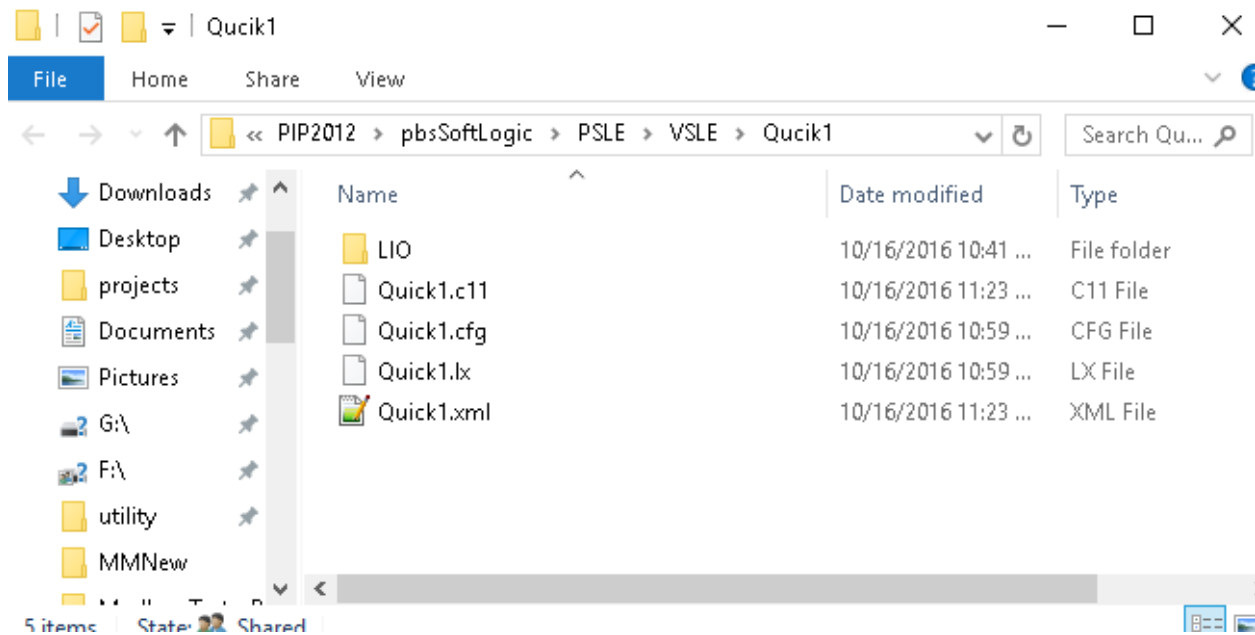


It will open LIO directory in your project:



pbsSoftLogic will make one directory for each Driver in project Directory .

If you open project directory you will see following file configuration :



Quick1.xml: source of your logic

Quick1.cfg: Project Configuration file.

Quick1.lx: Compiled Project Configuration file that is transferred to RTU

Quick1.c11: Compiled Logic that is transferred to RTU

LIO Directory: Directory for Local_IO Driver. Insider LIO directory pbsSoftLogic is make Local_IO.xml file.
For AMS-R3010 RTU Local_IO.xml file is as following:

```

<Tag Name="SYS.Reset" Type="SYS" Init="0" Address="0" />
<Tag Name="SYS.3GModemON" Type="SYS" Init="0" Address="1" />
<Tag Name="SYS.3GModemSignallevel" Type="SYS" Init="0" Address="2" />
<Tag Name="SYS.Temp1" Type="SYS" Init="0" Address="3" />
<Tag Name="SYS.Temp2" Type="SYS" Init="0" Address="4" />
<Tag Name="SYS.CNTTimer" Type="SYS" Init="200" Address="5" />
<Tag Name="SYS.AORange" Type="SYS" Init="1" Address="6" />
<Tag Name="SYS.Buzzer" Type="SYS" Init="0" Address="7" />
<Tag Name="SYS.IOScan" Type="SYS" Init="100" Address="8" />
<Tag Name="SYS.Total1" Type="SYS" Init="0" Address="9" />
<Tag Name="SYS.Total2" Type="SYS" Init="0" Address="10" />
<Tag Name="SYS.Total3" Type="SYS" Init="0" Address="11" />
<Tag Name="SYS.Total4" Type="SYS" Init="0" Address="12" />
<Tag Name="SYS.Total1RST" Type="SYS" Init="0" Address="13" />
<Tag Name="SYS.Total2RST" Type="SYS" Init="0" Address="14" />
<Tag Name="SYS.Total3RST" Type="SYS" Init="0" Address="15" />
<Tag Name="SYS.Total4RST" Type="SYS" Init="0" Address="16" />
<Tag Name="SYS.ChatterFilterCount" Type="SYS" Init="0" Address="17" />
<Tag Name="SYS.ChatterFilterBaseTimeMs" Type="SYS" Init="0" Address="18" />
<Tag Name="SYS.ChatterFilterFreezeTimeMs" Type="SYS" Init="0" Address="19" />
<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="BITag0" Type="DI" Init="0" Address="0" />
<Tag Name="BITag1" Type="DI" Init="0" Address="1" />
<Tag Name="BITag2" Type="DI" Init="0" Address="2" />
<Tag Name="BITag3" Type="DI" Init="0" Address="3" />
<Tag Name="BITag4" Type="DI" Init="0" Address="4" />
<Tag Name="BITag5" Type="DI" Init="0" Address="5" />
<Tag Name="BITag6" Type="DI" Init="0" Address="6" />
<Tag Name="BITag7" Type="DI" Init="0" Address="7" />
<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="CNTTag0" Type="CNT" Init="0" Address="0" />
<Tag Name="CNTTag1" Type="CNT" Init="0" Address="1" />
<Tag Name="CNTTag2" Type="CNT" Init="0" Address="2" />
<Tag Name="CNTTag3" Type="CNT" Init="0" Address="3" />
<Tag Name="AOTag0" Type="AO" Init="0" Address="0" />

```

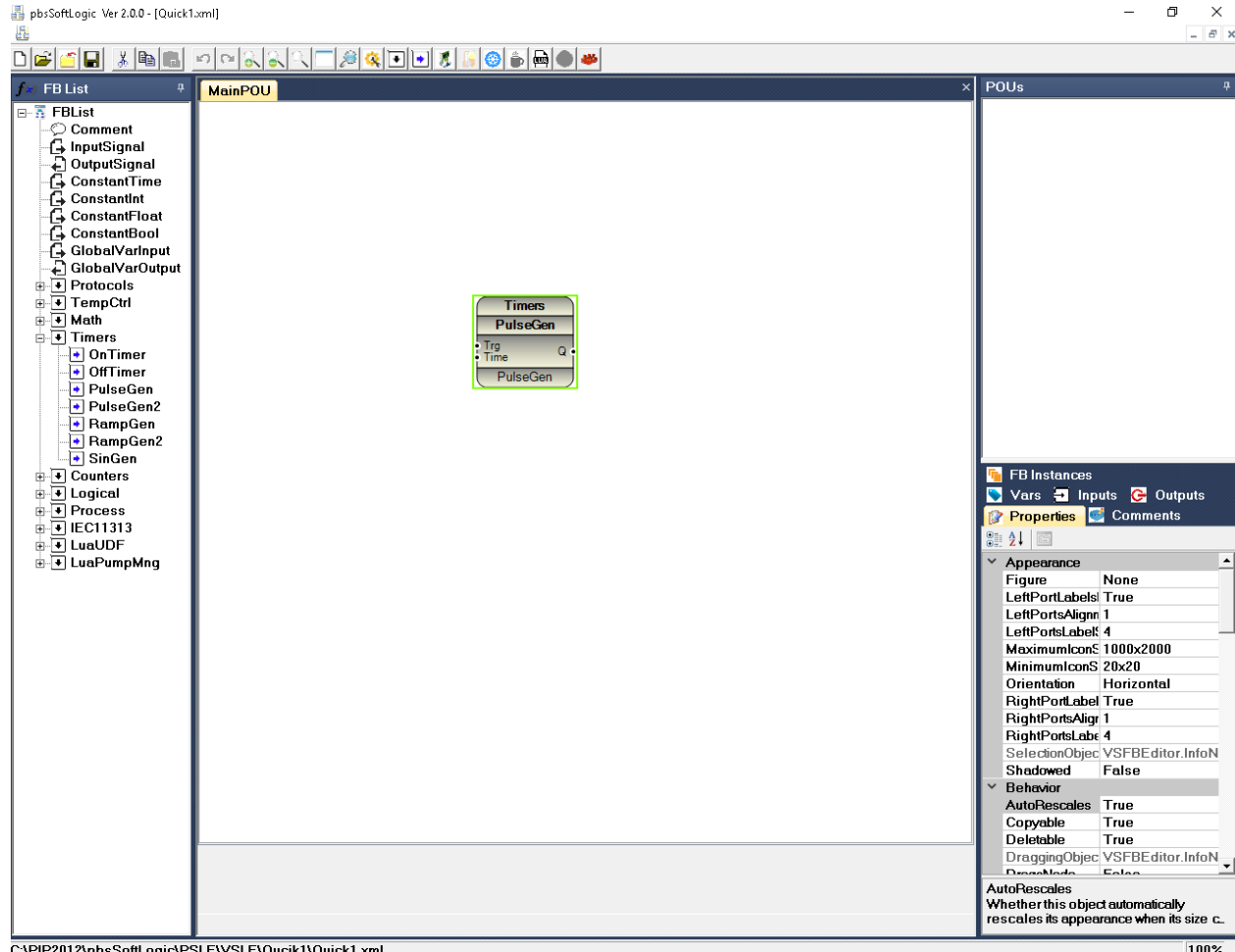
You should not change this file and only you can use it in your logic.

For detail description of above signals please refer to AMS-R3010 RTU configuration chapter.

Step3 :

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

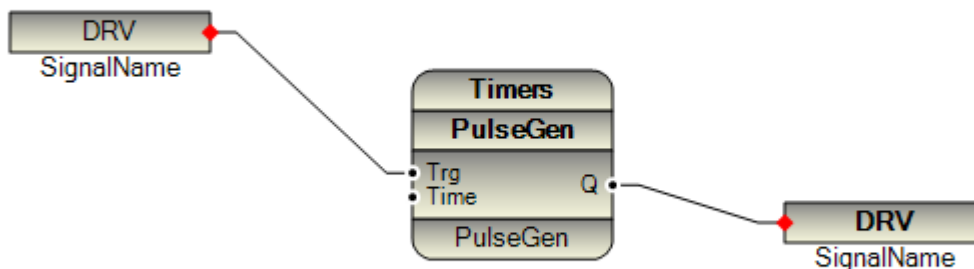
Open Timers Group and select PulseGen . Drag and Drop it to program area .



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

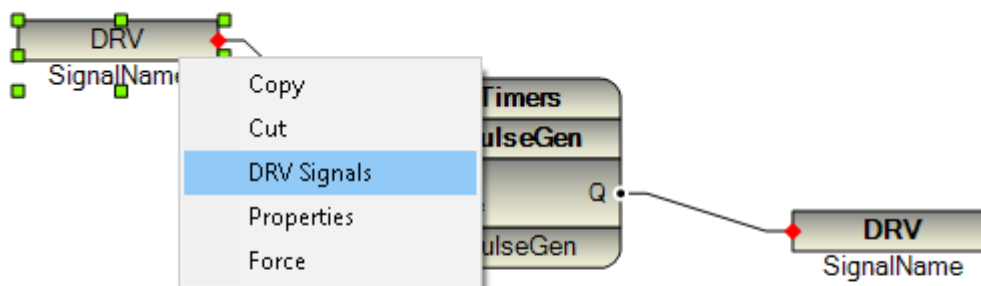
When Trg input is changing from low to high (0 to 1), Pulse train will start at Q output with Low and High Duration equal to Time input.

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.

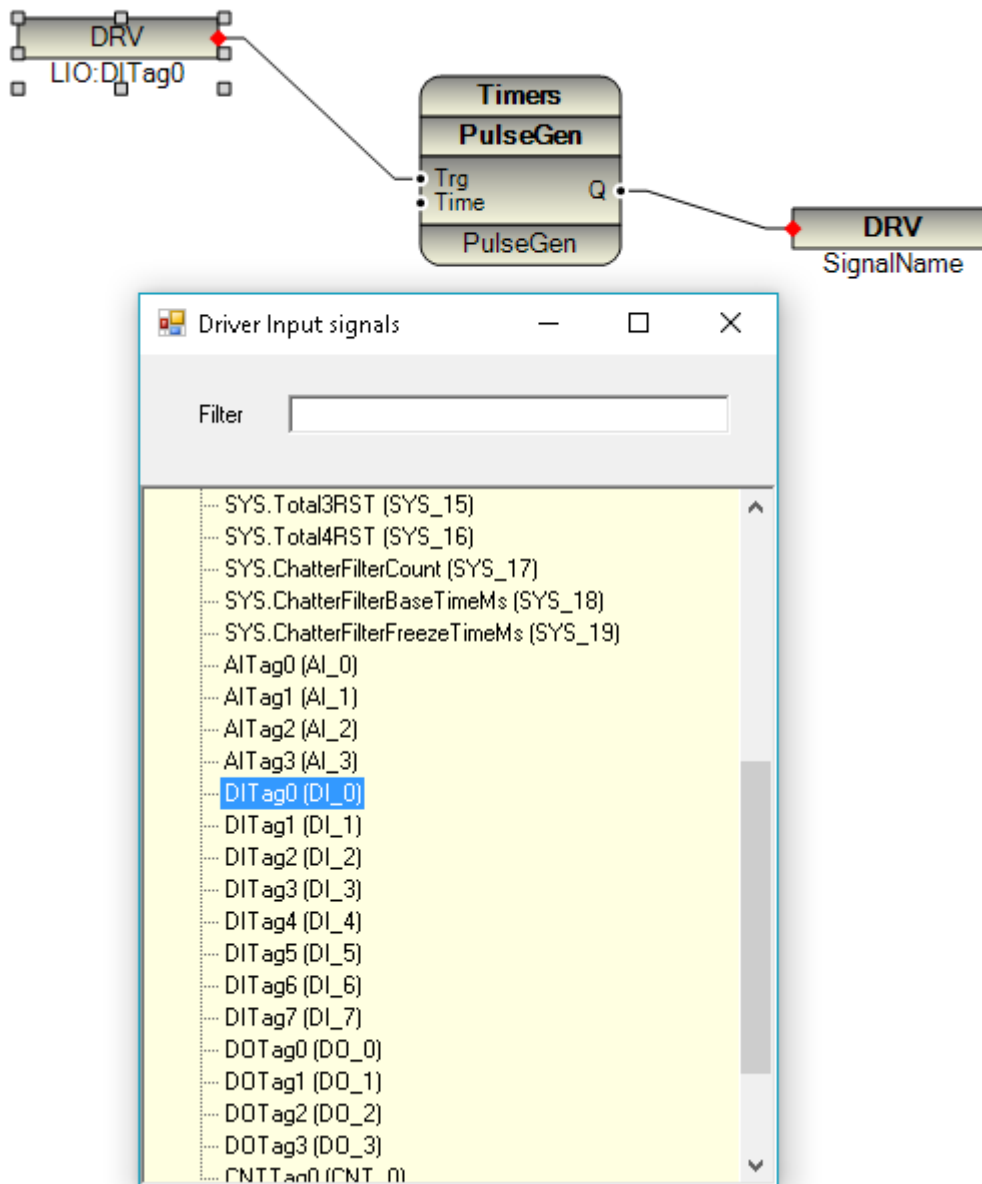


When an Input Port is not connecting to any signal, it will take default value that is preset for each FB (you can change FB Input Default values).

Click on InputSignal which is connected to Trg Input of PulseGen FB . Right click and select Driver Signals Option :



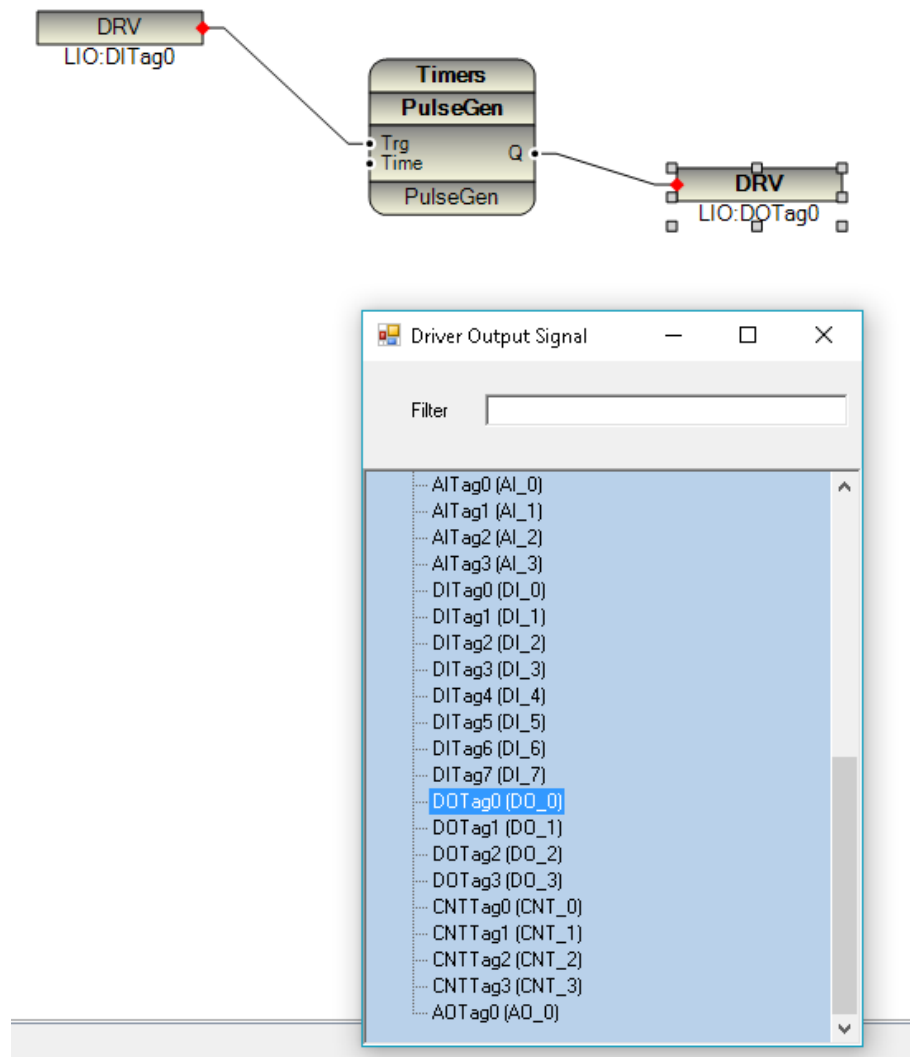
You can see list of defined driver for this project. Open LIO Driver and Double click on DI Tag0 Signal.



Name of signal is assigned to Input Signal Box that is connected to Trg Input of PulseGen Function Block.

Name of Signal is combination of Driver Name + “.”+Signal Name

Click on Output Block Box that is connected to Q Output of Function Block.



Click on Save and compile button at top.



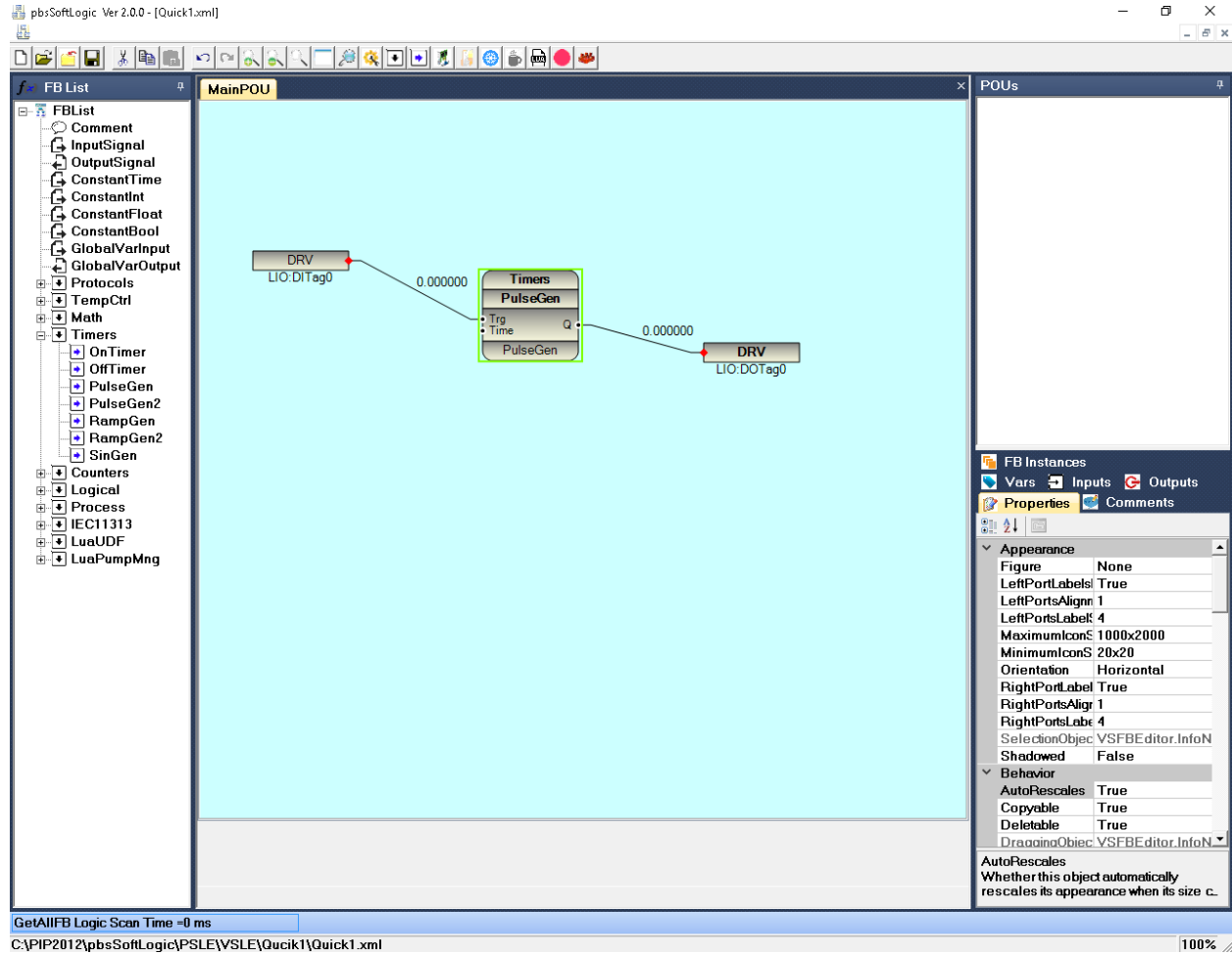
It will save and compile your logic .

Step4: simulate your logic.

Click on Simulate button at top.



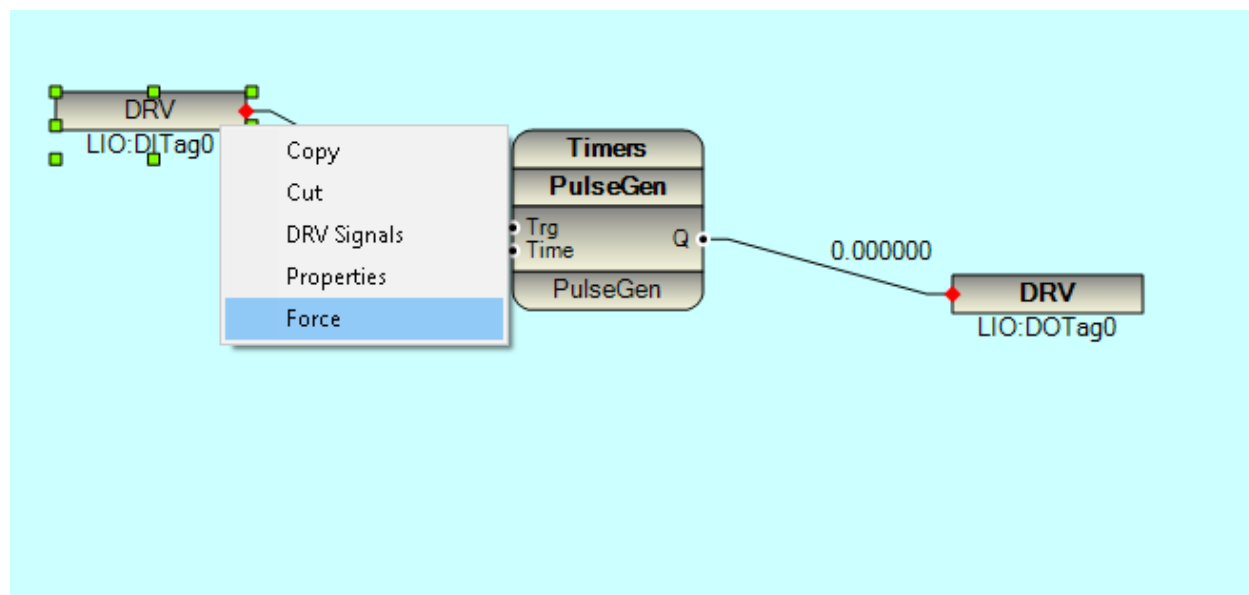
Your logic will change like following:



Online Indicator will start to blink and you can see real logic scan time at bottom part of page.

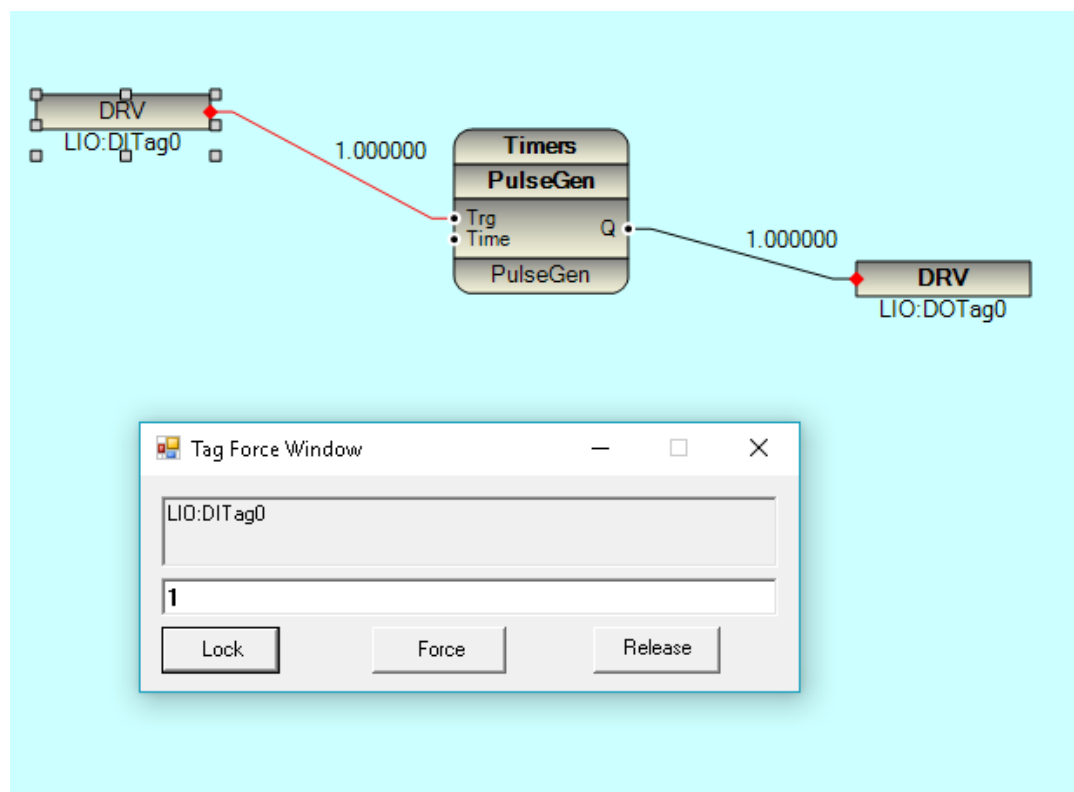
Value of all signals is showing in the link connections between elements.

Right click on LIO:TagDIO Signal and select Force option :



You can force all Input Signals of Function blocks . It is not forcing LIO:DITag0 Signal , it will force Trg Input of PulseGen Function block .

Force signal and you can see output Signal is star to toggle between 0 and 1 .



For proper working of Simulator you need to do following configuration :

- To Be sure there are no any removable Media like USB Disk , ... to your PC
- Install RAMDisk Driver from pbsSoftLogic \RamDisDrv Directory
- Simulator is using RAM Disk to keep Static data of function blocks.
- Open options.xml file in pbsSoftLogic Directory and set RAMDisk Path TempPath option .

<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>

Simulator is runtime kernel of pbsSoftLogic for Win32 . you can see when you click on Simulator button and task bar a new program will run .

```

C:\PIP2012\pbsSoftLogic\PSLE\Win32simulation\psleWin32simulation.exe
PSLE For Win32 Started ...
Kernel Runtime Version 1.7.0
Kernel Runtime Date 10/7/2014
www.pbscontrol.com
app_path=C:\PIP2012\pbsSoftLogic\PSLE\win32simulation\
lmp_Path=C:\PIP2012\pbsSoftLogic\PSLE\win32simulation\lmp\pbsLMP.dll
Logic_Cfg_Path=C:\PIP2012\pbsSoftLogic\PSLE\win32simulation\logic.cfg
RTU Target is PBS-2008RTU
Start Loading Logic...
Logic_Path=C:\PIP2012\pbsSoftLogic\PSLE\win32simulation\logic.c11
TmpNumberOfFB=1
start to add FBs for lmp
VMInit Started...
TmplibPathName=C:\PIP2012\pbsSoftLogic\PSLE\win32simulation\fblib\Timers.dll
LoadLibrary = C:\PIP2012\pbsSoftLogic\PSLE\win32simulation\fblib\Timers.dll ,Error=0
Timers ,PulseGen ,0
VMInit done
pbsMakeLinkLogic_TagsInputs Done
pbsMakeLinkLogic_TagsOutputs done
Logic is loaded NumberOfFB=1
  
```

Simulator is located at pbsSoftLogic\ Win32simulation Directory.

When you click Simulator button, pbsSoftLogic will copy Quick1.c11 and Quick1.lx to Win32Simulation directory and change name to logic.c11 and logic.cfg.

Also pbsSoftLogic will copy all Lua Codes from VSLELib to Win32Simulation\fbllib Directory.

Then psleWin32simulation.exe will run by platform. If psleWin32simulation.exe is already running, first it will close and run it again.

Warm Update and Clod update buttons are not supported for simulation because each time you run simulator it will run Simulator from scratch.

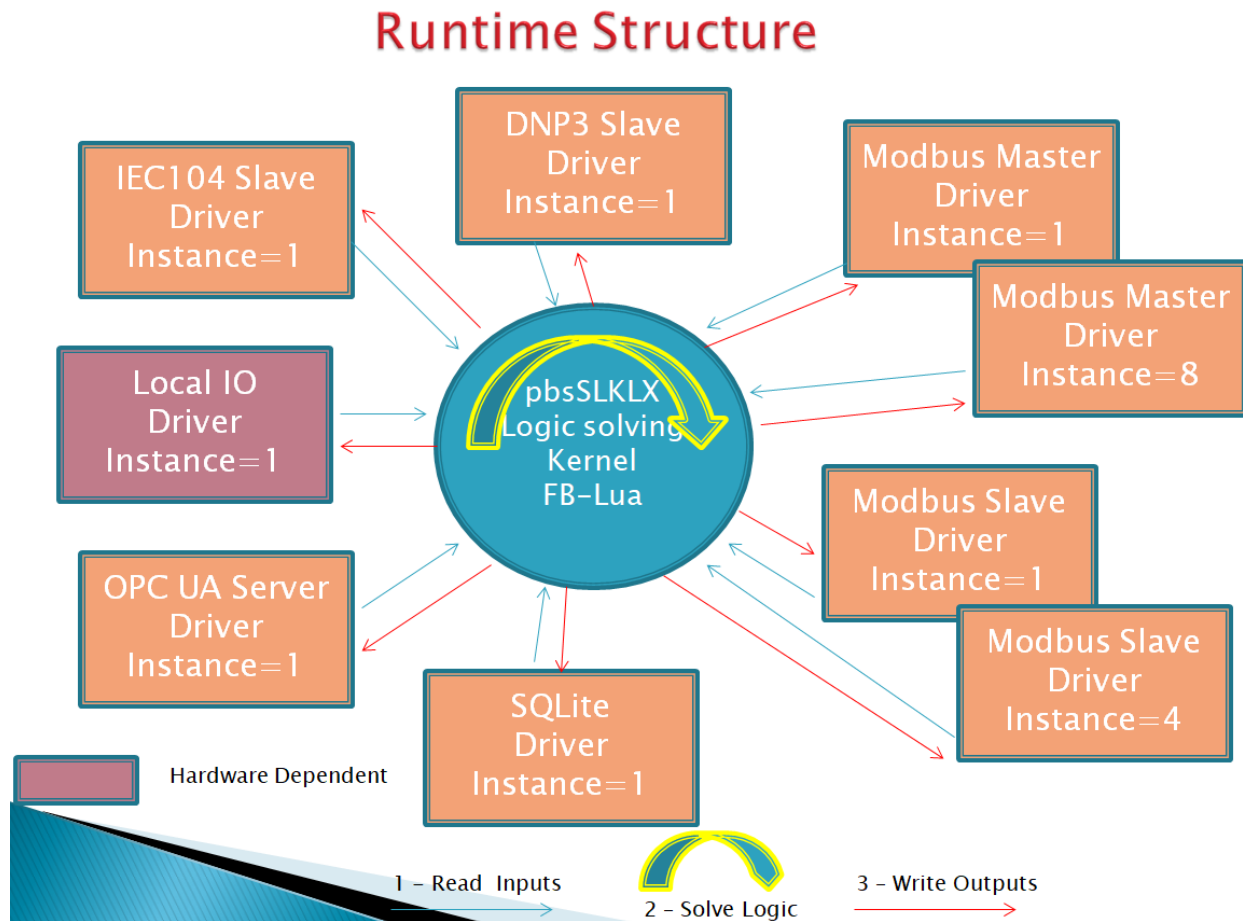
6 – Runtime Kernel for Linux, transferring License to Controller and working with Linux

In this chapter will talk about runtime structure of pbsSoftLogic inside RTU . pbsSoftLogic runtime kernel is based on very simple concept .

pbsSLKLX : this is main application in linux which will start automatically when RTU is booted or you can load it manually for Diagnostic purpose .

Communication Drivers: Different communication drivers which are supported by pbsSoftLogic .

In following figure you can see how above components are communicating with each other.



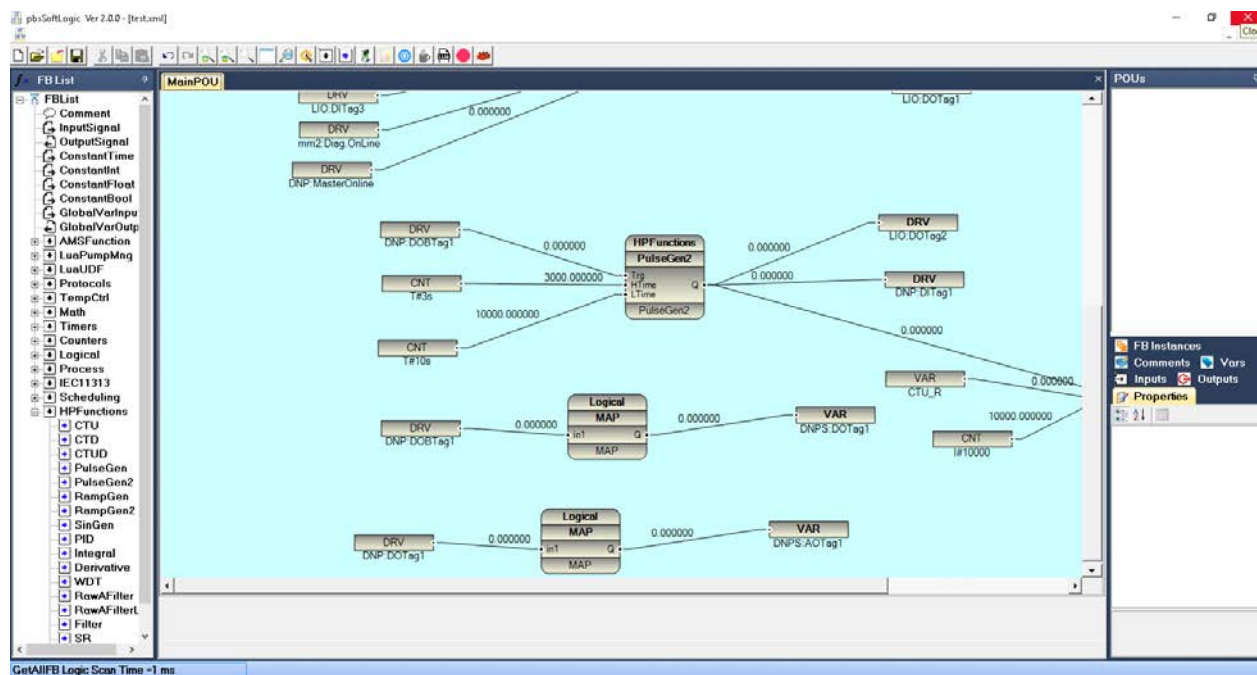
When you run pbsSLKLX following steps will done in RTU :

- 1- pbsSLKLX will load /home/pbsLX/logic.cfg file . This file contains all defined communication protocol for active project.
- 2 - pbsSLKLX will load communication driver library in to memory dynamically .

- 3 - pbsSLKLX get Driver parameters from logic.cfg file and pass one by one to Communication Driver library
- 4 - pbsSLKLX will add Communication Blocks , Slots and Finally Driver Tags
- 5 - pbsSLKLX will call pbsInit Function to initialize Communication Driver
- 6 - pbsSLKLX will repeat steps 2 , 3, 4, 5 for all defined drivers in logic.cfg
- 7 - pbsSLKLX will start to read input Tags for all Communication Drivers by calling pbsReadTag
- 8 - pbsSLKLX solve one time RTU logic
- 9 - pbsSLKLX will write to Communication Drivers all Output Tags by calling pbsWrite method .
- 10 – Repeat Steps 7 , 8 , 9

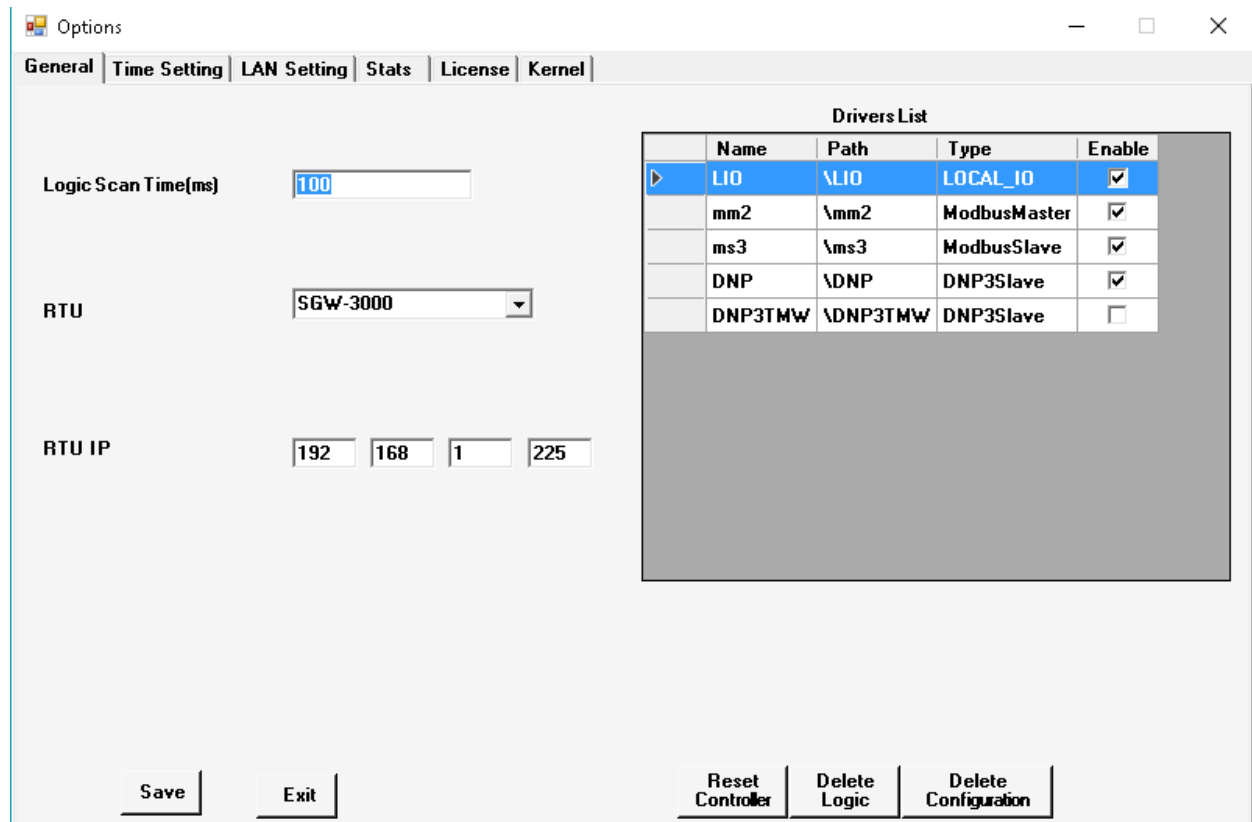
As it clear from above sequence, Communication Drivers has unified API interface for communication with pbsSLKLX .

Duration time for executing steps 7 , 8 , 9 is RTU Real Logic Scan Time . You can see Real Logic Scan Time when you connect to RTU by pbsSoftLogic Engineering . Real Logic Scan Time is shown in bottom of page at left side. In following figure Real Logic Scan time is 1 mse.



You couldn't control Real Logic Scan Time , but you can set Logic Scan Time in Project Setting Page .

For above sample Logic Scan Time is 100 msec .



It means every 100 msec , RTU will Read All Defined Driver Inputs (Step 7) , Execute Logic (Step 8) and Write all Define Driver Outputs (Step 9) . Time for executing Step 7 , 8 ,9 is 1 Msec . So pbsSLKLX CPU thread is at sleep for 99 msec .

pbsSLKLX has only one thread for Reading Driver Input Tags , Execute RTU logic and Write Driver Output Tags . But there is no any limitation in Drivers Library to get many CPU threads.

When you reduce Logic Scan Time and make it close to Real Logic Scan Time , then RTU CPU Usage will increase . So you should select Logic Scan Time Based on your Process Condition and it should be always more than Real Logic Scan Time.

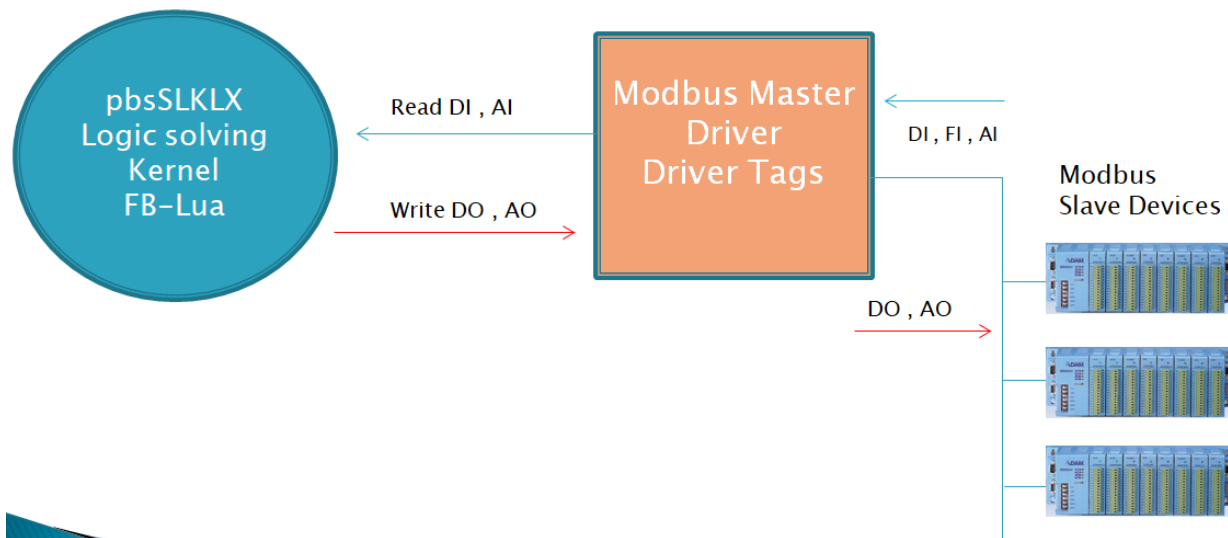
You can use less than 1 msec for logic scan time , 0.1 or 0.01 msec is also possible to Set .

Please notice that pbsSLKLX Scan Time is different than Communication protocol scan time. You can Read Modbus Slaves every second , but pbsSLKLX will read/write to driver tags every 10 msec as an example .

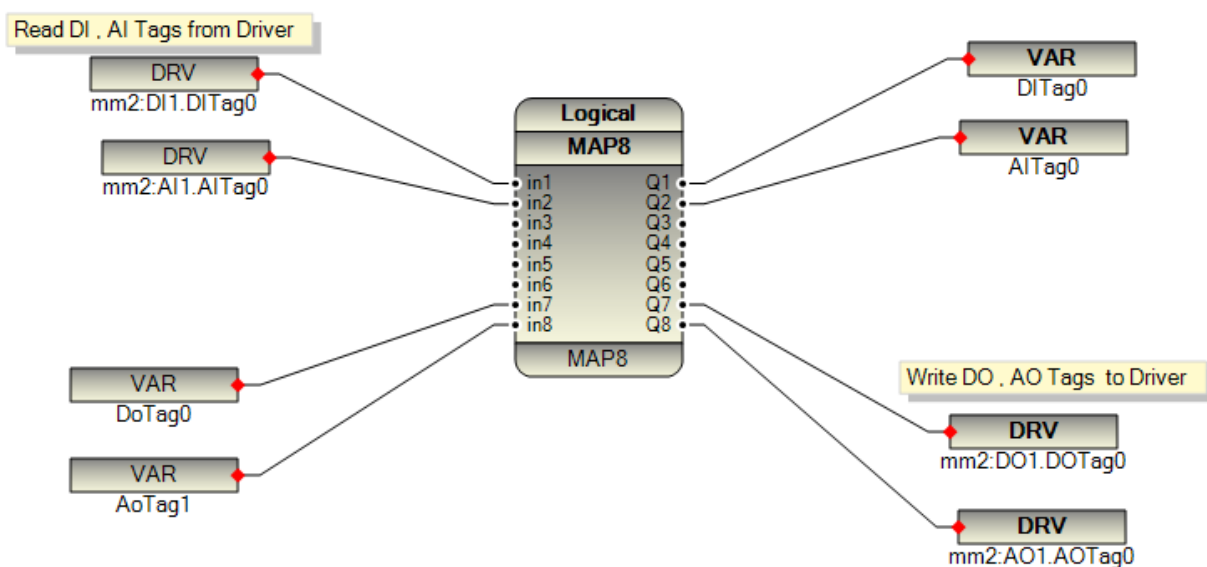
Slave and Master Drivers : There is a conceptual difference in Master Drivers Like Modbus TCP/RTU Master , IEC101 /103 Master and slave drivers like Modbus TCP/RTU Slave , DNP3 Slave , IEC101/104 Slave .

Master Drivers : In following figure you can see Modbus Master Driver is reading Digital and Analog Inputs from Slave devices and Write Digital and Analog Output to Slave devices .

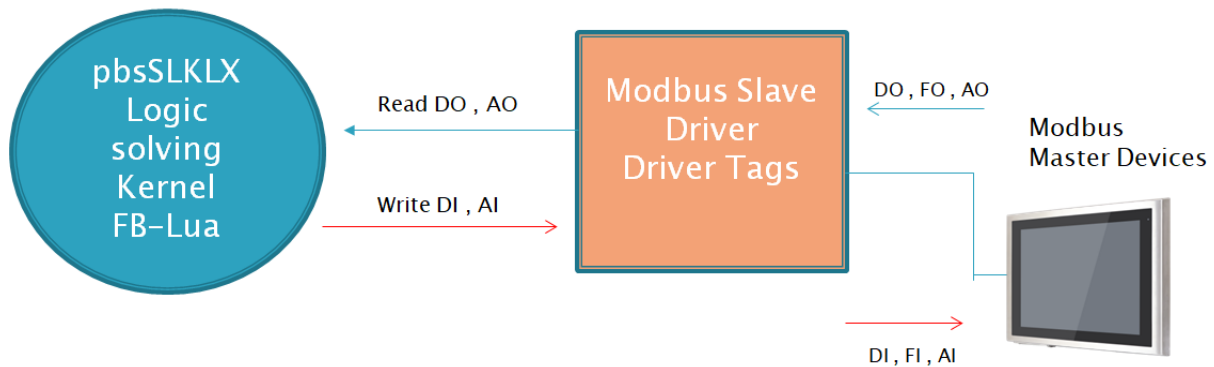
pbsSLKLX is Reading input Digital and Analog Tags from Driver and Write Output Digital and Analog Tags to Driver .



In RTU Logic you should use Input Tags in Left Side of Function Blocks to read from driver and use Output Tags in Right Site of Function Blocks to Write to Driver .

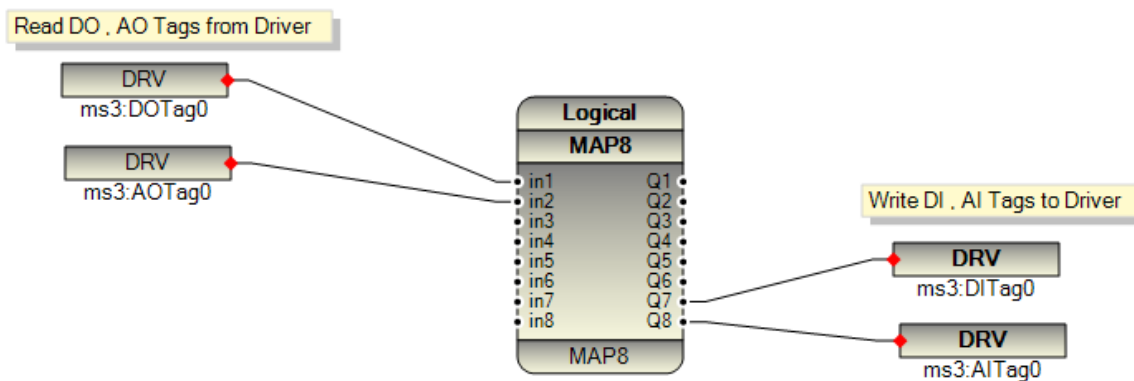


Slave Drivers: In following figure you can see Modbus Master Devices (HMI Panel, SCADA Software's) are read Digital and Analog Tags from Modbus Slave Driver and Write Digital and Analog Output Tags to Modbus slave driver.



pbsSLKLX is Writing Digital and analog Tags to Modbus Slave Driver (you will update by RTU Logic Modbus Slave DI , AI Tags for reading by Modbus Master Device) and pbsSLKLX is read Digital and Analog Outputs from Modbus Slave Driver . These Tags are written by external Modbus Master Device to Modbus Slave Driver.

In Slave Drivers , you Should use DO , AO tags (Read From Driver) in Left Side of Function Blocks and DI , AI Tags (Write to Driver) in right Side of Function Block .



pbsSoftLogic has two parts :

- 1 - Engineering station. Running on windows Operating system
- 2 – Runtime Engine. Running on Embedded Linux inside RTU

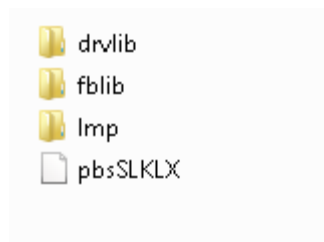
In this section we will talk about Linux Runtime engine.

You can download latest Linux runtime engine for different different controllers from <http://www.pbscontrol.com> 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/fbllib/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 .

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



For transferring pbsLX directory to controller do following tasks :

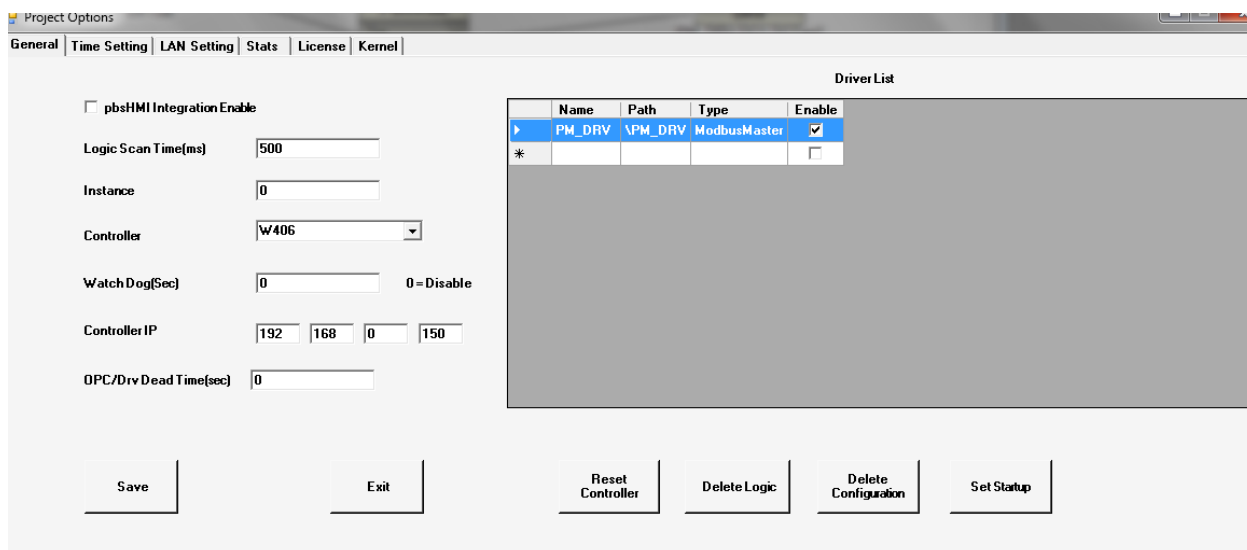
- 1 – Open project setting page and click on kernel Tab. To be sure that Controller IP address is correct on General Tab.
- 2 – Click on Browse Button and select pbsLX Directory that you want to transfer to controller. To be sure that you select correct runtime Kernel for your controller.



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:

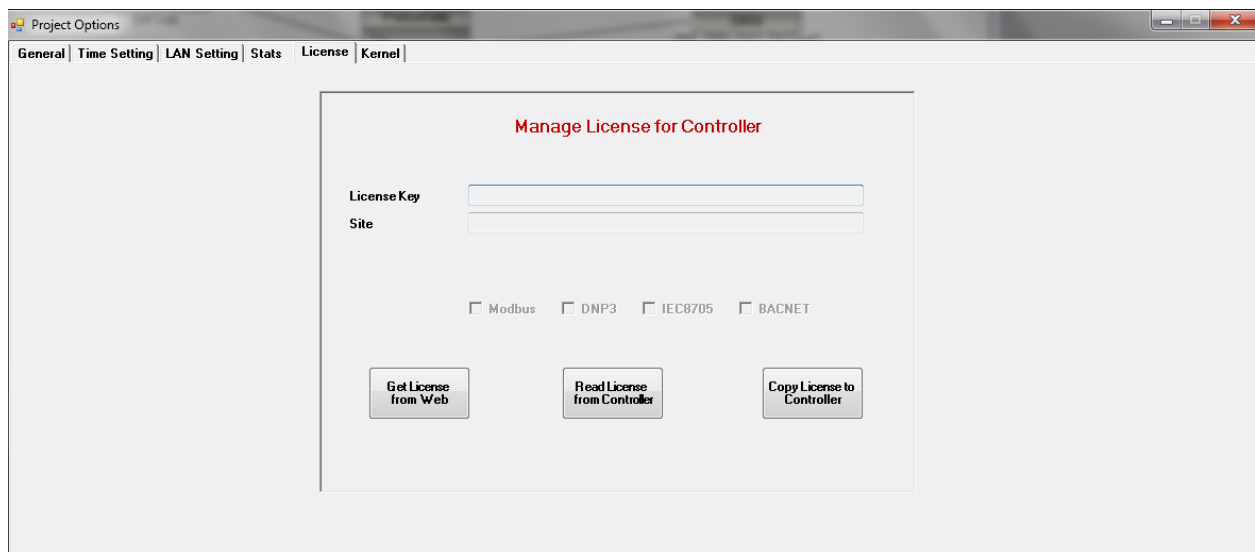
- RTU/PLC functionality and Modbus Master/Slave protocol. This is basic license for each controller.
- DNP3 Slave License.
- IEC870-5-101/104 Slave License

- 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 boxes 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.

Working with Linux OS

pbsSoftLogic Runtime kernel is running on Linux Operating system . So you need to have basic knowledge about Linux . We will review all necessary tools and Linux commands which is used in pbsSoftLogic in this section .

pbsSoftLogic Runtime kernel is developed by ANSI C and it is possible to port it to any Operating system . We already port it for QNX , Win32 and WinCE .But our default OS is embedded linux .

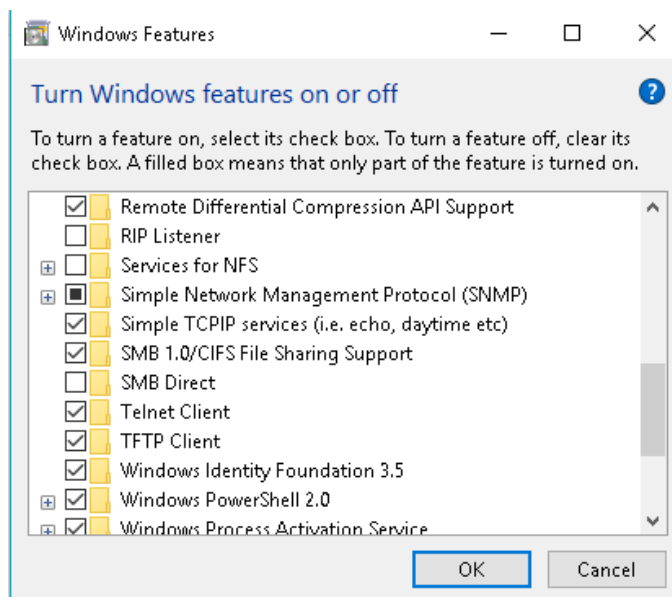
pbsSoftLogic is transferring Configuration and Logic file to RTU by FTP protocol . So you need to have FTP Server running inside RTU .

pbsSoftLogic is using Root user to transfer files , so check inside RTU Root user has access in /etc/ftpuser file . The best way to use FTP is using FileZilla Client Software . This is free utility and you can get latest version from <https://filezilla-project.org/>

Because Windows and Linux file format are not identical, so you need to use intelligent Editor like Notepad++ to edit Linux Configuration files in Windows and writing again files to linux . Download Notepad++ from <https://notepad-plus-plus.org/>

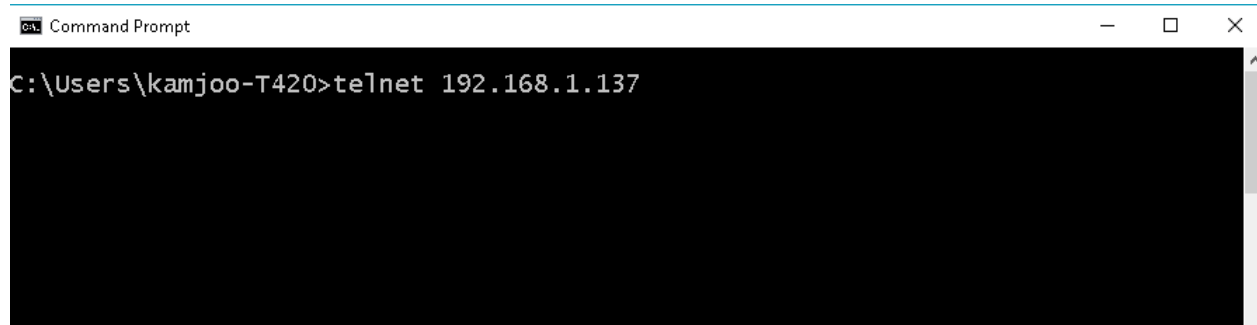
pbsSoftLogic is using Telnet to set Project setting like RTU Date and time , Restarting RTU , So you need to have running Telnet Server inside RTU.

For windows engineering station you need to enable Telnet Client utility. It is located in Control Panel /program and Features /Turn Windows Features On or Off.



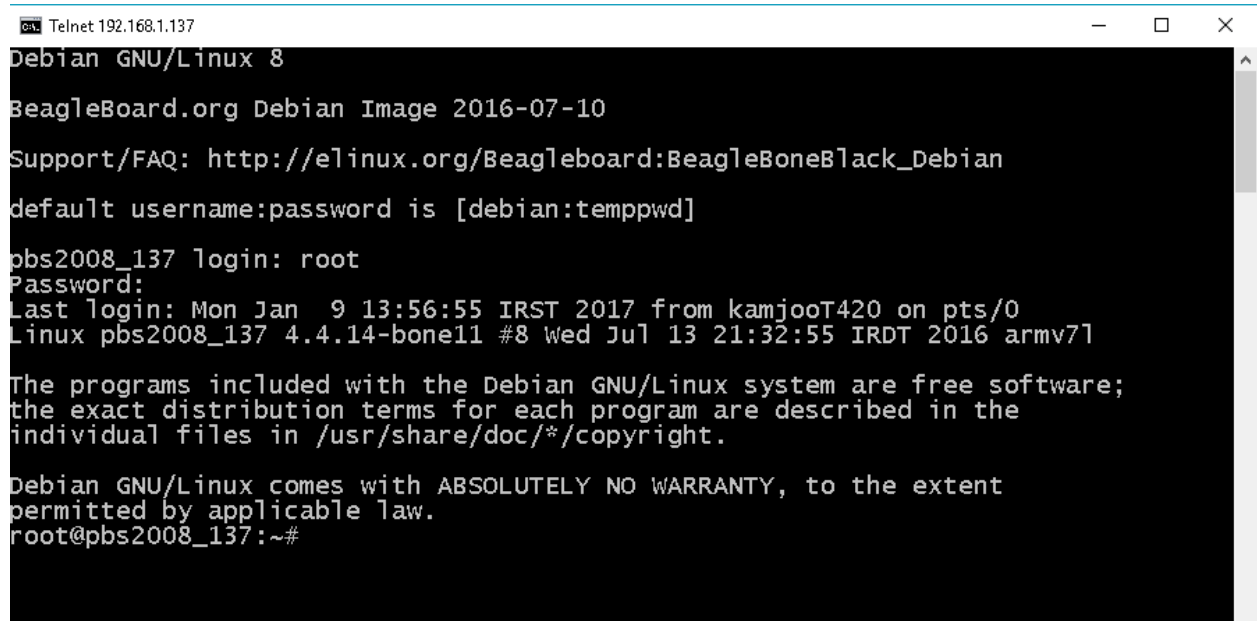
With telnet utility you can connect to RTU by TCP/IP network and manage RTU or change configuration remotely . Telnet is command based like old MS DOS command prompt .

In windows Engineering Station open command window and execute telnet command with RTU IP Address as following:



```
Command Prompt
C:\Users\kamjoo-T420>telnet 192.168.1.137
```

By default user name and password of pbsSoftLogic based RTUS is root, root.

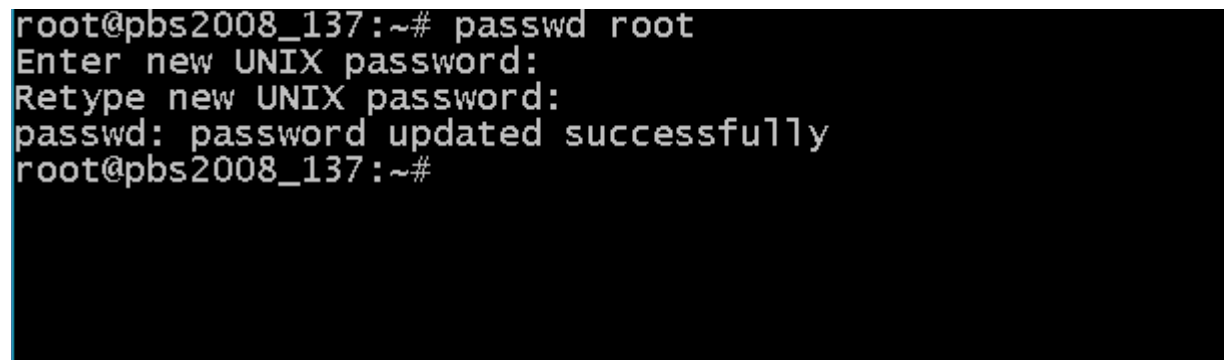


```
Telnet 192.168.1.137
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-07-10
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:tempwd]
pbs2008_137 login: root
Password:
Last login: Mon Jan  9 13:56:55 IRST 2017 from kamjooT420 on pts/0
Linux pbs2008_137 4.4.14-bone11 #8 Wed Jul 13 21:32:55 IRDT 2016 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pbs2008_137:~#
```

You can change password of root user by passwd command in linux .



```
root@pbs2008_137:~# passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@pbs2008_137:~#
```

You can change linux directory by cd command:

 Telnet 192.168.1.137

```
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX#
```

You can see list of files by ls command:

```
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# ls
ReadmeV2RC8.txt  drvlib  lmp      logic.cfg  startup.sh
ResetFactory     fblib   logic.c11 pbsSLKLX
root@pbs2008_137:/home/pbsLX#
```

You can see list of running process by ps aux command:

```
root@pbs2008_137:/home/pbsLX# ps aux | grep pbsSLKLX
root      495  1.9  0.6 80524 3404 ?        S1   14:20   2:41 ./pbsSLKLX
root      3133 0.0  0.2  2076 1124 pts/0    S+   16:39   0:00 grep pbsSLKLX
root@pbs2008_137:/home/pbsLX#
```

Pbs aux will show all process that is running in RTU. But if you are looking for a specific process you can use grep as above example.

Any process in linux has an ID . in above example pbsSLKLX ID is 495 .

You can kill process by its ID with kill command as following :

```
root@pbs2008_137:/home/pbsLX# ps aux | grep pbsSLKLX
root      495  1.9  0.6 80524 3404 ?        S1   14:20   2:41 ./pbsSLKLX
root      3133 0.0  0.2  2076 1124 pts/0    S+   16:39   0:00 grep pbsSLKLX
root@pbs2008_137:/home/pbsLX# kill 495
```

Or ou can kill process by its name with pkill command:

```
root@pbs2008_137:/home/pbsLX# ps aux | grep pbsSLKLX
root      495  1.9  0.6 80524 3404 ?        S1   14:20   2:41 ./pbsSLKLX
root      3133 0.0  0.2  2076 1124 pts/0    S+   16:39   0:00 grep pbsSLKLX
root@pbs2008_137:/home/pbsLX# pkill pbsSLKLX
```

You can see list of running process based on CPU and Memory usage by top command:

```
top - 16:45:51 up 2:25, 1 user, load average: 0.14, 0.11, 0.03
Tasks: 69 total, 2 running, 67 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.4 us, 1.0 sy, 0.0 ni, 96.2 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 506736 total, 90644 used, 416092 free, 12372 buffers
KiB Swap: 0 total, 0 used, 0 free, 43692 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
481	root	20	0	22344	8544	5168	R	3.6	1.7	5:05.01	OpenOpcUaC+
495	root	20	0	80524	3404	1464	S	1.7	0.7	2:47.63	pbsSLKLX
3236	root	20	0	2988	1632	1296	R	0.3	0.3	0:00.04	top
1	root	20	0	4456	3156	2128	S	0.0	0.6	0:06.79	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:04.74	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:+
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.04	watchdog/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kdevtmpfs
10	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kswork
12	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	perf
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khungtaskd
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
15	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioset
19	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
21	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	devfreq_wq
22	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	rpciod
23	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kswapd0
24	root	20	0	0	0	0	S	0.0	0.0	0:00.00	fsnotify_m+
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	nfsiod
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	xfsalloc
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	xfs_mru_ca+
49	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kthrotld
50	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kapmd
53	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	irq/30-480+
54	root	0	20	0	0	0	S	0.0	0.0	0:00.00	bind

You should stop top command by pressing ctrl+c keys.

If you want to run a program from current directory you should use ./ to run application .

 Telnet 192.168.1.137

```
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# ./pbsSLKLX
```

If you use & after program name , application will run in background and you have still Telnet command control .

```
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# ./pbsSLKLX &
```

For deleting files you can use rm command:


```
root@pbs2008_137:~# cd /home/pbsLX/  
root@pbs2008_137:/home/pbsLX# rm logic.c11  
root@pbs2008_137:/home/pbsLX# rm logic.cfg  
root@pbs2008_137:/home/pbsLX#
```

In above example pbsSoftLogic , Logic and hardware configuration files are removed .

For making a new directory you can use mkdir command.

```
root@pbs2008_137:/home/pbsLX# mkdir test  
root@pbs2008_137:/home/pbsLX#
```

When you copy an executable file to RTU and wants to run it , you need to make it as executable by chmod command :


 Telnet 192.168.1.137

```
root@pbs2008_137:/home/pbsLX# chmod +x pbsSLKLX
```

If you want to see RTU hardware information you can use /proc directory files:

```
root@pbs2008_137:/home/pbsLX# cat /proc/cpuinfo  
processor       : 0  
model name     : ARMv7 Processor rev 2 (v7l)  
BogoMIPS      : 996.14  
Features       : half thumb fastmult vfp edsp thumb2 neon vfpv3 tls vfpd32  
CPU implementer : 0x41  
CPU architecture: 7  
CPU variant    : 0x3  
CPU part       : 0xc08  
CPU revision   : 2  
  
Hardware       : Generic AM33XX (Flattened Device Tree)  
Revision      : 0000  
Serial        : 0000000000000000  
root@pbs2008_137:/home/pbsLX#
```

date command will show or set current Date Time of RTU :

 Telnet 192.168.1.137

```
root@pbs2008_137:/home/pbsLX# date  
Mon Jan  9 17:13:30 IRST 2017  
root@pbs2008_137:/home/pbsLX#
```

Uptime shows for how long RTU is not restarted.

```
root@pbs2008_137:/home/pbsLX# uptime
17:14:45 up 25 min, 1 user, load average: 0.16, 0.06, 0.01
root@pbs2008_137:/home/pbsLX#
```

You can see flash memory and mount derives status by `df -h` command

```
root@pbs2008_137:/home/pbsLX# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            10M    0    10M   0% /dev
tmpfs           99M   4.3M   95M   5% /run
/dev/mmcblk0p1  3.6G  1.1G  2.3G  32% /
tmpfs           248M   4.0K  248M   1% /dev/shm
tmpfs           5.0M    0   5.0M   0% /run/lock
tmpfs           248M    0  248M   0% /sys/fs/cgroup
tmpfs           32M   20K   32M   1% /mnt/ramdisk
root@pbs2008_137:/home/pbsLX#
```

You can see memory status by `cat /proc/meminfo` command

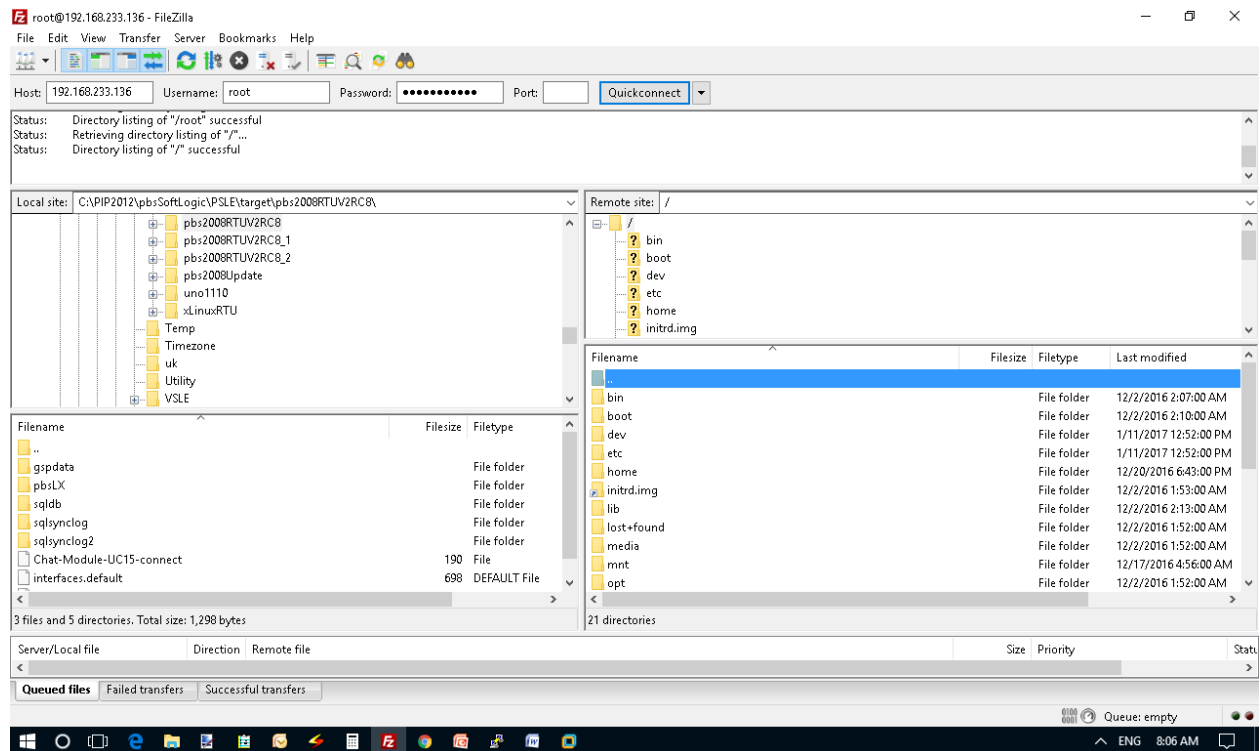
```
root@pbs2008_137:/home/pbsLX# cat /proc/meminfo
MemTotal:      506736 kB
MemFree:       432860 kB
MemAvailable:  464880 kB
Buffers:       7980 kB
Cached:        31712 kB
SwapCached:    0 kB
Active:        35992 kB
Inactive:      19456 kB
Active(anon):  15876 kB
Inactive(anon): 4284 kB
Active(file):  20116 kB
Inactive(file): 15172 kB
Unevictable:   0 kB
Mlocked:       0 kB
HighTotal:     0 kB
HighFree:      0 kB
LowTotal:      506736 kB
LowFree:       432860 kB
SwapTotal:     0 kB
SwapFree:      0 kB
Dirty:         104 kB
Writeback:     0 kB
AnonPages:     15768 kB
Mapped:        15580 kB
Shmem:         4408 kB
Slab:          13612 kB
SReclaimable:  7844 kB
SUnreclaim:    5768 kB
KernelStack:   720 kB
```

Working with FileZilla

You can use FileZilla client utility to explore and edit RTU Files and directories.

Download filezilla from <https://filezilla-project.org/>

Run filezilla client you will see following page :



Type RTU IP at host field. Type root and root password in user name and password fields .

All RTU directories are showing at right panels and your PC directories at left s panels.

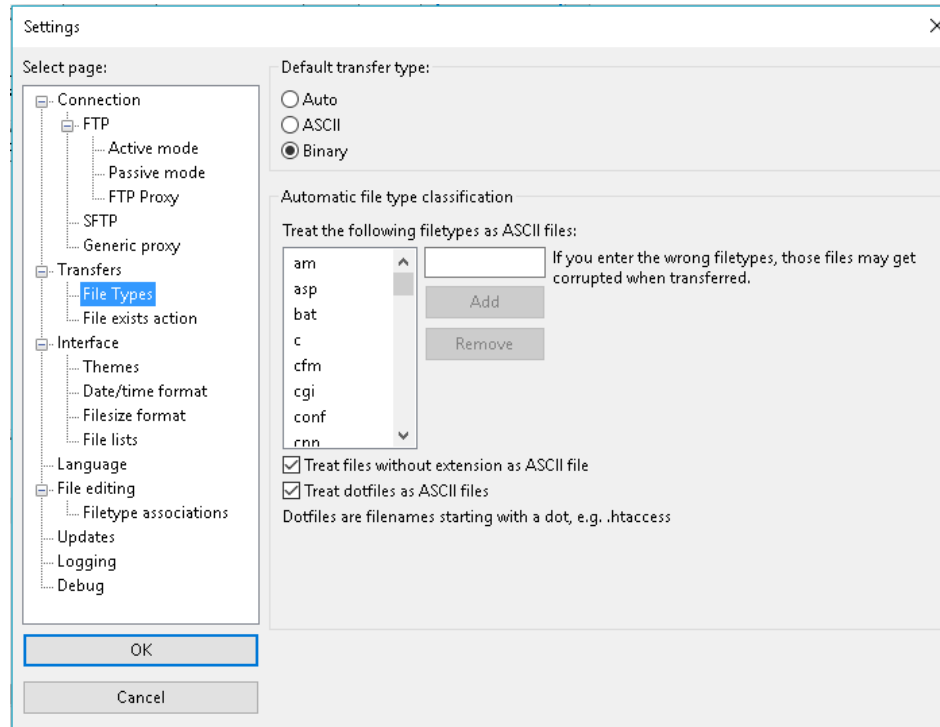
Double click on home directory. You will go inside home directory. Double click on pbsLX . pbsLX directory is runtime kernel of pbsSoftLogic for Linux Operating system .

Filename	Filesize	Filetype	Last modified
..			
drvlib		File folder	1/11/2017 1:00:00 PM
fblib		File folder	1/11/2017 1:00:00 PM
lmp		File folder	1/11/2017 1:00:00 PM
logic.c11	1,349	C11 File	1/11/2017 1:05:00 PM
logic.cfg	24,893	CFG File	1/11/2017 1:05:00 PM
pbsSLKLX	474,232	File	1/11/2017 1:00:00 PM

If you install debian X86 Linux on your pc over VMWare or Virtual Box, then you can easily use pbsSoftLogic Debian X86 Runtime and test all functionalities of pbsSoftLogic on your PC.

pbsSoftLogic Kernel for UNO1252G is Debian Runtime for X86 CPU .

Note : for transferring files between Windows and Linux Systems , always set Transfer File Type to Binary. you can find this option in Edit Menu , Setting menu and Transfers Segment .



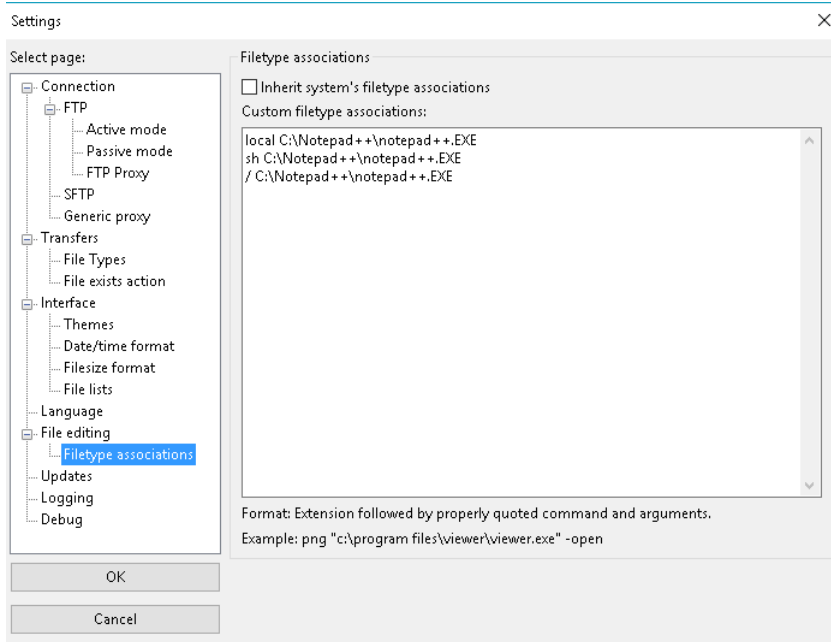
By default it is set to “Auto” that is damaging Linux files at transfer time from Windows to RTU.

For editing RTU configuration files in windows you need to use NotePad++ Editor to not damage Text file format when transfer to windows System .

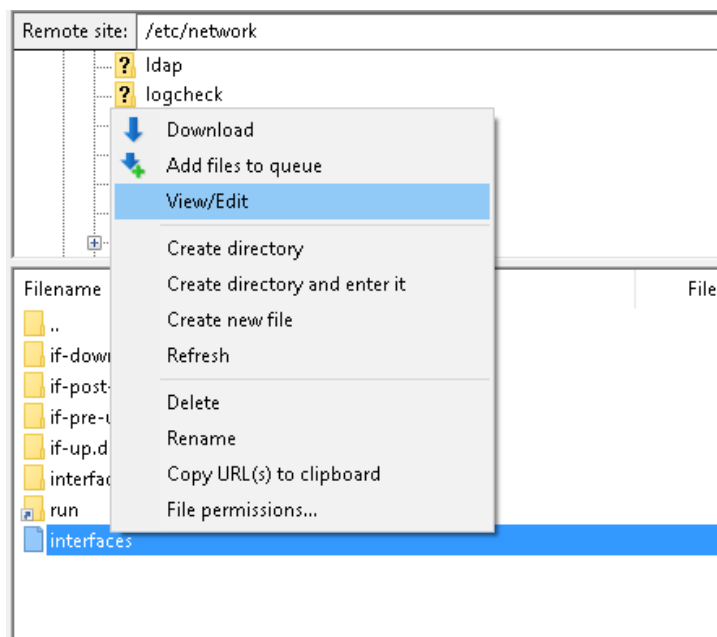
Install NotePad++ utility from <https://notepad-plus-plus.org/>

At first time that you View/Edit any Linux Configuration file , Filezilla will ask you for Custom Editor .

In this Stage set Nodepad++ as default editor in Filezilla . This will change File Editing Option in Setting page as following :



For changing Network Interfaces in RTU , View/Edit /etc/network/interfaces file .



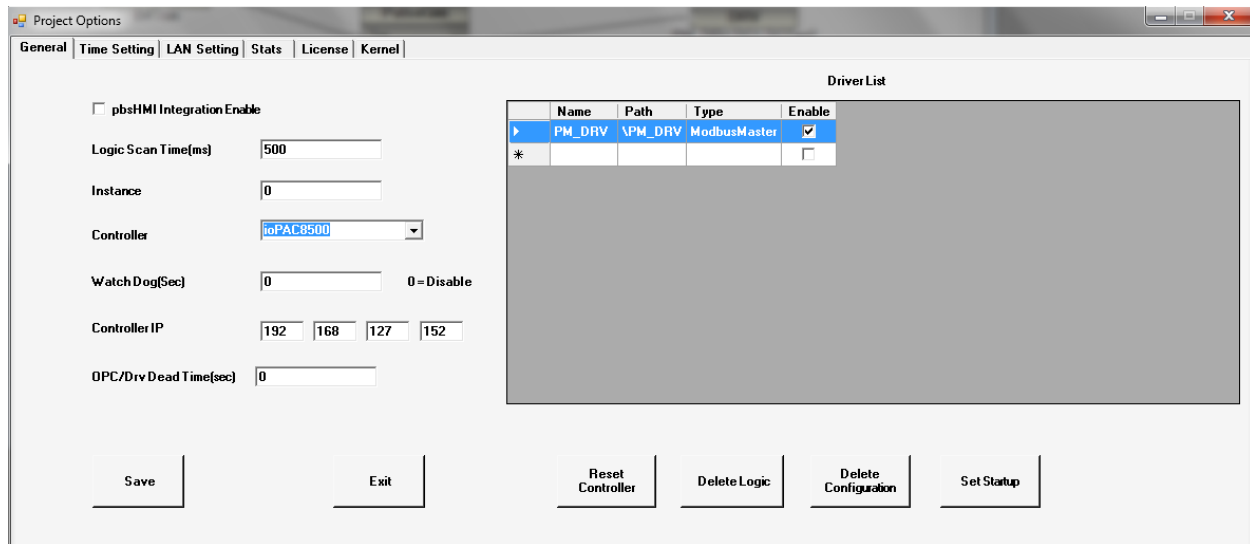
For transferring files from RTU to Windows, Select File, Right click on name of File or Directory and run "Download" command.

You can delete or Rename files inside RTU by selecting file, right click on name of file and select Delete or Rename Commands.

7 – Project Settings facilities

There are many facilities in setting page in pbsSoftLogic Editor.

Open Setting Page you can see following tabs:

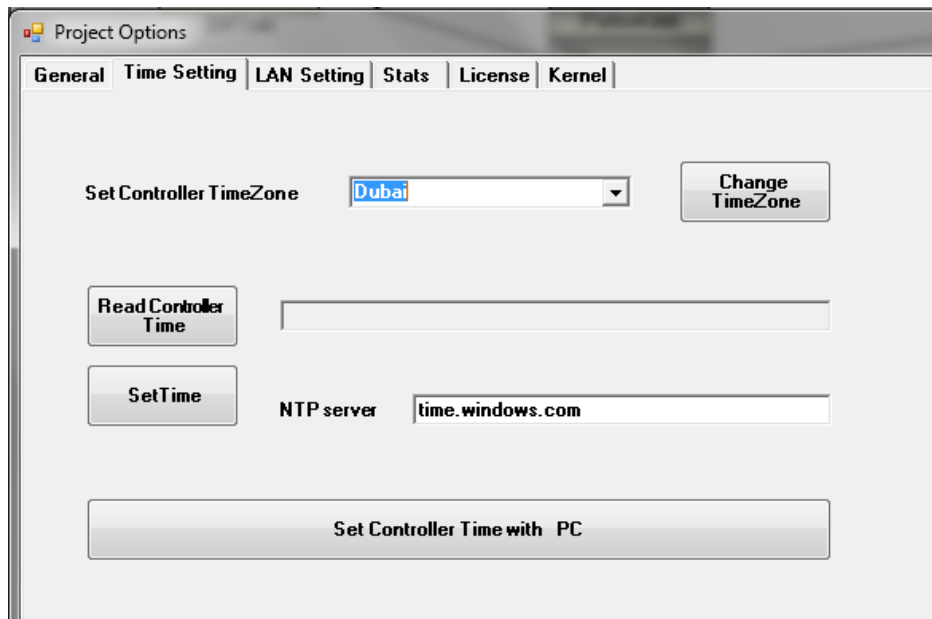


- General
- Time Setting
- LAN Setting
- Stats
- License
- Kernel

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:



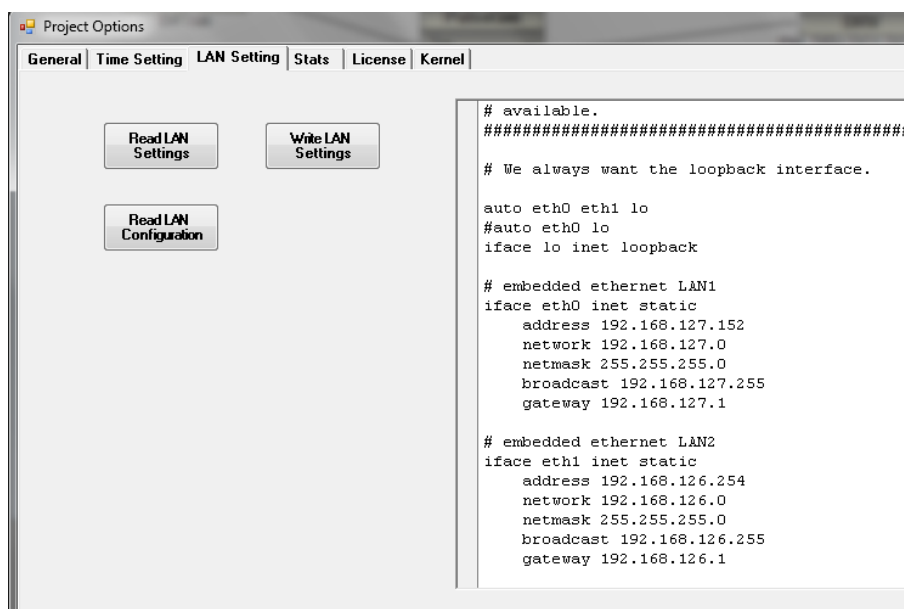
Set Controller Time Zone: Select your location from list box, and click on Change TimeZone .

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

Set Time : will set Controreller 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

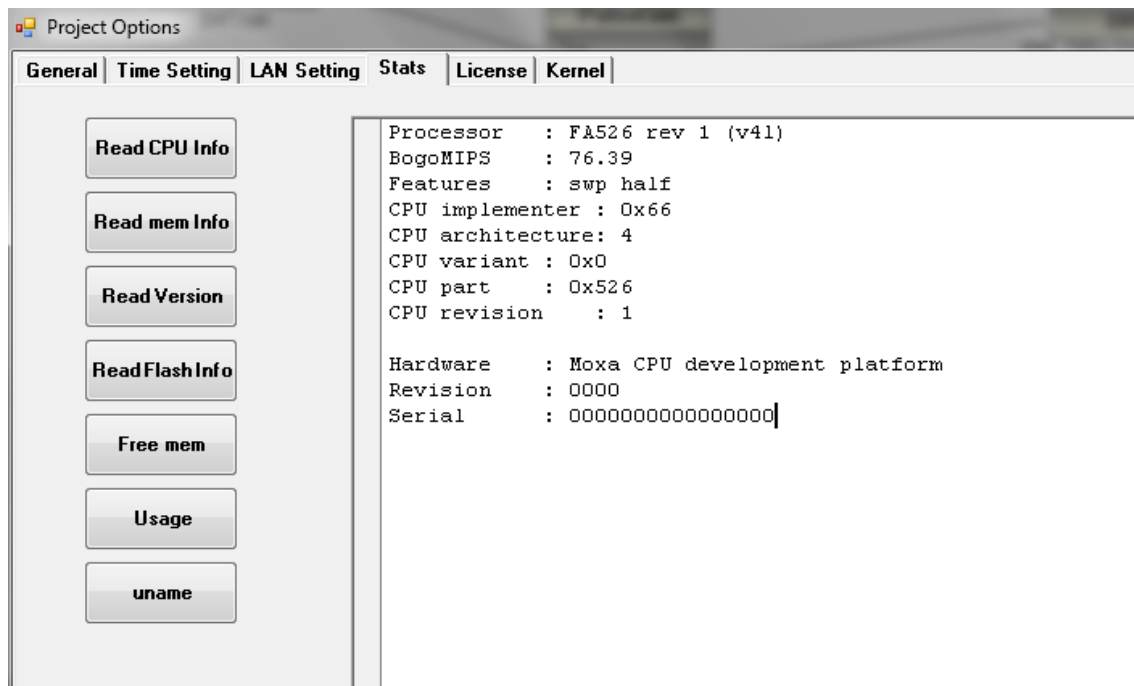
Read LAN Configuration: it will read current ALN configuration fro controller.

For changing controller IP address:

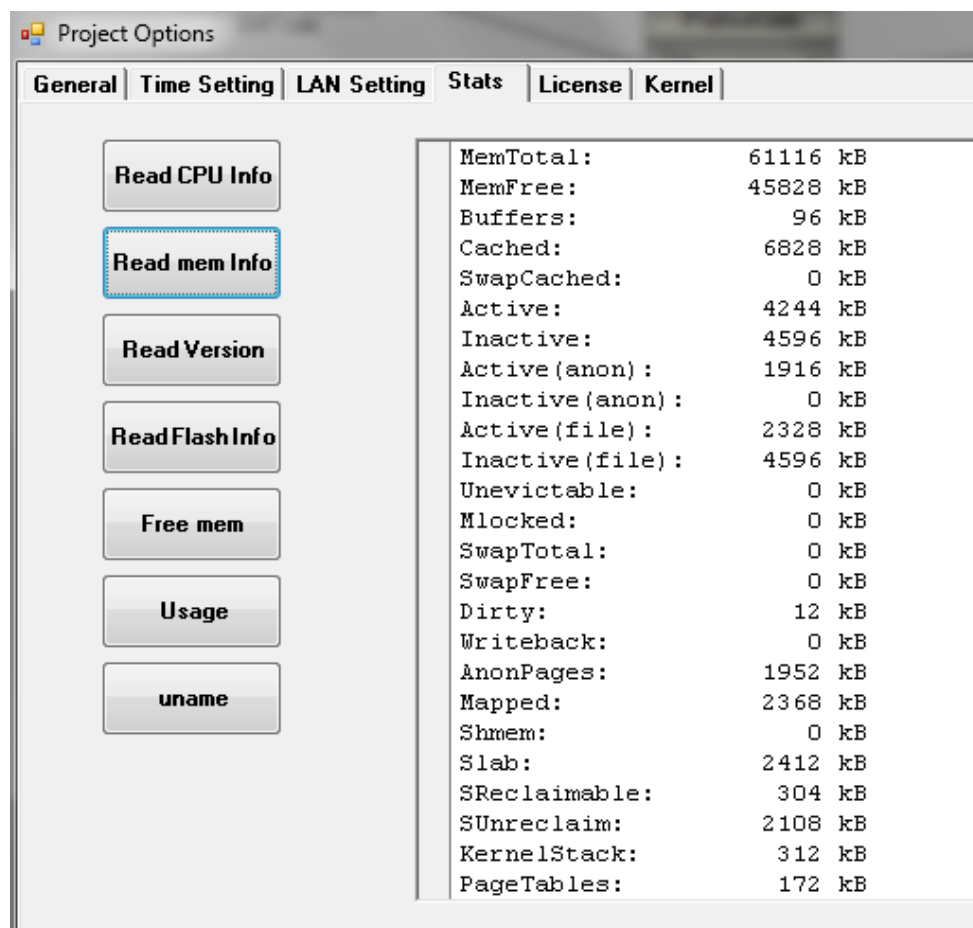
- 1 – Read LAN Settings
- 2 – Change IP address for each LAN port and other settings
- 3 – Write New Settings to Controller.

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:

Project Options

General | Time Setting | LAN Setting | Stats | License | Kernel

Read CPU Info

Read mem Info

Read Version

ReadFlashInfo

Free mem

Usage

uname

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
rootfs	8832	7824	1008	89%	/
/dev/root	8832	7824	1008	89%	/
/dev/ram3	1003	9	943	1%	/dev
/dev/ram0	499	21	453	4%	/var
/dev/mtdblock3	5120	812	4308	16%	/tmp
/dev/mtdblock3	5120	812	4308	16%	/home
/dev/mtdblock3	5120	812	4308	16%	/etc
tmpfs	14748	0	14748	0%	/dev/shm
/dev/ram1	15863	3	15041	0%	/var/ramdisk

Read Free Memory:

Project Options

General | Time Setting | LAN Setting | Stats | License | Kernel

Read CPU Info

Read mem Info

Read Version

ReadFlashInfo

Free mem

Usage

uname

	total	used	free	shared	buffers
Mem:	29500	15744	13756	0	2148
Swap:	0	0	0		
Total:	29500	15744	13756		

Usage: this is equal to top command in Linux .

Project Options

General | Time Setting | LAN Setting | Stats | License | Kernel

Read CPU Info

Read mem Info

Read Version

ReadFlashInfo

Free mem

Usage

uname

```

+ [H+] [JMem: 16116K used, 13384K free, 0K shrd, 2152K buff, 7016K cached
Load average: 0.00 0.00 0.00
+ [7m PID PPID USER STAT VSZ %MEM %CPU COMMAND+ [Om
1002 994 root R 2308 8% 23% top -n 3
867 866 root S 12948 44% 6% ./pbsSLKLX
864 846 root S 12948 44% 0% ./pbsSLKLX
866 864 root S 12948 44% 0% ./pbsSLKLX
868 866 root S 12948 44% 0% ./pbsSLKLX
875 866 root S 12948 44% 0% ./pbsSLKLX
987 935 nobody S 7168 24% 0% /usr/bin/httpd -k start -d /etc/apache
989 935 nobody S 7168 24% 0% /usr/bin/httpd -k start -d /etc/apache
990 935 nobody S 7168 24% 0% /usr/bin/httpd -k start -d /etc/apache
988 935 nobody S 7168 24% 0% /usr/bin/httpd -k start -d /etc/apache
991 935 nobody S 7168 24% 0% /usr/bin/httpd -k start -d /etc/apache
935 1 root S 7144 24% 0% /usr/bin/httpd -k start -d /etc/apache
994 993 root S 2428 8% 0% -bash
846 816 root S 2384 8% 0% /bin/sh /etc/init.d/rcS
816 761 root S 2380 8% 0% /bin/sh /etc/init.d/rcS
761 760 root S 2376 8% 0% /bin/sh /etc/init.d/moxarcs
1000 836 root S 1776 6% 0% /bin/ftp -l
993 836 root S 1376 5% 0% /bin/telnetd
836 1 root S 1356 5% 0% /bin/inetd
985 1 root S 1320 4% 0% dhcpd eth0
1 0 root S 1316 4% 0% init [#]
  
```

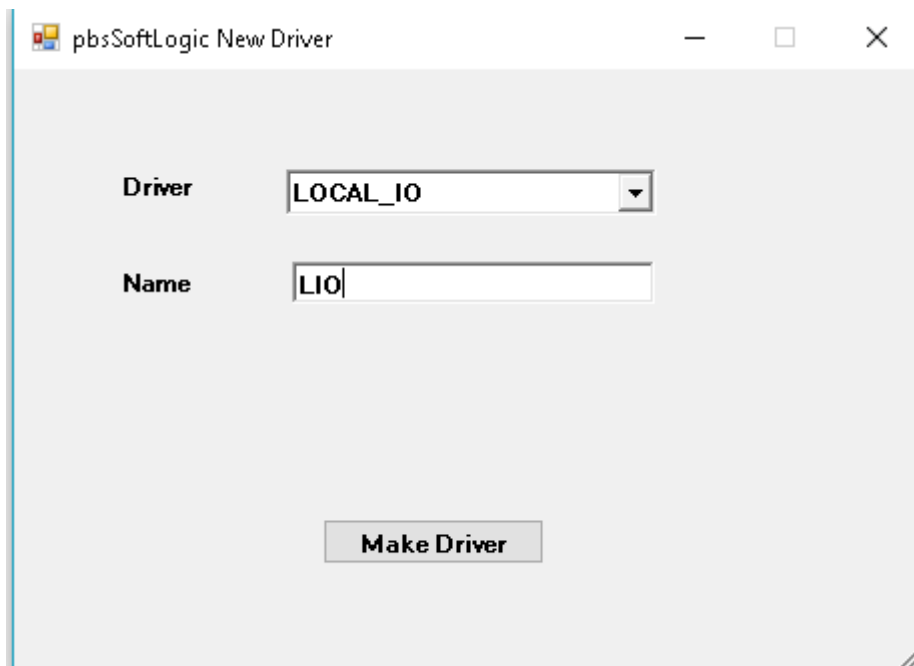
8 – AMS-R3010 RTU Configuration

AMS-R3010 RTU is a common product between Alborz Micro System (AMS) and pbsControl. Hardware is manufactured by AMS and it is powered by pbsSoftLogic. AMS-R3010 RTU supports all pbsSoftLogic protocols. See datasheet and user manual for more detail. (<http://www.alborzmicrosystem.com/ams-rtu3000/>)



Local IO configuration:

For using AMS-R3010 RTU resources, you need to add Local_IO driver to project. Local_IO controls LEDs, Watch Dog, Local Digital Inputs , Analog Inputs , Digital Outputs and temperature Inputs . You can make 3G Modem Off / On by Local _IO Tags in your logic.



When you add Local_IO to your project, pbsSoftLogic will define following configuration file in driver directory.

```
<Tag Name="SYS.Reset" Type="SYS" Init="0" Address="0" />
<Tag Name="SYS.3GModemON" Type="SYS" Init="0" Address="1" />
<Tag Name="SYS.3GModemSignallevel" Type="SYS" Init="0" Address="2" />
<Tag Name="SYS.Temp1" Type="SYS" Init="0" Address="3" />
<Tag Name="SYS.Temp2" Type="SYS" Init="0" Address="4" />
<Tag Name="SYS.CNTTimer" Type="SYS" Init="200" Address="5" />
<Tag Name="SYS.AORange" Type="SYS" Init="1" Address="6" />
<Tag Name="SYS.Buzzer" Type="SYS" Init="0" Address="7" />
<Tag Name="SYS.IOScan" Type="SYS" Init="100" Address="8" />
<Tag Name="SYS.Total1" Type="SYS" Init="0" Address="9" />
<Tag Name="SYS.Total2" Type="SYS" Init="0" Address="10" />
<Tag Name="SYS.Total3" Type="SYS" Init="0" Address="11" />
<Tag Name="SYS.Total4" Type="SYS" Init="0" Address="12" />
<Tag Name="SYS.Total1RST" Type="SYS" Init="0" Address="13" />
<Tag Name="SYS.Total2RST" Type="SYS" Init="0" Address="14" />
<Tag Name="SYS.Total3RST" Type="SYS" Init="0" Address="15" />
<Tag Name="SYS.Total4RST" Type="SYS" Init="0" Address="16" />
<Tag Name="SYS.ChatterFilterCount" Type="SYS" Init="0" Address="17" />
<Tag Name="SYS.ChatterFilterBaseTimeMs" Type="SYS" Init="0" Address="18" />
<Tag Name="SYS.ChatterFilterFreezeTimeMs" Type="SYS" Init="0" Address="19" />
<Tag Name="SYS.3GConnected" Type="SYS" Init="0" Address="20" />
<Tag Name="SYS.3GModemSecIP1" Type="SYS" Init="192" Address="21" />
<Tag Name="SYS.3GModemSecIP2" Type="SYS" Init="168" Address="22" />
<Tag Name="SYS.3GModemSecIP3" Type="SYS" Init="1" Address="23" />
<Tag Name="SYS.3GModemSecIP4" Type="SYS" Init="15" Address="24" />
```

SYS.Reset : when change from 0 to 1 in logic , it will restart RTU

SYS.3GmodemOn : When changed from 0 to 1 , will make 3G Modem On , when changed from 1 to 0 , will make 3G modem off .

SYS.3GModemsignallevel: shows GPRS/3G Signal level in percentage

SYS.3GConnected: When value is 1 , it shows 3G Modem Got valid IP address from Mobile Network .This Signal is checking every Minute .

SYS.3GModemSecIP1 , SYS.3GModemSecIP2 , SYS.3GModemSecIP3, SYS.3GModemSecIP4 :

This is second Virtual IP address for 3G Modem. In above example 3G modem Second IP is 192.168.1.15

SYS.Temp1, SYS.Temp2: shows DS18B20 Digital Temperature sensor value.

SYS.CNTTimer : four DI channels of AMS-R3010 RTU can be used as counter .SYS.CNTTime is base timer for doing counting in IO CPU . it's unit is msec .

SYS.AORange : AMS-R3010 RTU has one AO channel as option .SYS.AORange shows AO Signal Range as following :

SYS.AORange = 1 , 4~20 mA

SYS.AORange = 2 , 0~20 mA

SYS.Buzzer : When changed to 1 in logic will activate RTU buzzer .

SYS.IOScan: this parameter is used in pbsSoftLogic Local IO Driver for AMS-R3010 RTU to read/Write Local I/Os of RTU. Typical value is 100 msec .

SYS.Total1 , 2 , 3 , 4 : Shows Total Pulse counted for DI channels 1 ,2,3 and 4 .

SYS.Total1RST , Total2RST , Total3RST , Total4RST: When changed from 0 to 1 in logic , it will reset SYS.Total1 , SYS.Total2, SYS.Total3 and SYS.Total4 values .

SYS.ChatterFilterCount , SYS.ChatterFilterBaseTimeMS,SYS.ChatterFilterFreezeTimeMS

AMS-R3010 RTU has built in Chatter Filter for DI channels. Chatter filter calculation is handling in IO CPU .But Parameter should pass from Local IO Driver. Typical value for chatter filters parameters:

ChatterFilterCount = 10

ChatterFilterBaseTime = 1000msec

ChatterFilterFreezeTime = 2000 msec

When DI signal is changed for 10 times (ChatterFilterCount) in 1000 msec (ChatterFilterBaseTime) then DI value will freeze for 2000 msec (ChatterFilterFreezeTime)

-When chatter happening bit 7 of DITagx will set to 1

- When there is no chattering bit 7 of DITagx will set to 0

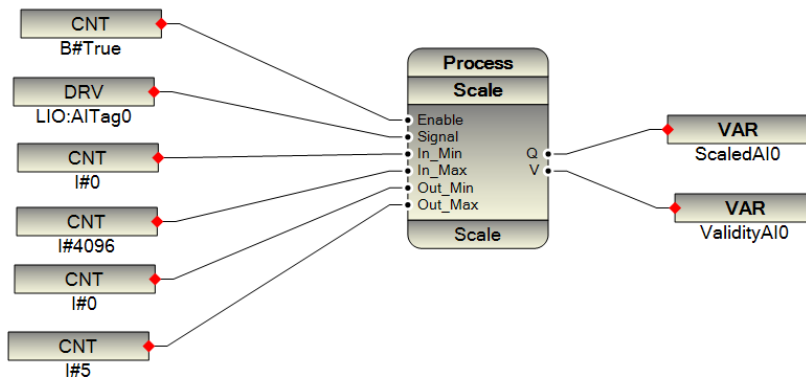
DITagX Value:

- If Signal is 0 and Chatter happened 128
- If Signal is 1 and Chatter happened 129
- If Signal is 0 and Chatter not happened 0
- If Signal is 1 and Chatter not happened 1

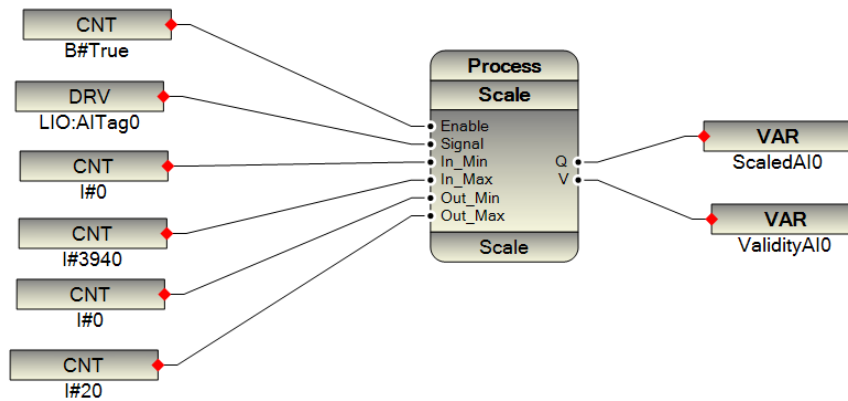
AI0, AI1, AI2, AI3 : Shows value of Analog input Signals

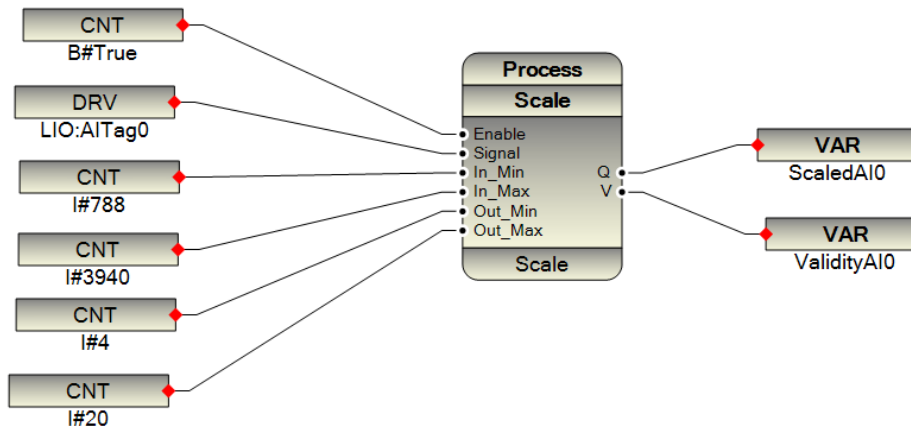
Analog input in AMS-R3010 RTU has 12 bit resolution. You can set AI Range by Switches in back side of RTU.

If Analog Input Signal is set to Voltage mode, then Value of Signal is changing between 0 to 4096. You can use Scale Function block for scaling signal to actual value.



If Analog Input Signal is set to Current Mode then 0 to 20 mA will map to 0 to 3940. For scaling you can use scale Function block for 0~20 mA and 4~20 mA as following samples :





DITag0 to DITag7 : Shows Value of Digital Input Signals with Chatter Filter Status Signal

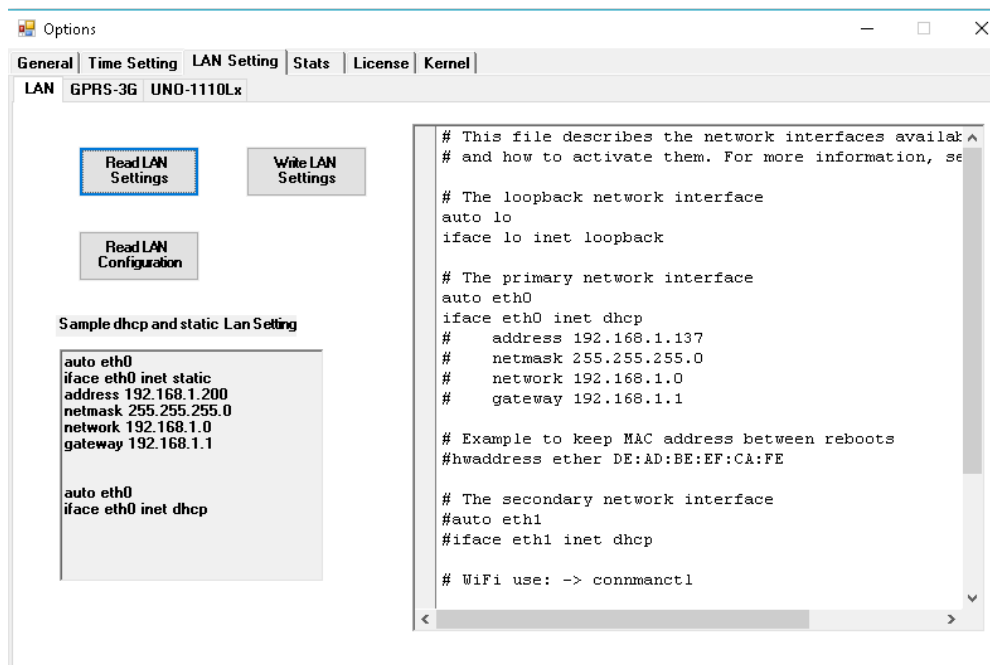
DOTag0 , DOTag1 , DOTag2 and DOTag3 : Read/Write DO signals in logic .

CNTTag0 , CNTTag1 , CNTTag2 , CNTTag3 : Shows Counter Value of four DI channels .Maximum you can count 1 KHz Pulses .

AOTag0: Read/Write Value of Analog Output Signal by logic .

Changing LAN Settings:

For changing LAN Settings you can use pbsSoftLogic. Open project Option page and select LAN Setting Tab:



AMS-R3010RTU eth0 (LAN) port default setting is 192.168.1.137 (like above page) or it is dhcp .

Click on Read LAN setting Button, it will show all LAN setting of AMS-R3010RTU. eth0 is name of RTU LAN port.

Change setting as following for dhcp configuration:

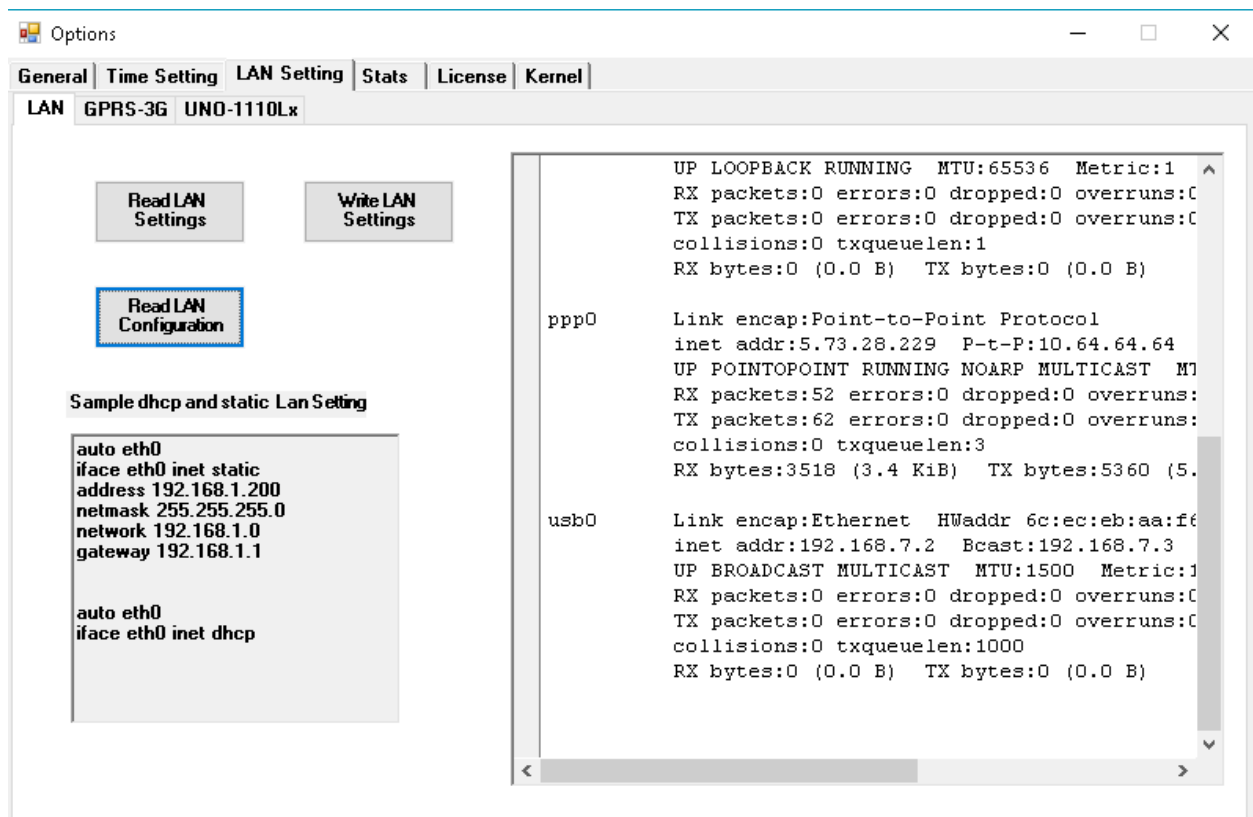
```
auto eth0
iface eth0 inet dhcp
```

Change setting as following for Static IP configuration:

```
auto eth0
iface eth0 inet static
address 192.168.1.200
netmask 255.255.255.0
network 192.168.1.0
gateway 192.168.1.1
```

After changing settings, click on Write LAN setting button. When you restart RTU, it will get new IP setting.

Click on Read LAN Configuration, you can see all LAN configuration of RTU(it is equal to ifconfig command of linux)



If 3G modem is ON and connected to network then you can see ppp0 in list of TCP/IP

Setting GPRS/3G Modem Parameters:

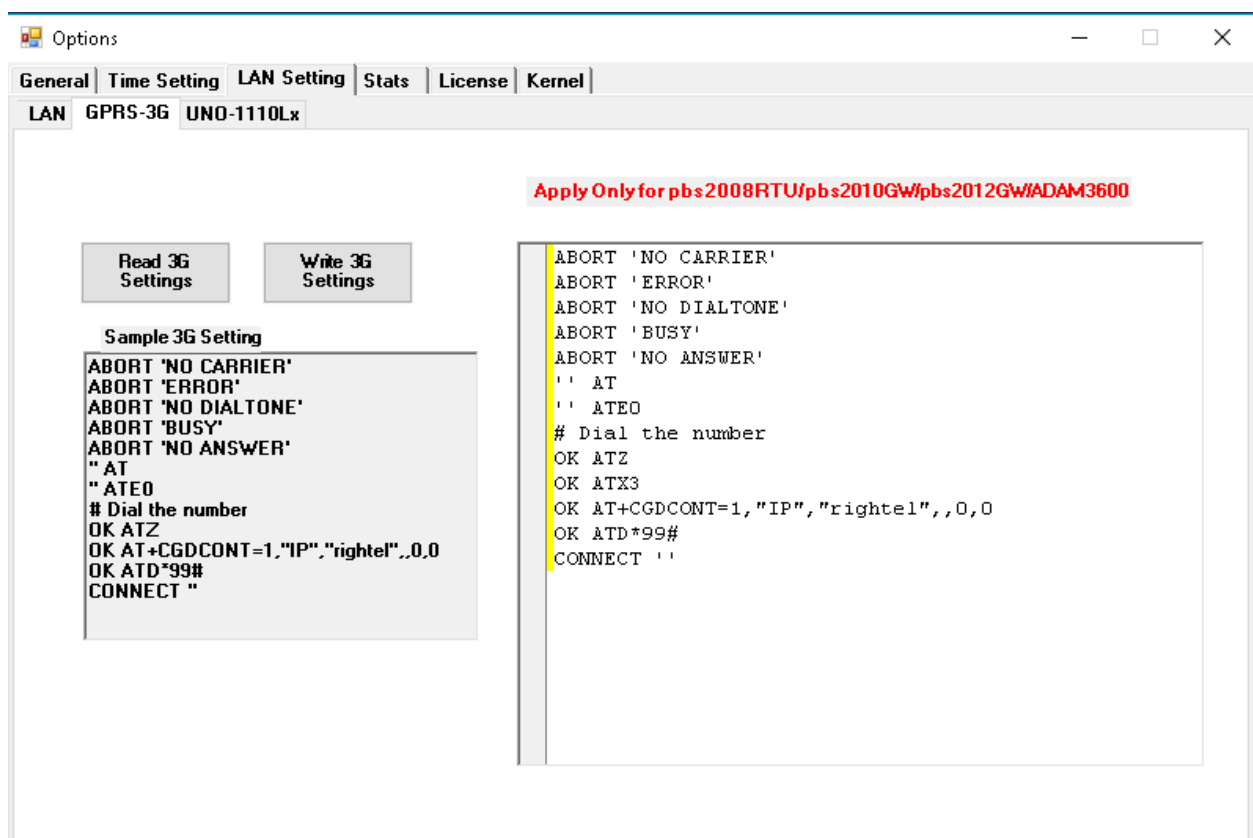
AMS-R3010RTU has a built in 3G Modem for communication with master SCADA . You can make modem Off/On from logic by help of SYS.3GModemOn Local_IO signal .

When you set SYS.3GModemOn to 1 , it will make Modem On . Net LED will start to blink.

When you set SYS.3GModemOn to 0 , it will make modem off .

When Modem is On and connected to network , you can see configuration by clicking on Read Configuration Button . ppp0 is port name for 3G modem .

For setting APN name, select GPRS-3G tab:



In above page change rightel APN name to your service provider name and to be sure that connect command to modem is *99#.

After Settings changed, click on Write 3G Settings and restart RTU.

3G Communication Notes:

- If you only have dynamic IP address for RTUs , then you can use GSPClnet driver to communicate with Master SCADA (You need to have one Fix and valid IP address for SCADA Master)

- By using SQLite Driver and using MS SQL Server automatic data synchronization, you can use dynamic IP address for RTUs and again you need one Fixed and valid IP address for SQL Server in Master control room.
- Other protocols like Modbus TCP, DNP3, IEC104, OPC UA they need static IP address for all RTUs.

Updating AMS-R3010 RTU Runtime Kernel:

- Download latest Runtime kernel from www.pbscontrol.com web site
- Run telnet and connect to RTU . by default User name and password is root ,root.

```
Telnet 192.168.1.137
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-07-10
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:tempPWD]
pbs2008_137 login: root
Password:
Last login: Fri Feb 24 15:17:09 IRST 2017 from kamjooT420 on pts/1
Linux pbs2008_137 4.4.14-bone11 #8 Wed Jul 13 21:32:55 IRDT 2016 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pbs2008_137:~#
```

- go to /home/pbsLX and remove logic.cfg and logic.c11 files by rm command

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# rm logic.cfg
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# rm logic.c11
```

- restart RTU by reboot command

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# reboot
```

- again connect to RTU by telnet and kill pbsSLKLX by pkill command

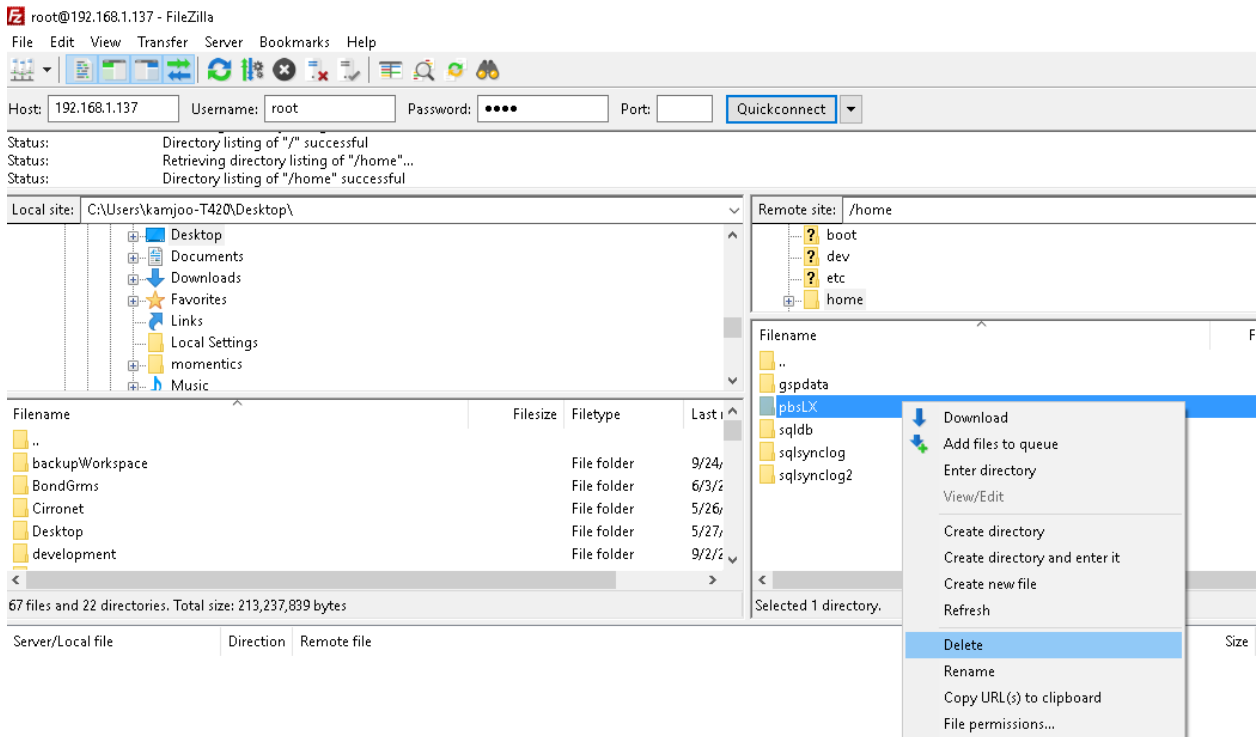
```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pbs2008_137:~# cd /home/pbsLX/
root@pbs2008_137:/home/pbsLX# pkill pbsSLKLX
```

- run ps aux command and check is there OpenOPCUaCoreServer program in program list, If exist kill it by Process ID

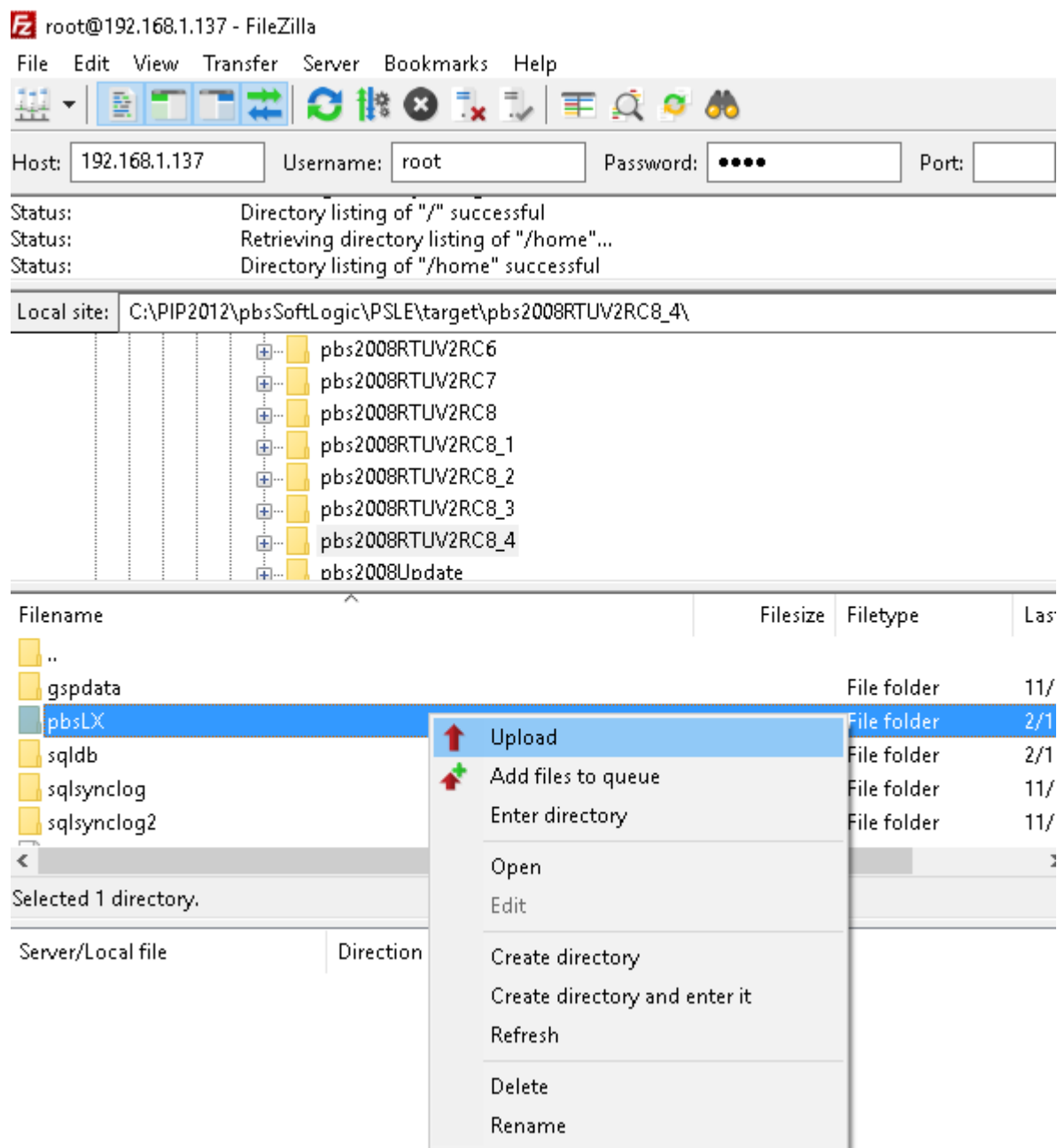
```
root@pbs2008_137:/home/pbsLX# ps aux
```

```
root      454  5.2  1.7 22924  8984 ?        S1   15:18   0:17 ./openOpcUaCoreServer ./Conf
root      483  1.4  0.4  61488  2256 ?        S1   15:18   0:04 ./pbsSLKLX
telnetd   507  0.0  0.2   2124  1372 ?        Ss   15:18   0:00 in.telnetd: kamjooT420
root      508  0.0  0.4   4660  2144 pts/0    Ss   15:18   0:00 login -h kamjooT420 -p
root      510  0.0  0.4   2932  2208 pts/0    S    15:18   0:00 -bash
root      519  0.0  0.2   2632  1340 pts/0    R+   15:23   0:00 ps aux
```

- If OpenOpcUaCoreServer is exist kill it by ID, in above example Process ID is 454
- Check that both pbsSLKLX and OpenOpcUaCoreServer is not in task List.
- run Filezilla and delete /home/pbsLX directory



- copy new pbsLX directory to /home/pbsLX



-
- transfer your logic and configuration
- restart RTU

For Proper setting of Filezilla please refer to chapter 11 . Otherwise you may damage RTU

OpenVPN client on AMS-R3010 RTU

AMS-R3010 RTU Operating system is Standard Debian Linux .We installed before OpenVPN Package on all RTUs , but you can install by following steps on RTU :

- connected AMS-R3010 RTU to Internet (Change Its IP Address to DHCP and connect it to a Internet Router with DHCP Server functionality)
- Connect to RTU by telnet and run “apt-get install openvpn ” command . It will download and install openvpn package from Debian Repository .
- If openvpn package installed before and it is updated you will see following message :

```
root@pbs2008_137:~# apt-get install openvpn
Reading package lists... Done
Building dependency tree
Reading state information... Done
openvpn is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 43 not upgraded.
root@pbs2008_137:~#
```

- Following files will be installed in RTU :

Full Path and File Installed by OpenVPN	Function
/etc/openvpn	Directory containing configuration files
/etc/network/if-up.d/openvpn	Start/stop openvpn when the network goes up/down
/etc/network/if-down.d/openvpn	
/etc/init.d/openvpn	Start/stop script for services
/sbin/openvpn	The binary
/usr/share/doc/openvpn	Documentation files
/usr/share/man/man8/openvpn.8.gz	Manual page
/usr/share/doc/openvpn/examples/sample-config-files	Example configuration files
/usr/share/doc/openvpn/examples/sample-keys	Example keys
/usr/share/doc/openvpn/examples/easy-rsa	easy-rsa—a collection of scripts useful for creating tunnels
/usr/share/doc/openvpn/changelog.Debian.gz	Version history
/usr/share/doc/openvpn/changelog.gz	
/usr/share/openvpn/verify-cn	verify-cn function (revoke command)
/usr/lib/openvpn/openvpn-auth-pam.so	Libraries for PAM-Authentication and chroot mode
/usr/lib/openvpn/openvpn-down-root.so	

Setting symmetric key encryption in both side

You need to provide encryption key for RTU to connect with Server. Same key file should be used in server.

In windows make a new key file with following command:

```
openvpn.exe --pause-exit --verb 3 --genkey --secret "C:\OpenVPN\config\key.txt"
```

Before running above command you should install openvpn on your windows station. Please look at www.openvpn.net for getting openvpn for windows.

By filezilla transfer openvpn sample configuration file from C:\Program Files\OpenVPN\sample-config and generated key file to to AMS-R3010RTU to /home/openvpn directory.

You need to set three parameters in sample configuration file to connect to openvpn server:

- remote : Real IP address of Server .
- ifconfig : Virtual IP address of RTU
- secret : put path of key file

For making connection to server only run openvpn with following command:

```
openvpn --config sample.ovpn
```

If server is configured properly then RTU will connect to server and you can see a new virtual IP address is included in RTU . use ifconfig command to check interface :

```

root@beaglebone:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 98:5d:ad:4a:cb:6a
          inet addr:192.168.1.225  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::9a5d:adff:fe4a:cb6a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23714 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2249 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2701496 (2.5 MiB)  TX bytes:644309 (629.2 KiB)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

tap0      Link encap:Ethernet  HWaddr 3a:b8:39:ae:8a:43
          inet addr:10.3.0.2  Bcast:10.3.0.255  Mask:255.255.255.0
          inet6 addr: fe80::38b8:39ff:feae:8a43/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6355 errors:0 dropped:8 overruns:0 frame:0
          TX packets:933 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:620106 (605.5 KiB)  TX bytes:394649 (385.3 KiB)

usb0      Link encap:Ethernet  HWaddr 98:5d:ad:4a:cb:60
          inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

tap0 is new virtual interface for RTU . tap0 is like a normal eth0 or ppp0 interface and you can use it in pbsSoftLogic for doing protocol configuration .

Setting X509 Certificate

File	Location and purpose
VPN-Server.crt	Signed certificate of the VPN-Server, must be on VPN-Server
VPN-Server.key	Private RSA key of the VPN-Server, must be on VPN-Server
VPN-Server.csr	Certificate signing request of VPN-Server, can be deleted
VPN-client.crt	Signed certificate of the VPN-client, must be on VPN-client
VPN-client.key	Private RSA key of the VPN-client, must be on VPN-client
VPN-client.csr	Certificate Signing request of VPN-Client, can be deleted
ca.crt	CA certificate, must be available on both machines
ca.key	The key to the CA, must be kept only on CA; must be kept very secret
dh2048.pem	The Diffie-Hellman key, must only be available on VPN-Server

For connection of AMS-R3010 RTU to an OpenVPN Master by X509 certificate you need to get following files from OpenVPN Master configurator Team for your RTU:

XXXX.crt signed Certificate

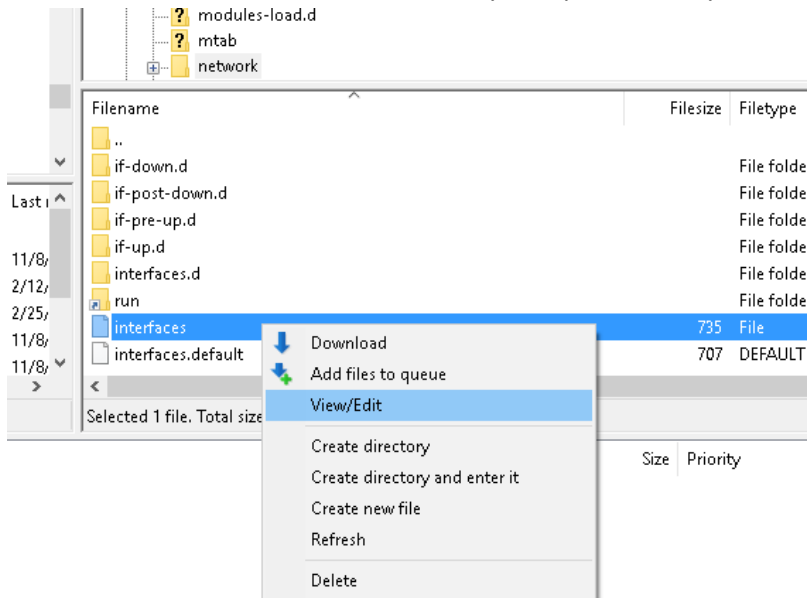
XXXX.Key private RSA key of RTU

Ca.key the key to CA

Make a directory in /home/openvpn and copy above files by filezilla to /home/openvpn directory .

Changing IP address by FileZilla :

- Run filezilla client software and connect to RTU . default user and pass is root ,root
- View/edit /etc/network/interfaces file by notepad ++ utility



- Change IP address for static as following for eth0 port :

```
# The loopback network interface
auto lo
iface lo inet loopback

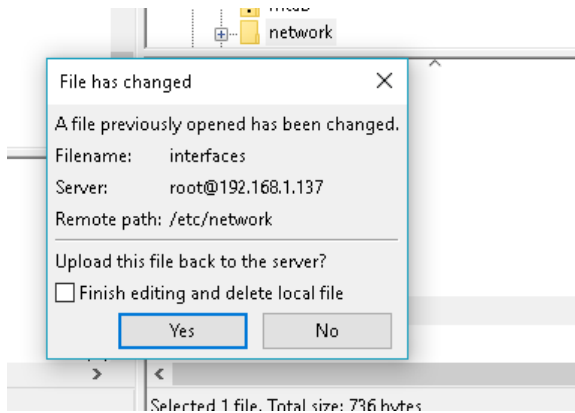
# The primary network interface
#auto eth0
#iface eth0 inet dhcp
auto eth0
iface eth0 inet static
address 192.168.1.137
netmask 255.255.255.0
```

- Save file and exit notepad++
- If you want to change IP to dhcp change it as following :

```
# The primary network interface
auto eth0
iface eth0 inet dhcp

#auto eth0
#iface eth0 inet static
#address 192.168.1.137
#netmask 255.255.255.0
```

- Confirm write to RTU in filezilla software .



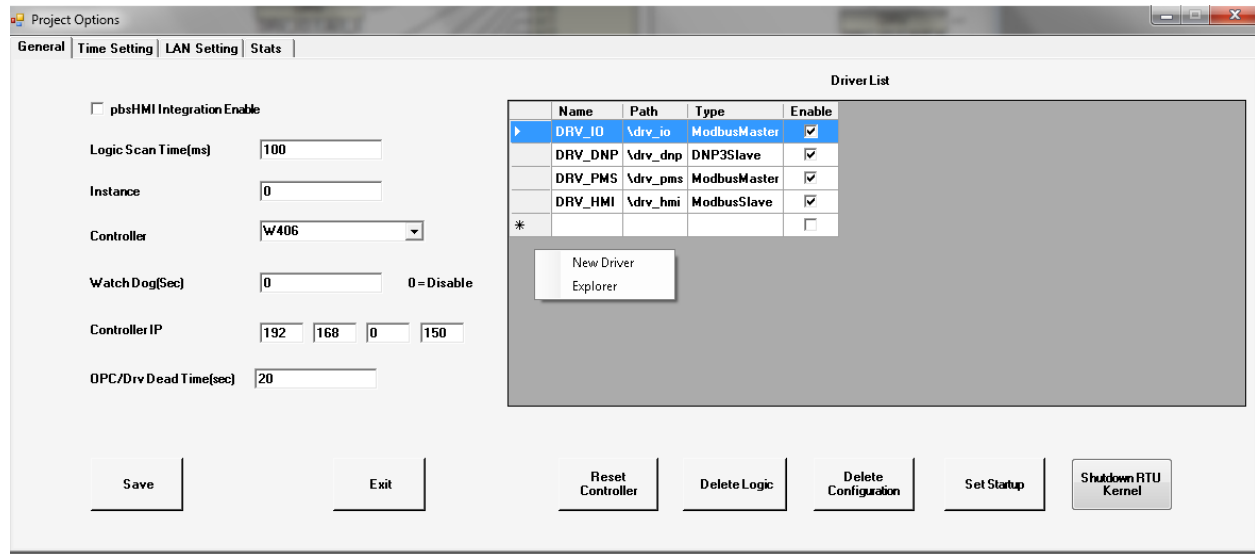
- When using filezilla always check that file transfer is set in Binary . otherwise filezilla will damage linux file in transfer .
- For editing linux file always use Notepad+ , otherwise when you edit files in windows it will damaged file by simple editor like notepad .

9 – 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.

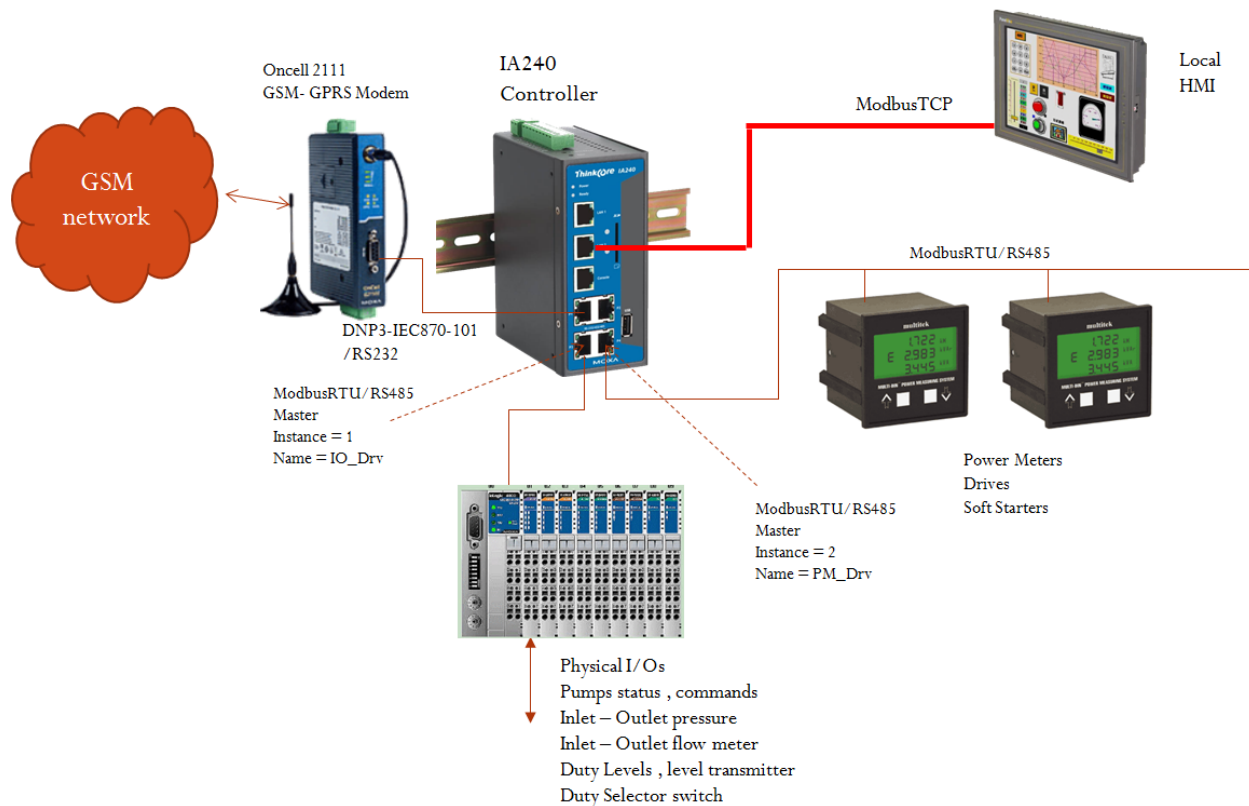


In new driver page, select communication protocol, Type Driver Name and select Driver instance.

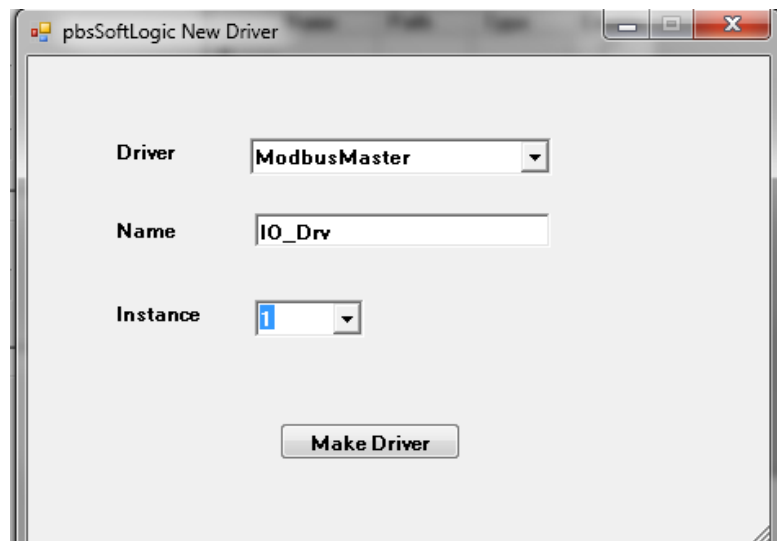
Driver : pbsSoftLogic supports many protocols . Select ModbusMaster for Modbus Master protocol.

Name: Unique Driver Name.

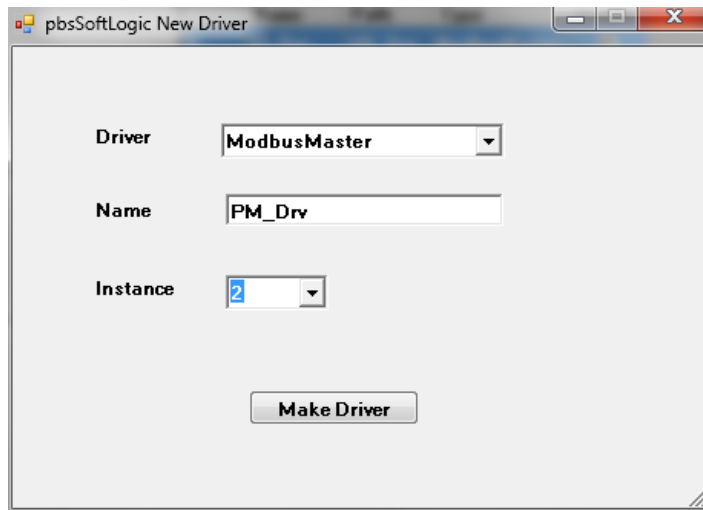
Instance = Instance number for each type of Driver. If you have two Modbus Master Network in project , then you need to define two ModbusMaster Driver with Different name and different instance number. Look at following example, IA240 should connect to I/O Modules and Power monitor network by two different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



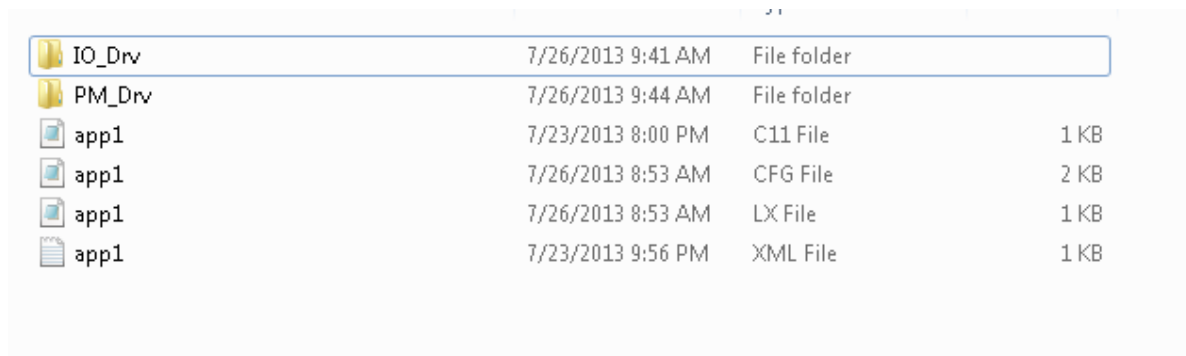
Configuration for Modbus Master Driver for I/O Modules:



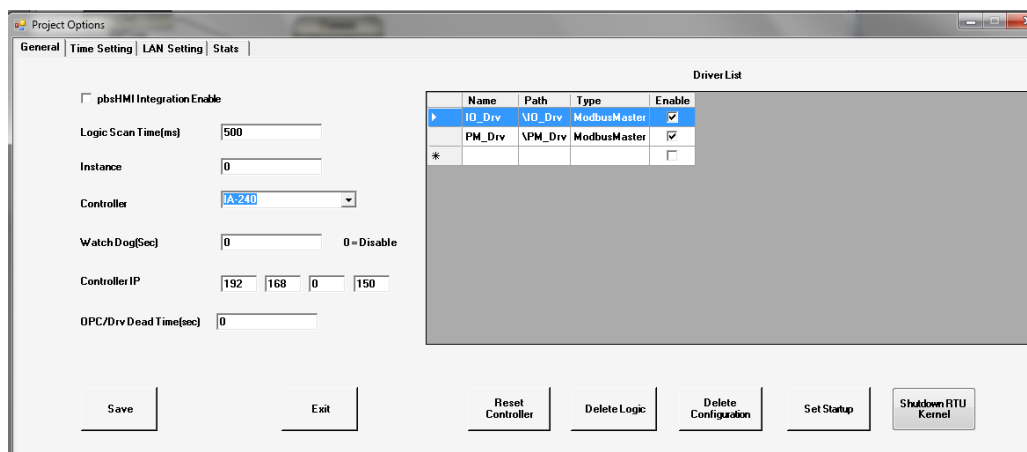
Configuration for Modbus Master Driver for Power Monitor Devices:



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: ModbusTCP Port number. Default Value 502

<Node>

<Name>ContPoll</Name>

<Desc>Continuous polling of Slaves , 1= Enable , 0= Poll Slave by System.Poll Signal</Desc>

<Value>0</Value>

</Node>

ContPoll: This is used for controlling Modbus Slave Device Polling time .

If ContPoll=1 , then RTU will poll slaves permanently

If ContPoll=0 RTU will poll Slaves based on Value of SYS.Poll Signal .

-When SYS.Poll is 1 in your logic , RTU will poll Slaves

-When SYS.Poll is 0 in your logic , RTU not polling Slaves

WakeUpString : Some Modbus Slaves (Specially in Gas Distribution SCADA) start to communicate with RTU if they receive wakeup string . Wakeup string has following format: x,x,x,x,x

X is a decimal number and separated by ,

Example: 255,255,255,255,255

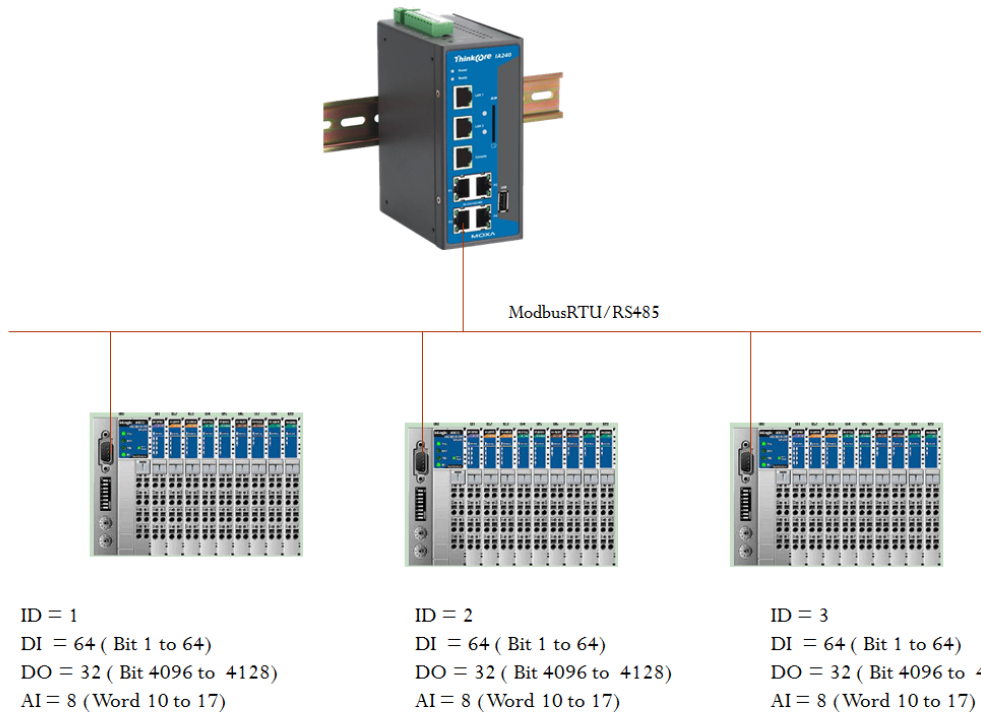
If WakeUpString is Blank , then it is disabled in driver.

WakeUpString and ContPoll should use with each other . for using WakeUpString you should set ContPoll to 0 and control Slave Polling in your logic . Every time SYS.Poll is changed from 0 to 1 , WakeUpString is send to Slave device .

There is a specific Function Block in Process Group (Scheduler) . you can use this FB to control Slave Device Polling .

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:



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 of Block.

Type: Block Type

BI = DI: Digital Input = Modbus Input status (Read from Slave) Send FC = 2

BO= DO: Digital Output = Modbus Coil(Write to slave) , Send FC = 5

BOS=DOS: Digital Output Status = Modbus Coils Status (Read from Slave) Send FC = 1

AI: Analog Input = Modbus input Register (2 bytes , Signed) (Read from Slave) Send FC = 4

AO: Analog Output = Modbus Holding Register (2 bytes , Signed) (Write to slave) Send FC = 6

AOS: Analog Output Status = Modbus Holding Register status (2 bytes , Signed) (Read from Slave) Send FC = 3

FI : Float Input . use same Address Space of AI . Each FI tag is getting two AI register . (4 Bytes)
(Read from Slave) Send FC = 4

SFI : Swap Float Input . use same Address Space of AI . Each SFI tag is getting two AI register
(Read from Slave) Send FC = 4

LI : Long Input . use same Address Space of AI . Each LI tag is getting two AI register . (4 bytes
, Signed) (Read from Slave) Send FC = 4

SLI : Swap Long Input . use same Address Space of AI . Each SLI tag is getting two AI register .
(Read from Slave) Send FC = 4

FO : Float Output . use same Address Space of AO . Each FO tag is getting two AO holding
register . (Write to slave) Send FC = 16

SFO : Swap Float Output . use same Address Space of AO . Each SFO tag is getting two AO
holding register . (Write to slave) Send FC = 16

FOS : Float Output Status . use same Address Space of AO . Each FOS tag is getting two AO
holding register . (Read from Slave) Send FC = 3

SFOS : Swap Float Output Status . use same Address Space of AO . Each SFOS tag is getting two
AO holding register . (Read from Slave) Send FC = 3

LO : Long Output . use same Address Space of AO . Each LO tag is getting two AO holding
register . (Write to slave) Send FC = 16

SLO : Swap Long Output . use same Address Space of AO . Each SLO tag is getting two AO
holding register . (Write to slave) Send FC = 16

LOS : Long Output Status . use same Address Space of AO . Each LOS tag is getting two AO holding register . (Read from Slave) Send FC = 3

SLOS : Swap Lang Output Status . use same Address Space of AO . Each SLOS tag is getting two AO holding register . (Read from Slave) Send FC = 3

SYS: Internal for pbsSoftLogic . Can be used for reading status of communication. It has 4 Signal:
Online : Shows Slave is Working properly and answering to RTU Request

SendNum : Number of Send request from RTU to Slave . It is changing from 0 to 32000

RecNum: Number of Recived messages from Slave . It is changed between 0 to 32000

Poll : When ConPoll is equal to 0 , then Driver is polling Slaves based on value of Poll Signal .

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)

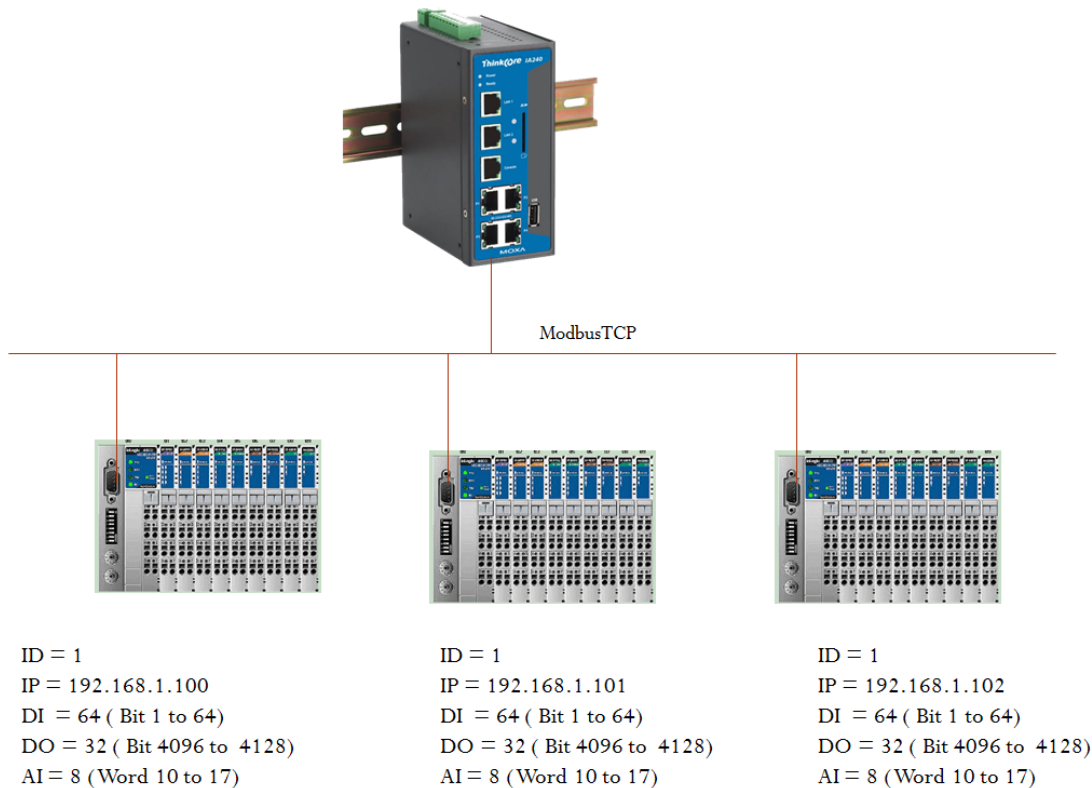
Count = Channel Count , for FI , LI , SFI , SLI , FO,LO,SFO,SLO,... pbsSoftLogic Kernel will automatic read double register or Holding register .

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 :



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:

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

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

$200(DI3)+50+200(AI3)+50 = 3000 \text{ msec} = 3 \text{ sec} .$ (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 . FBEditor used this file for accessing tags.

Modbus Tag has following format In ModbusTags.xml file :

```

<?xml version="1.0"?>
<OPCSrvTags>
  <Version>1.0.0</Version>
  <Tag BlockName="DI1" Address="0" Name="P1_Auto" />
  <Tag BlockName="DI1" Address="1" Name="P1_Run" />
  <Tag BlockName="DI1" Address="2" Name="P1_Trip" />
  <Tag BlockName="DI1" Address="3" Name="P2_Auto" />
  <Tag BlockName="DI1" Address="4" Name="P2_Run" />
  <Tag BlockName="DI1" Address="5" Name="P2_Trip" />
  <Tag BlockName="DI1" Address="6" Name="SP_Auto" />
  <Tag BlockName="DI1" Address="7" Name="SP_Run" />
  <Tag BlockName="DI1" Address="8" Name="SP_Trip" />

  <Tag BlockName="DO1" Address="4096" Name="P1_StartCMD" />
  <Tag BlockName="DO1" Address="4097" Name="P2_StartCMD" />
  <Tag BlockName="DO1" Address="4098" Name="SP_StartCMD" />
  <Tag BlockName="DO1" Address="4099" Name="EF_StartCMD" />
  <Tag BlockName="DO1" Address="4100" Name="RTU_Healthy_LAMP" />
  <Tag BlockName="DO1" Address="4101" Name="DO Spare4101" />
  <Tag BlockName="DO1" Address="4102" Name="TakeOverFromRTU" />
  <Tag BlockName="DO1" Address="4103" Name="DO Spare4103" />
  <Tag BlockName="DO1" Address="4104" Name="FDOSPump_StartCMD" />
  <Tag BlockName="DO1" Address="4105" Name="DO Spare4105" />
  <Tag BlockName="DO1" Address="4106" Name="DO Spare4106" />
  <Tag BlockName="DO1" Address="4107" Name="DO Spare4107" />
  <Tag BlockName="DO1" Address="4108" Name="DO Spare4108" />
  <Tag BlockName="DO1" Address="4109" Name="DO Spare4109" />
  <Tag BlockName="DO1" Address="4110" Name="DO Spare4110" />
  <Tag BlockName="DO1" Address="4111" Name="DO Spare4111" />

```

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 . If Block Is starting by 4096 , then you need to start Tag Address from 4096 and add one by one all tags . Please define all Tags for a block .

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

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

For Diag Block you need to define following tags :

```

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

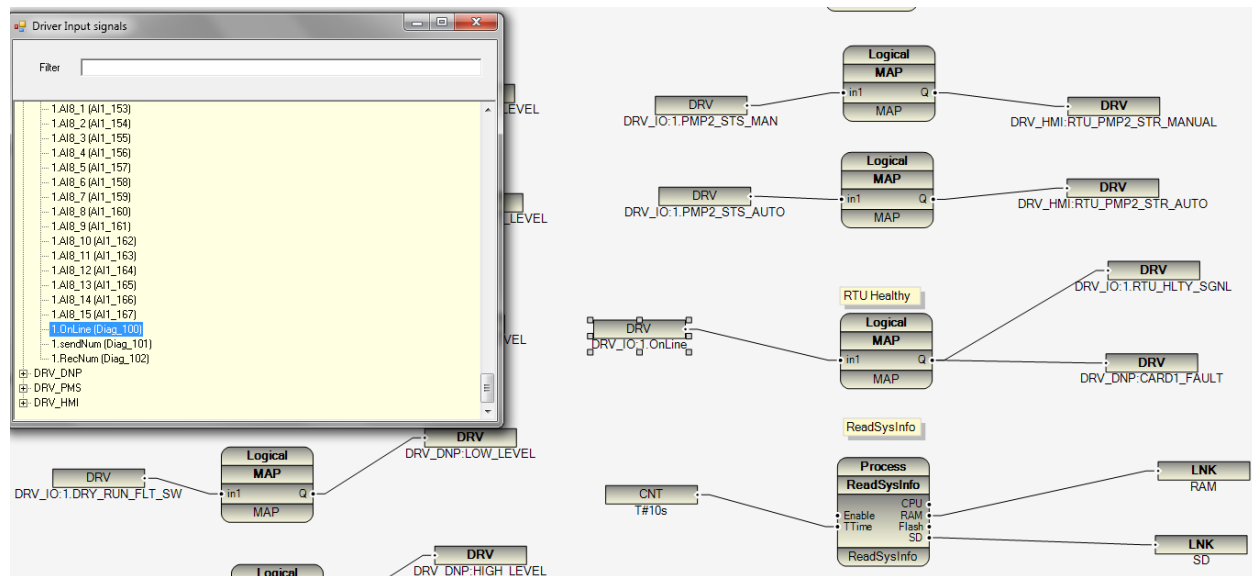
```

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 Devices for each instance: 32

Note : it is recommended that first using Modbus Tester utility Like Modscan (<https://www.win-tech.com/html/modbus1.htm>) to find detail of Modbus Tags address inside Slave devices (Power meter , IO Modules , SoftStarter , Flow Computers , ...) and then configure pbsSoftLogic Modbus Master Blocks parameters .

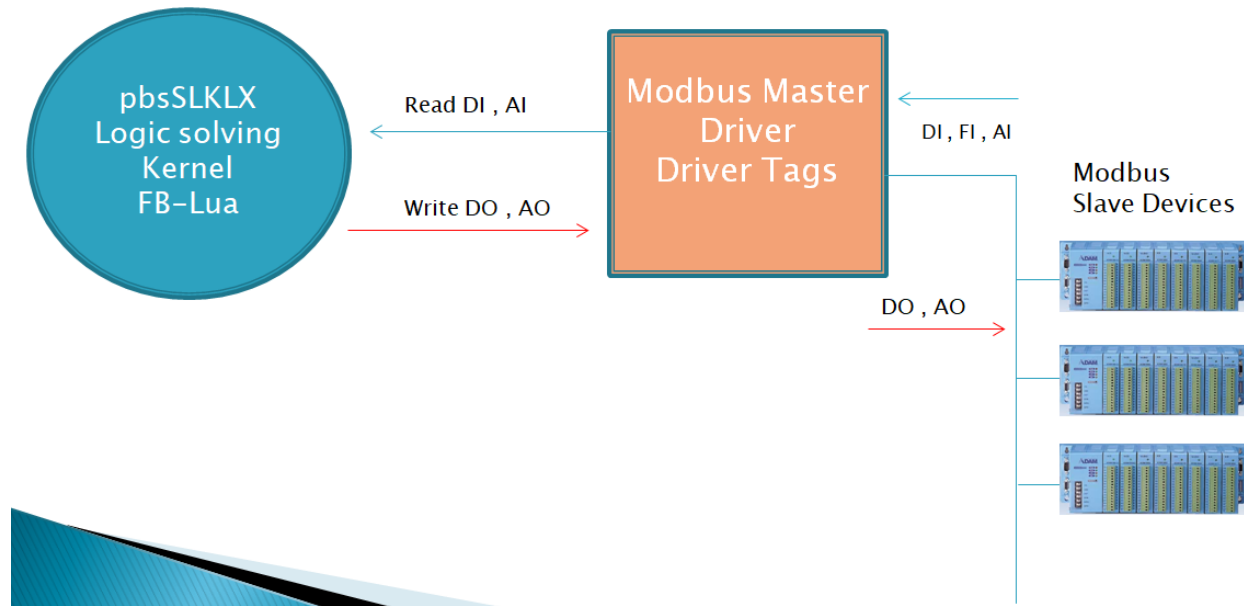
Normally when you read /write a Modbus Block by Modscan , you should reduce One address from Block start Address in Modscan and set in pbsSoftLogic Modbus Block Definition .

Suppose in Modscan you read an Input Register Block from Address 101 and read 32 register.

In pbsSoftLogic you should set Block Start Address to 100 and Count to 32.

In following figure you can see Modbus Master Driver and its relation with pbsSLKLX kernel.

- Each Driver has it Thread and Tags
- Logic is reading Inputs and Writing Outputs at each cycle
- Driver is independent of other module , communicates with Third Party and update Driver Tags



Modbus Master Driver Loading Steps :

- pbsSLKLX is reading logic.cfg file and find there is Modbus Master Driver in Configuration
- pbsSLKLX Load Modbus Master Driver from /home/pbsLX/drvlib/mmix/libpbsModbusMLx.so
- pbsSLKLX will pass Driver Parameters and Tags based on logic.cfg file and Initialize driver
- modbus Master Driver (libpbsModbusMLx.so) will make new CPU thread and start to read write Modbus Blocks from Modbus Slaves and update in ternal Driver Tags .
- pbsSLKLX has access to Modbus Driver Internal Tags by unified API Interface

10 – Modbus Slave Configuration

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

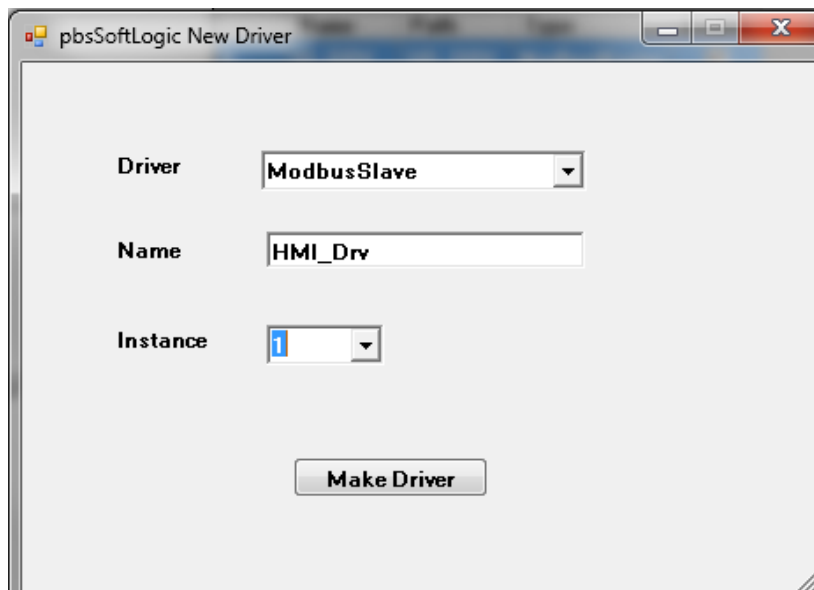
You can run Modbus master and slave on the same Controller in the same time but they should have separate resource. For example COM Port 1 can be Modbus Master and COM Port 2 Modbus Slave.

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.

Totally you can define 1024 Tags for each Modbus Slave Instances.

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

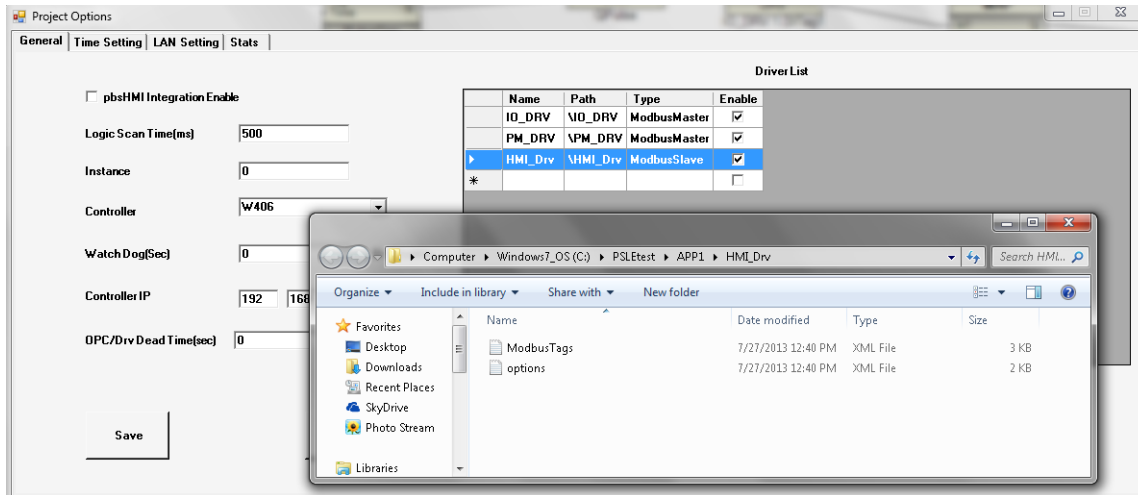
Select ModbusSlave Driver and fill other fields.



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 ModbusRTU

<Node>

<Name>BaudRate</Name>

<Desc>9600,19200,36400,52700,115200</Desc>

<Value>9600</Value>

</Node>

BaudRate : Modbus RTU Baudrate 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 many 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.

```

ModbusTags - Notepad
File Edit Format View Help
<?xml version="1.0"?>
<OPCSrvTags>
  <version>1.0.0</version>
  <Tag Name="DITag1" Type="DI" Init="0" Address="1" Log="0" />
  <Tag Name="DITag2" Type="DI" Init="0" Address="2" Log="0" />
  <Tag Name="DITag3" Type="DI" Init="0" Address="3" Log="0" />
  <Tag Name="DITag4" Type="DI" Init="0" Address="4" Log="0" />
  <Tag Name="DITag5" Type="DI" Init="0" Address="5" Log="0" />
  <Tag Name="DITag6" Type="DI" Init="0" Address="6" Log="0" />
  <Tag Name="DITag7" Type="DI" Init="0" Address="7" Log="0" />
  <Tag Name="DITag8" Type="DI" Init="0" Address="8" Log="0" />

  <Tag Name="AITag1" Type="AI" Init="0" Address="1" Log="0" />
  <Tag Name="AITag2" Type="AI" Init="0" Address="2" Log="0" />
  <Tag Name="AITag3" Type="AI" Init="0" Address="3" Log="0" />
  <Tag Name="AITag4" Type="AI" Init="0" Address="4" Log="0" />
  <Tag Name="AITag5" Type="AI" Init="0" Address="5" Log="0" />
  <Tag Name="AITag6" Type="AI" Init="0" Address="6" Log="0" />
  <Tag Name="AITag7" Type="AI" Init="0" Address="7" Log="0" />
  <Tag Name="AITag8" Type="AI" Init="0" Address="8" Log="0" />

  <Tag Name="DOTag1" Type="DO" Init="0" Address="1" Log="0" />
  <Tag Name="DOTag2" Type="DO" Init="0" Address="2" Log="0" />
  <Tag Name="DOTag3" Type="DO" Init="0" Address="3" Log="0" />
  <Tag Name="DOTag4" Type="DO" Init="0" Address="4" Log="0" />
  <Tag Name="DOTag5" Type="DO" Init="0" Address="5" Log="0" />
  <Tag Name="DOTag6" Type="DO" Init="0" Address="6" Log="0" />
  <Tag Name="DOTag7" Type="DO" Init="0" Address="7" Log="0" />
  <Tag Name="DOTag8" Type="DO" Init="0" Address="8" Log="0" />

  <Tag Name="AOTag1" Type="AO" Init="0" Address="1" Log="0" />
  <Tag Name="AOTag2" Type="AO" Init="0" Address="2" Log="0" />
  <Tag Name="AOTag3" Type="AO" Init="0" Address="3" Log="0" />
  <Tag Name="AOTag4" Type="AO" Init="0" Address="4" Log="0" />
  <Tag Name="AOTag5" Type="AO" Init="0" Address="5" Log="0" />
  <Tag Name="AOTag6" Type="AO" Init="0" Address="6" Log="0" />
  <Tag Name="AOTag7" Type="AO" Init="0" Address="7" Log="0" />
  <Tag Name="AOTag8" Type="AO" Init="0" Address="8" Log="0" />
</OPCSrvTags>

```

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 .

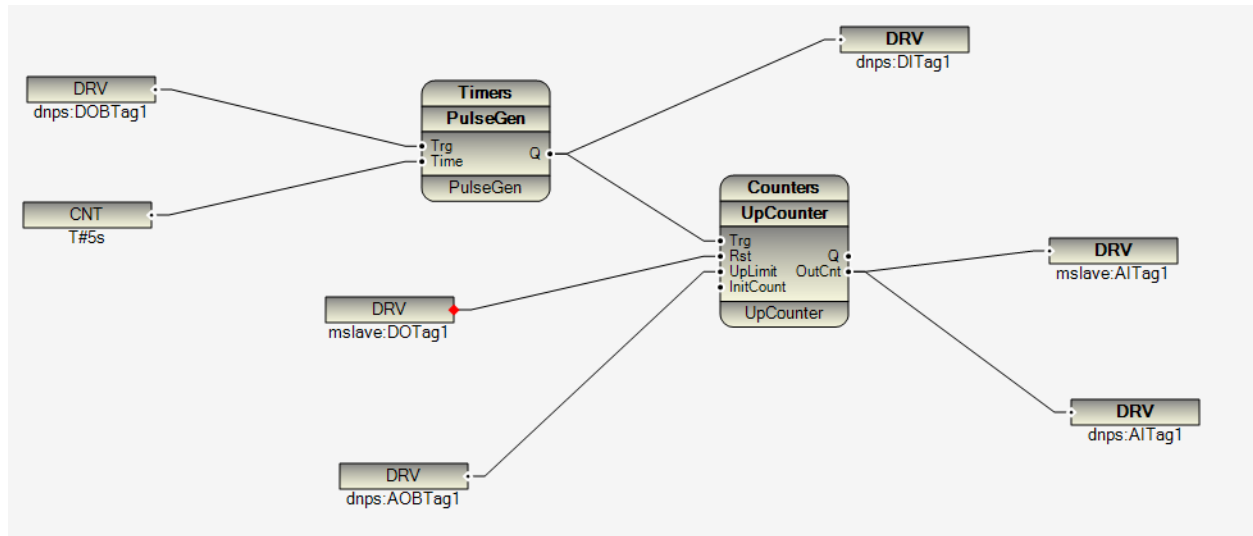
Modbus Slave Driver operation:

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:AI Tag1 is an input signal to modbus master (Linked to FB output ports)

There is a new SYS type signal in Modbus Slave Tags which is automatically define by pbsSoftLogic .

```
<Tag Name="MasterOnline" Type="SYS" Init="0" Address="0" Log="0" />
<Tag Name="DI Tag0" Type="DI" Init="0" Address="0" Log="0" />
<Tag Name="DI Tag1" Type="DI" Init="0" Address="1" Log="0" />
```

MasterOnLine Signal Shows Master is connected to Slave Driver or not. If Master is sending proper Message to Slave Driver and getting Answer from Driver , MasterOnLine Will set to 1 , otherwise it will set to 0 .

Important Point about **PhysicalLayerScanTime**

In Some projects that DCS is operating as Modbus Master and pbsSoftLogic is working as Modbus Slave , Because DCS is writing all signals mostly as Holding Register into Slave , Modbus Frame length from DCS side is not small so you need to increase PhysicalLayerScanTime to accept DCS Modbus Frame . Otherwise RTU is not communication with DCS .

11 – DNP3 Slave Configuration

pbsSoftLogic supports DNP3 slave driver. Please refer to www.dnp.org web site for detail information about DNP3 protocol.

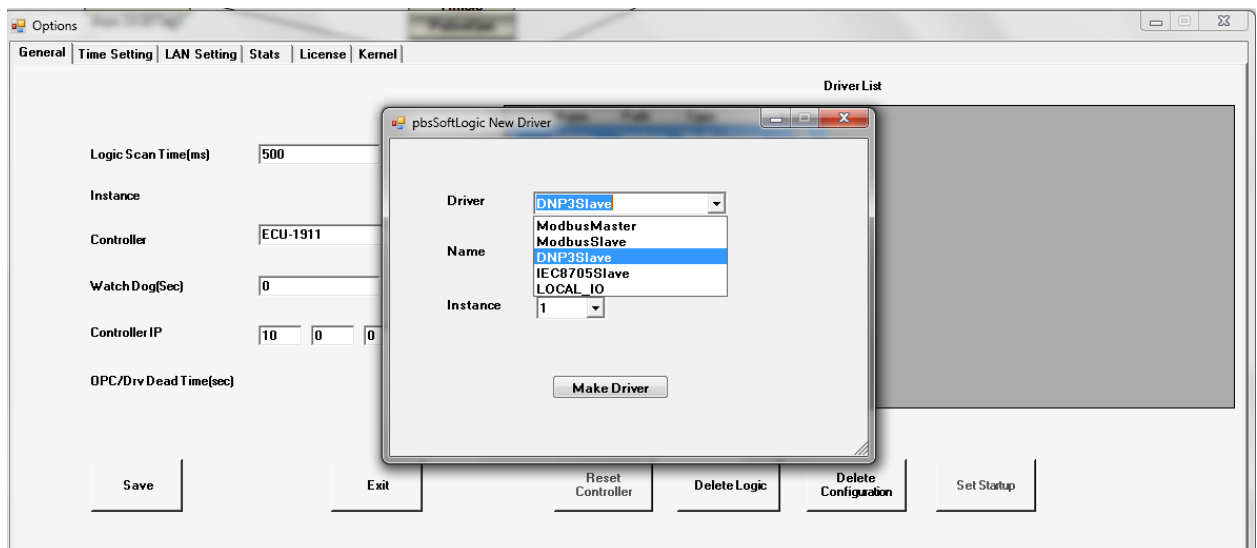
You can define up to 4 dnp3 slave instances for a controller. Each DNP3 slave instance can be connected to separate DNP3 master SCADA.

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.

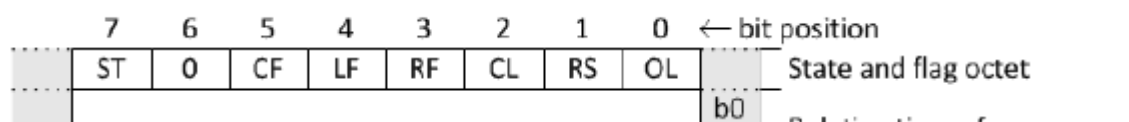
Address: DNP3 tag address. AI and FI are using same address range.

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" />
```

Tag Flag: Based on DNP3 Standard, DNP3 Data Types has Flag Status with following definition.

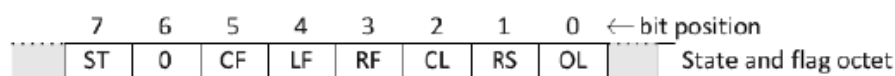


Name	Functional description
ONLINE	<p>For input data objects:</p> <p>If clear, the point is inactive or disabled (for example: powered-down, faulty, etc.) and unable to obtain field data. The flag may optionally be cleared by a non-originating device if communications to the originating device fail. In this case the COMM_LOST flag shall also be set.</p> <p>For output status objects:</p> <p>If clear, the output point is inactive, unavailable, out-of-service, not installed, or operating in local mode. The point may not be observable or may be not controllable. Commands sent to the point may fail.</p> <p>When an output point is in local mode, it shall clear this flag.</p>
RESTART	<p>The RESTART flag indicates that the data has not been updated from the field since device reset.</p> <p>Originating devices shall set this bit immediately upon restarting and keep the bit set until they have an updated value in their database.</p> <p>Non-originating devices shall set this bit immediately upon restart and keep the bit set until it is overwritten by collecting data from a reporting device.</p> <p>For input data objects: If set, the object is in the initialization state, having a value that has never been updated from the field since restart. The bit is cleared when the object is first updated. In an originating device, this is when the field value is first acquired. In a non-originating device, the bit remains set until it is overwritten by collecting data from a reporting device and that data does not have the RESTART flag set.</p> <p>For output status objects:</p> <p>The RESTART flag shall only be set while a device is restarting. In an originating device, the flag shall be cleared after the device is available to accept commands, irrespective of whether or not an output value (control) has been sent to the output object. In a non-originating device, the bit remains set until it is overwritten by collecting output status information from a reporting device and that data does not have the RESTART flag set.</p>
COMM_LOST	<p>COMM_LOST indicates that there is a communication failure in the path between the device where the data originates and the reporting device. This flag indicates that the value reported for the object may be stale.</p> <p>If set, the data value reported shall be the last value available from the originating device before communications were lost.</p> <p>An originating device never sets this flag. A non-originating device sets this flag if it loses communication with the adjacent downstream device; otherwise it propagates the state of COMM_LOST flag as received from the downstream device. Once set, this flag may only be cleared when data for this point is received from the adjacent downstream device and the COMM_LOST flag received with that data is cleared.</p>

REMOTE_FORCED	<p>If set, the data value is overridden in a downstream reporting device.</p> <p>Only a non-originating device may set this flag. The flag is set when an overridden value is received. The REMOTE_FORCED flag shall be set in an object if either the REMOTE_FORCED or LOCAL_FORCED flags (see below) are set in an object received from a downstream device.</p> <p>An originating device may never set this bit.</p> <div data-bbox="483 552 1333 657"> <pre> graph RL A[Reported value = X with REMOTE_FORCED set] --> B[Non-originating Device] C[Received value = X with LOCAL_FORCED or REMOTE_FORCED set] --> B </pre> </div> <p>See flag description 11.6.1.1, NOTE 3.</p>
LOCAL_FORCED	<p>If set, the data value is overridden by the device that reports this flag as set. This may be due to the device operating in a diagnostic or temporary mode or due to human intervention.</p> <div data-bbox="483 814 1243 919"> <pre> graph RL A[Reported value = Y with LOCAL_FORCED set] --> B[Device in which X value is overridden by Y] C[Input value = X] --> B </pre> </div> <p>If the value is forced in a non-originating device and overridden in a downstream device, then the non-originating device shall set both REMOTE_FORCED and LOCAL_FORCED flags.</p> <div data-bbox="483 1031 1333 1136"> <pre> graph RL A[Reported value = Y with LOCAL_FORCED and REMOTE_FORCED set] --> B[Device in which X value is overridden by Y] C[Received value = X with LOCAL_FORCED or REMOTE_FORCED set] --> B </pre> </div> <p>See flag description 11.6.1.1, NOTE 3.</p>

CHATTER_FILTER	<p>Only applicable to single-bit binary input and double-bit binary input object groups.</p> <p>If set, the binary data value is presently changing between states at a sufficiently high enough rate to activate a chatter filter. The binary data value reported does not necessarily represent the actual state because the chatter filter may clamp the reported value to a single state during the time it is active.</p> <p>The purpose of the chatter filter is to suppress event reporting for binary and double-bit binary inputs that are experiencing a rapid series of state changes. The determination of what constitutes “chattering” is device-dependent.</p> <p>While a binary input is chattering, the originating device shall set the CHATTER_FILTER flag. When the chattering input again becomes stable, the originating device shall clear the flag and report the current state of the input. The CHATTER_FILTER bit indicates that the binary input point has been filtered in order to remove unneeded transitions in the state of the point.</p> <p>Events are generated when the CHATTER_FILTER flag is set and cleared.</p>
ROLLOVER	<p>Only applicable to counter object groups.</p> <p>This flag is obsolete and should not be set in new designs. Information is presented here for historical reasons.</p> <p>There is no mechanism within DNP3 for the outstation to report the value at which counter rollover occurs (i.e., the maximum possible counter value). Hence outstations shall not set the ROLLOVER flag and master devices shall ignore the ROLLOVER flag. If polled data reporting is used, the master is responsible for polling counter data frequently enough to detect rollover.</p>
OVER_RANGE	<p>Only applicable to analog input and analog output object groups.</p> <p>If set, the data object's true value exceeds the valid measurement range of the object.</p> <p>See flag description 11.6.1.1, NOTE 4, for more details.</p>

State Flag for Digital signals:

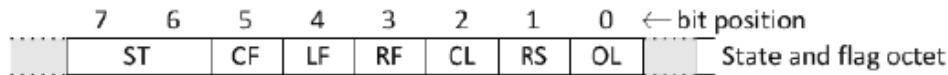


A.2.2.2.2 Formal structure

BSTR8: Flag Octet

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	CHATTER_FILTER
Bit 6:	Reserved, always 0
Bit 7:	STATE—Has a value of 0 or 1, representing the state of the physical or logical input.

State Tag for Double Bit signals :



A.4.2.2.2 Formal structure

BSTR6: Flags

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	CHATTER_FILTER

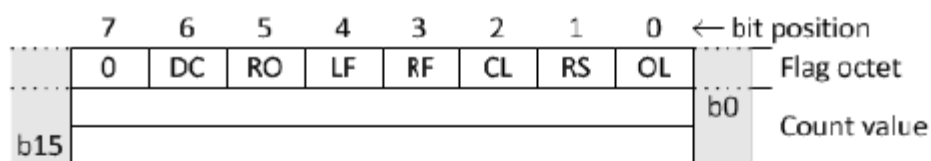
UINT2: State.

This integer contains the state of the double-bit binary input as defined for a Double-bit Binary Input Point Type in 11.9.6. These values represent states INTERMEDIATE, DETERMINED_OFF, DETERMINED_ON, and INDETERMINATE.

State Flag for DO Signals :

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	Reserved, always 0
Bit 6:	Reserved, always 0
Bit 7:	STATE—Value is 1 or 0, where 1 indicates that the output signal is active and 0 indicates that the output signal is not active. Where the output object does not have a meaningful output state, the STATE flag shall be 0.

State Flag for Counters :

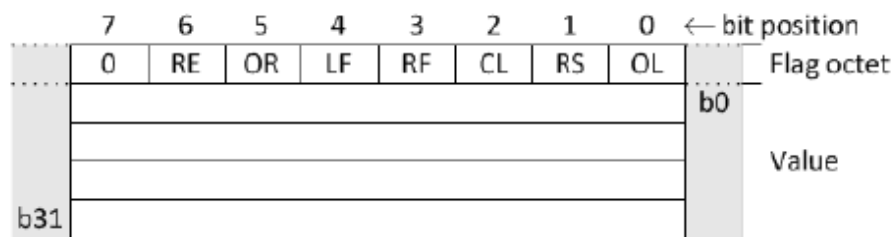


A.10.2.2.2 Formal structure

BSTR8: Flag Octet

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	ROLLOVER
Bit 6:	DISCONTINUITY
Bit 7:	Reserved, always 0.

State Flag for AI :

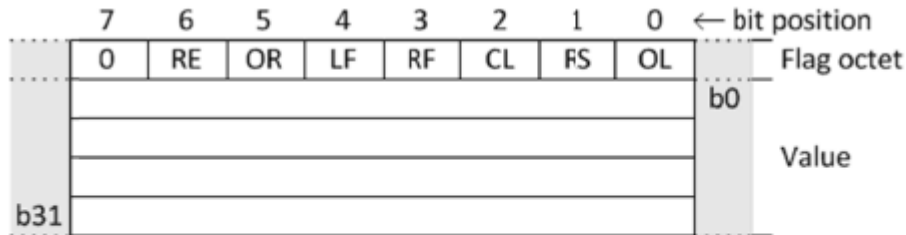


A.14.1.2.2 Formal structure

BSTR8: Flag Octet

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	OVER_RANGE
Bit 6:	REFERENCE_ERR
Bit 7:	Reserved, always 0.

State Flag for AO :



A.19.1.2.2 Formal structure

BSTR8: Flag Octet

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	OVER_RANGE
Bit 6:	REFERENCE_ERR
Bit 7:	Reserved, always 0.

You can set DNP3 Flags to DNP3 Driver by defining Status Tags.

DIS, AIS, FIA, DPIS, CNTS, DOS, AOS are data types for Status flag.

For Tags that you want to define Status Tag, you need to define a new tag with Data type changed to Status Type and with new Name. Other Parameters are no changed. It should define exactly after Main Tag. Look at following examples:

```
<Tag Name="DITag1" Type="DI" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="DITag1.s" Type="DIs" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="AITag1" Type="AI" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="AITag1.s" Type="AIS" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="FITag1" Type="FI" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="FITag1.s" Type="FIS" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="CNTTag1" Type="CNT" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="CNTTag1.s" Type="CNTS" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="DOTag1" Type="DO" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="DOTag1.s" Type="DOS" Class="1" Init="0" Address="1" Log="0" />
```

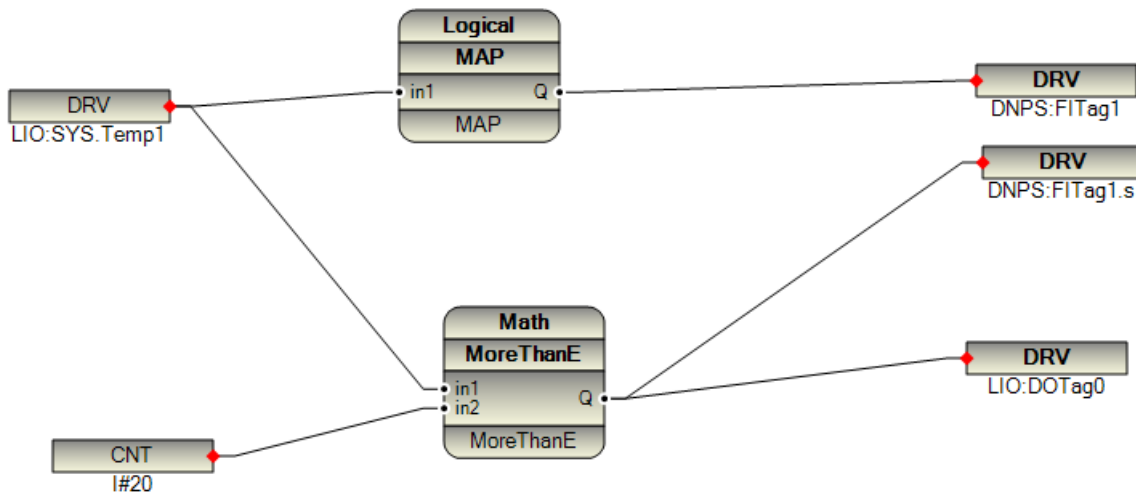
```
<Tag Name="AOTag1" Type="AO" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="AOTag1.s" Type="AOs" Class="1" Init="0" Address="1" Log="0" />
```

For Tags without Status Tag , psle consider Tag always Online .

For Tags with Status Tag defined, you need to pass 1 for online and 0 for Offline Status of Tag in your logic.

Look at following sample: When Temperature is more than 20 deg, then FITag1 Status is Online and valid. Otherwise it is Offline.



Example 2: suppose you connect COM3 of AMS-R3010 RTU to Power meter by Modbus Master. Then you can link online status of DNP3 Tags for power meter to Modbus SYS.Online Tag.

So in Master SCADA when there is no communication between RTU and Power meter, it will show Tag Offline.

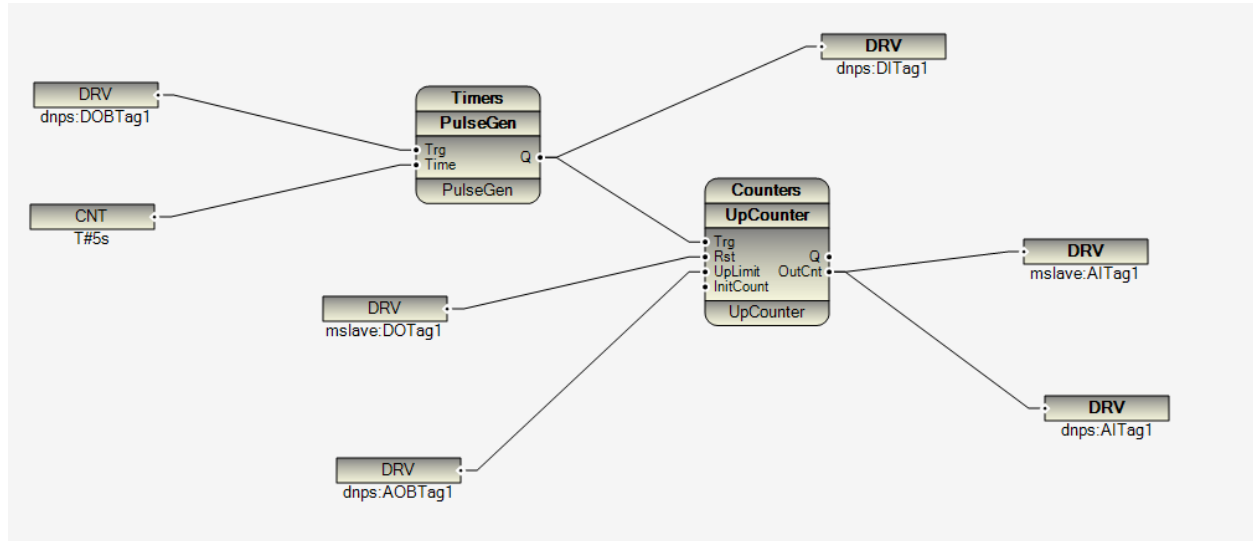
DNP3 Slave driver Operation:

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)



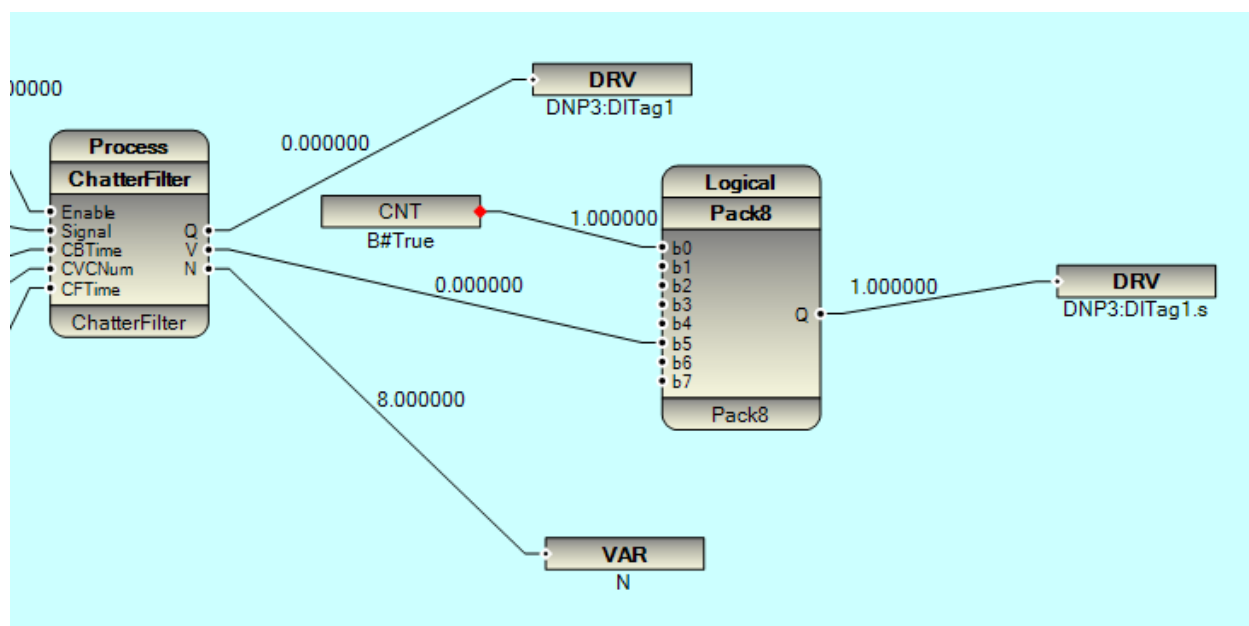
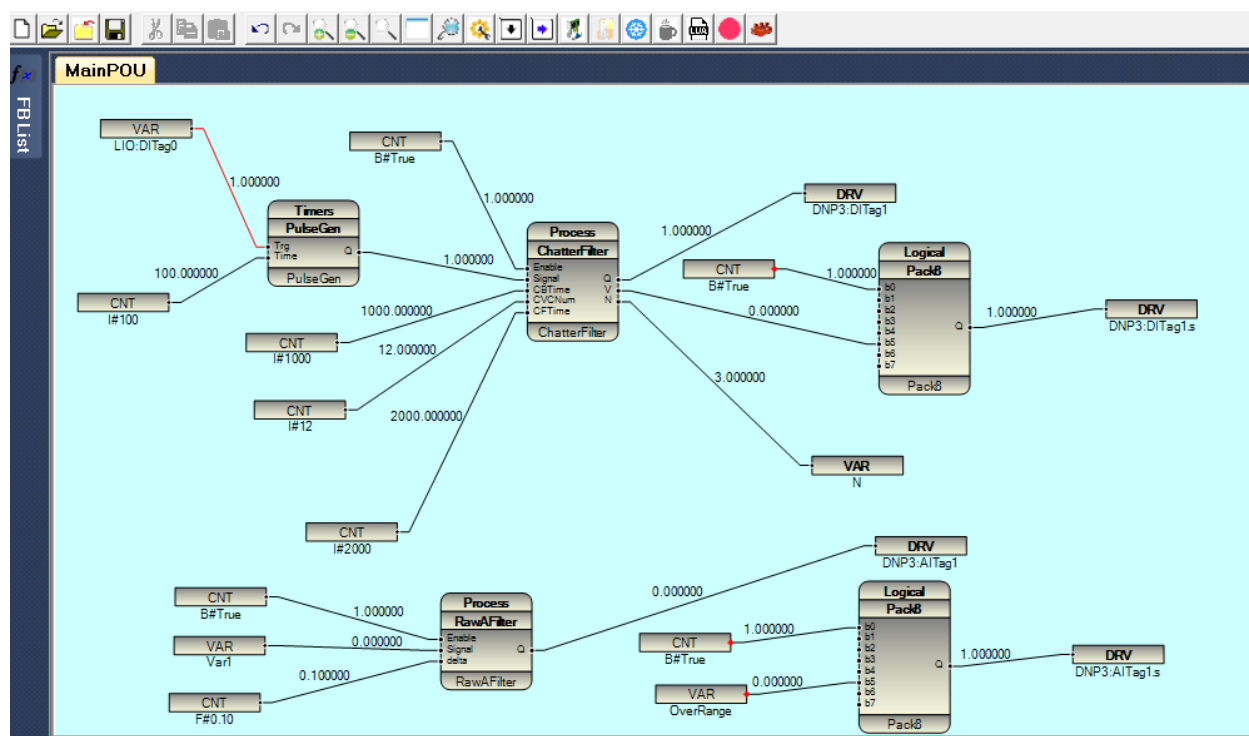
In above logic we have following DNP signals:

- 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

Sample Logic for handling State Flag for Digital and Analog Signals:



Test DNP3 driver with Kepware OPC Server

For testing DNP3 Driver you can use KepWare OPC Server. Please download KepWare OPC suite from www.kepware.com

Our sample configuration is as following. You can download it from www.pbscontrol.com/psleSample/dnp_Kepware.zip

DNP Type: TCP communication with port Number 20,000

Unsolicited communication: Automatic and Master is reading class1 events every 5 Sec

RTU Type: AMS-R3010 RTU

RTU IP: 192.168.1.137

RTU DNP3 ID = 3

Master ID = 1

Master IP Address: 192.168.1.152

Keep Alive Timer in RTU (NoCommTimeout): 60 Sec , we will set Keep Alive time to 20 sec in master .

RTU DNP3 Tags as Following :

```
<Tag Name="MasterOnline" Type="SYS" Class="0" Init="0" Address="0" Log="0" />
```

```
<Tag Name="DITag1" Type="DI" Class="1" Init="0" Address="1" Log="0" />
```

```
<Tag Name="DITag2" Type="DI" Class="1" Init="0" Address="2" Log="0" />
```

```
<Tag Name="DITag3" Type="DI" Class="1" Init="0" Address="3" Log="0" />
```

```
<Tag Name="DITag4" Type="DI" Class="1" Init="0" Address="4" Log="0" />
```

```
<Tag Name="DITag5" Type="DI" Class="1" Init="0" Address="5" Log="0" />
```

```
<Tag Name="DITag6" Type="DI" Class="1" Init="0" Address="6" Log="0" />
```

```
<Tag Name="DITag7" Type="DI" Class="1" Init="0" Address="7" Log="0" />
```

```
<Tag Name="DITag8" Type="DI" Class="1" Init="0" Address="8" Log="0" />
```

```
<Tag Name="DITag9" Type="DI" Class="1" Init="0" Address="9" Log="0" />
```

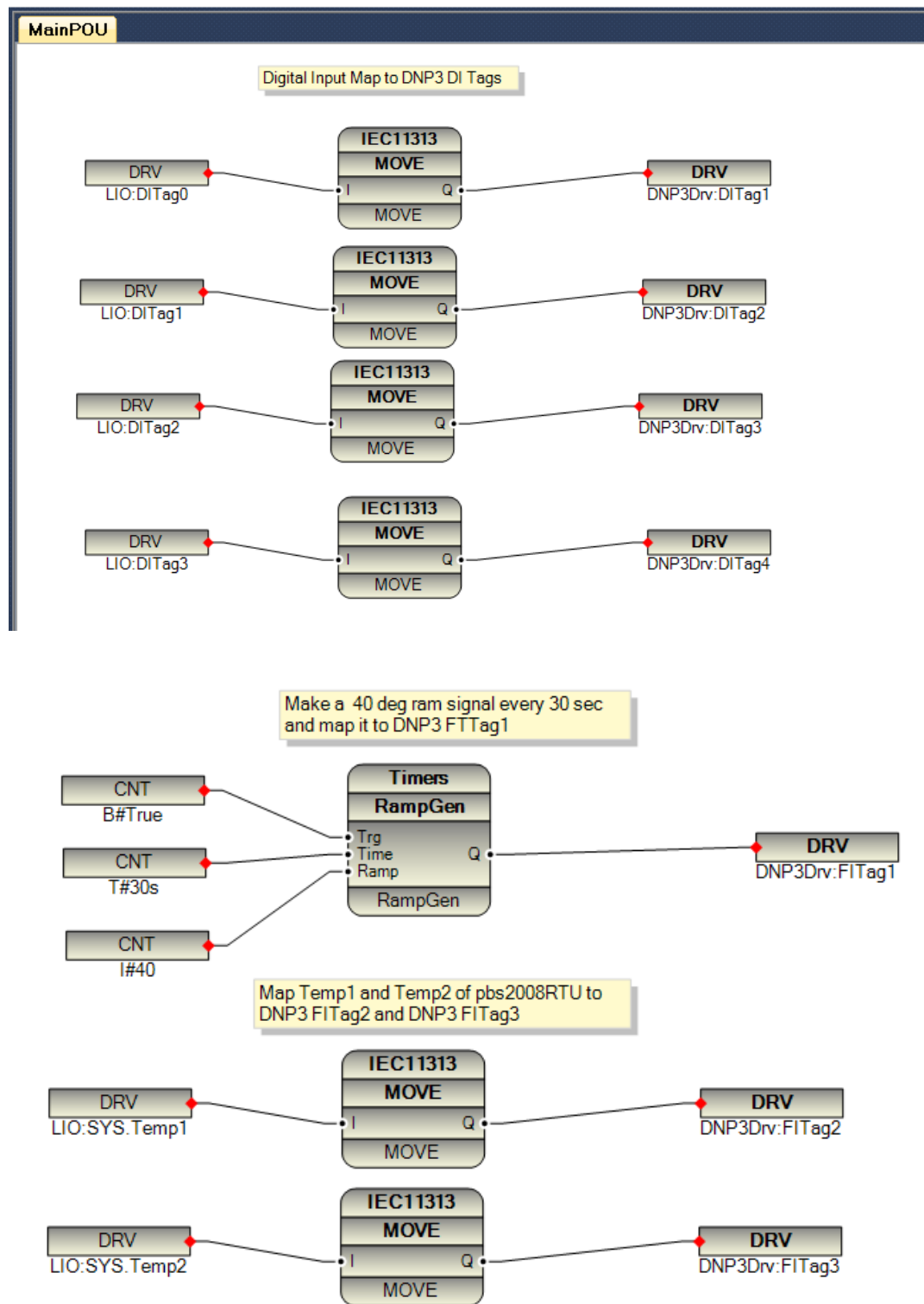
```
<Tag Name="DITag10" Type="DI" Class="1" Init="0" Address="10" Log="0" />
```

```
<Tag Name="AITag1" Type="AI" Class="1" Init="0" Address="1" Log="0" />
```

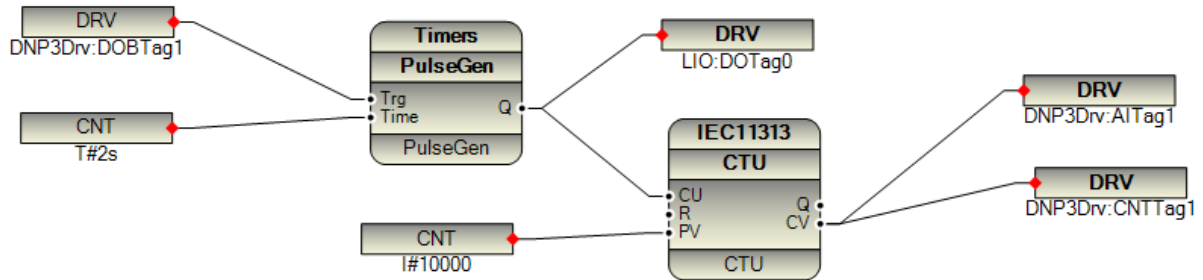
```
<Tag Name="AITag2" Type="AI" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="AITag3" Type="AI" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="AITag4" Type="AI" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="AITag5" Type="AI" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="AITag6" Type="AI" Class="1" Init="0" Address="6" Log="0" />
<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="AITag9" Type="AI" Class="1" Init="0" Address="9" Log="0" />
<Tag Name="AITag10" Type="AI" Class="1" Init="0" Address="10" Log="0" />
<Tag Name="FITag1" Type="FI" Class="1" Init="0" Address="11" Log="0" />
<Tag Name="FITag2" Type="FI" Class="1" Init="0" Address="12" Log="0" />
<Tag Name="FITag3" Type="FI" Class="1" Init="0" Address="13" Log="0" />
<Tag Name="FITag4" Type="FI" Class="1" Init="0" Address="14" Log="0" />
<Tag Name="FITag5" Type="FI" Class="1" Init="0" Address="15" Log="0" />
<Tag Name="FITag6" Type="FI" Class="1" Init="0" Address="16" Log="0" />
<Tag Name="FITag7" Type="FI" Class="1" Init="0" Address="17" Log="0" />
<Tag Name="FITag8" Type="FI" Class="1" Init="0" Address="18" Log="0" />
<Tag Name="FITag9" Type="FI" Class="1" Init="0" Address="19" Log="0" />
<Tag Name="FITag10" Type="FI" Class="1" Init="0" Address="20" 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="CNTTag9" Type="CNT" Class="1" Init="0" Address="9" Log="0" />
<Tag Name="CNTTag10" Type="CNT" Class="1" Init="0" Address="10" Log="0" />
<Tag Name="DOBTag1" Type="DOB" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="DOBTag2" Type="DOB" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="DOBTag3" Type="DOB" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="DOBTag4" Type="DOB" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="DOBTag5" Type="DOB" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="AOBTag1" Type="AOB" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="AOBTag2" Type="AOB" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="AOBTag3" Type="AOB" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="AOBTag4" Type="AOB" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="AOBTag5" Type="AOB" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="DOTag1" Type="DO" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="DOTag2" Type="DO" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="DOTag3" Type="DO" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="DOTag4" Type="DO" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="DOTag5" Type="DO" Class="1" Init="0" Address="5" Log="0" />
<Tag Name="AOTag1" Type="AO" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="AOTag2" Type="AO" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="AOTag3" Type="AO" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="AOTag4" Type="AO" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="AOTag5" Type="AO" Class="1" Init="0" Address="5" Log="0" />

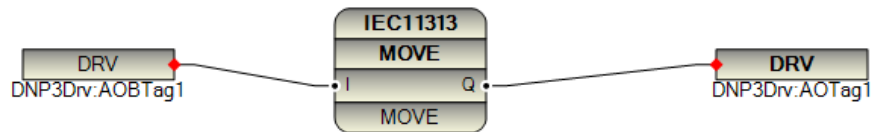
Develop RTU logic as following:



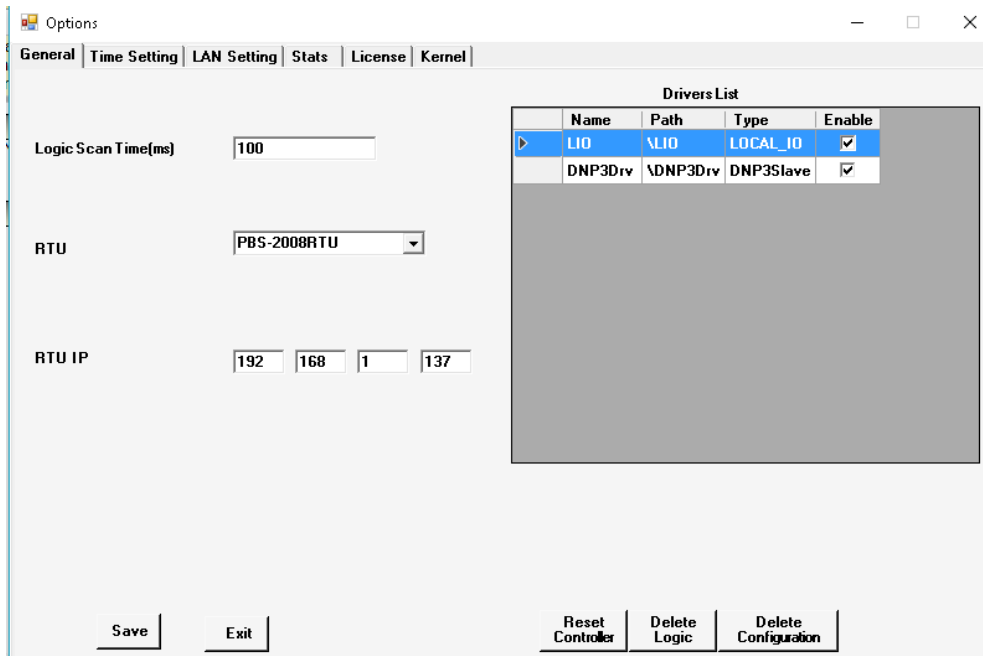
Make 2 Sec Pulse Gen and use Up Counter to count pulse
and map to DNP3TagAI1 and DNP3 Counter 1. Link Pulse Gen Start Command to DNP3 DOB Tag1
Link Pulse gen Output to RTU DOTag0



Map DNP3 AOB Tag1 to DNP3 AOTag1

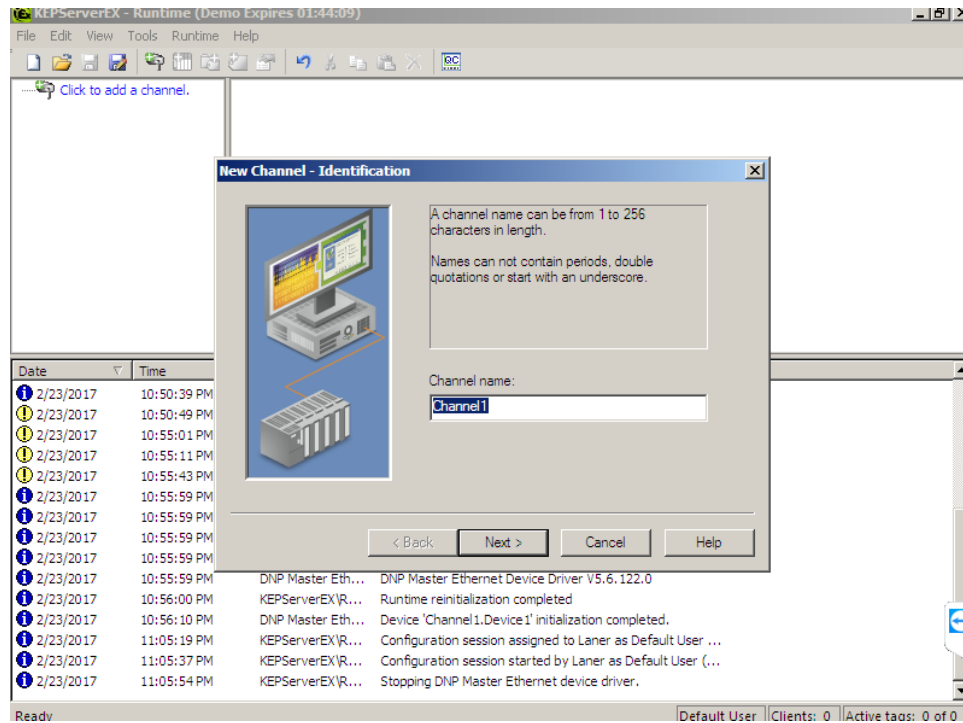


RTU Driver Configuration: Define one Local IO and one DNP3 Slave Driver.

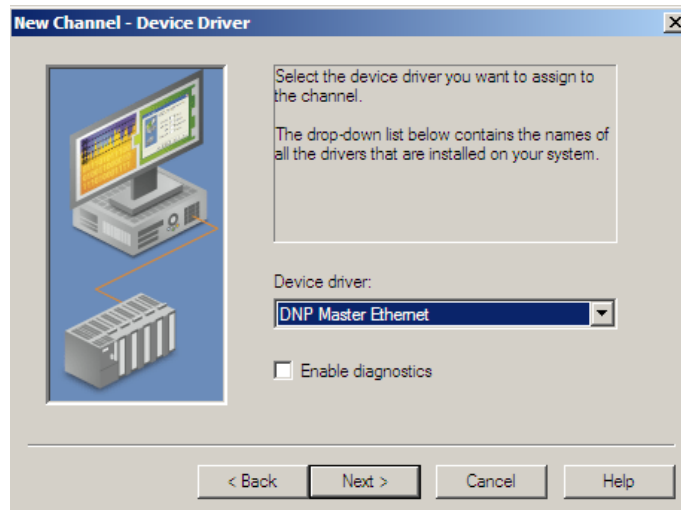


Step 1 : run Kepware Configurator Software

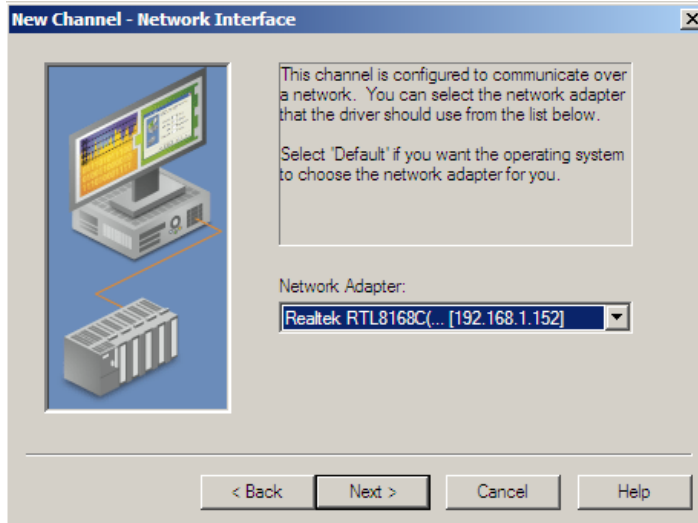
Step 2 : Add new communication channel to configuration



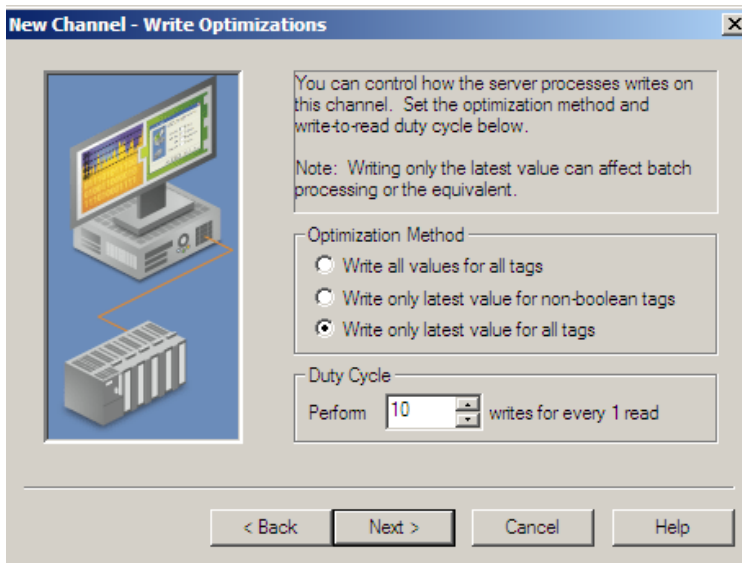
Step 3 – Select “DNP Master Ethernet” to communicate with RTU with DNP3 over TCP connection.



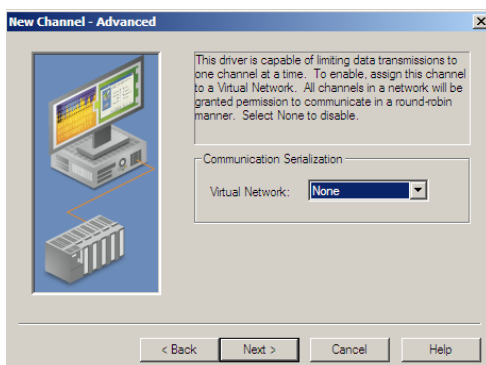
Step 4 : Select your PC Ethernet card



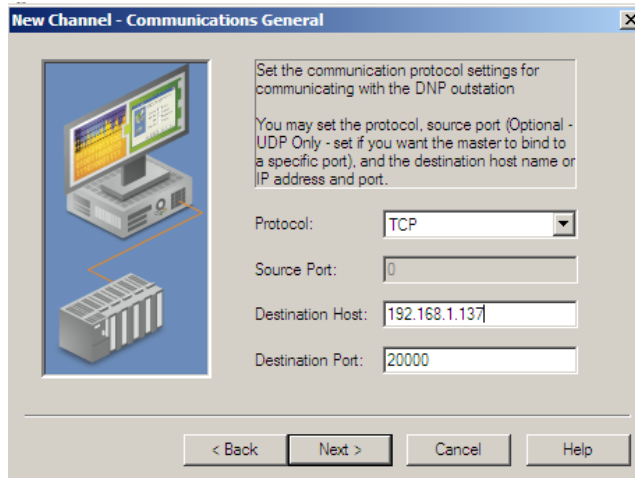
Step 5 : Select default options in Write Optimization parameters



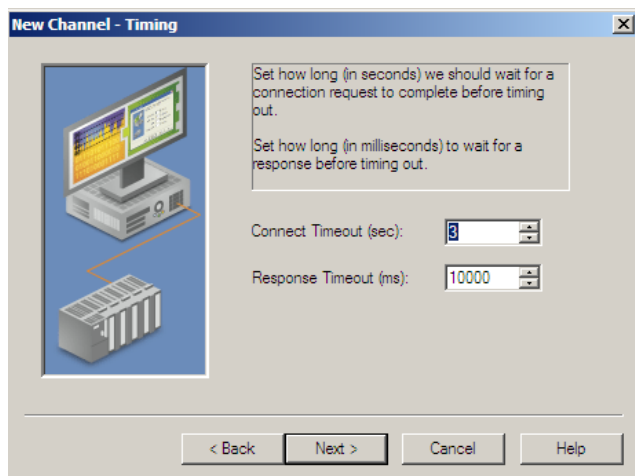
Step 6 : No Need to select VPN in this step .



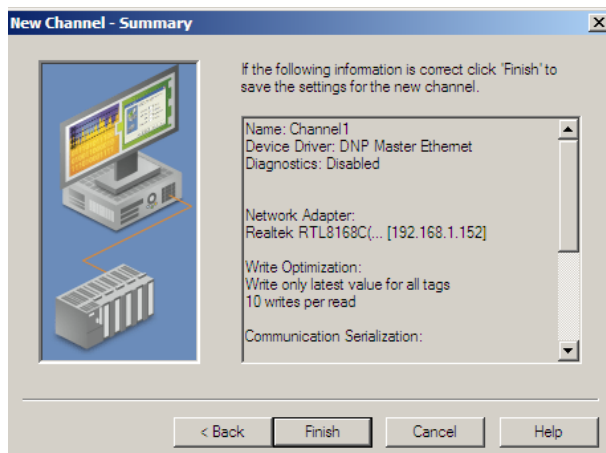
Step 7 : select TCP protocol , Write IP address of RTU and communication port (20,000 as default) in this stage .



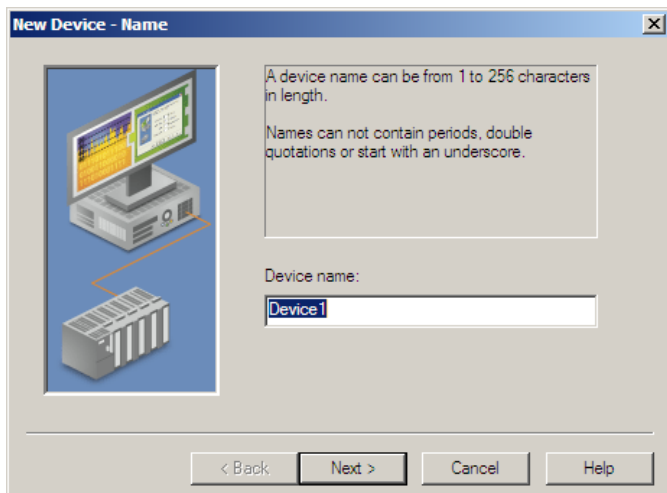
Step 8 : Keep timeout times as default .



Step 9 : Communication channel configuration is done on this stage . Click on Finish button .

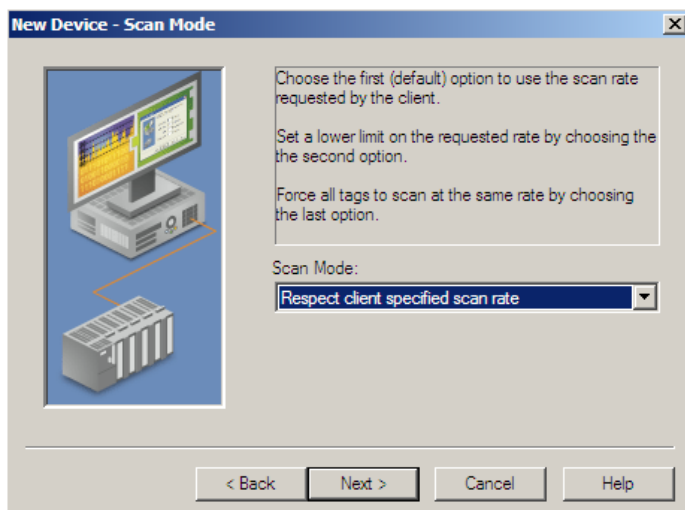


Step 10 : click on Add device item .



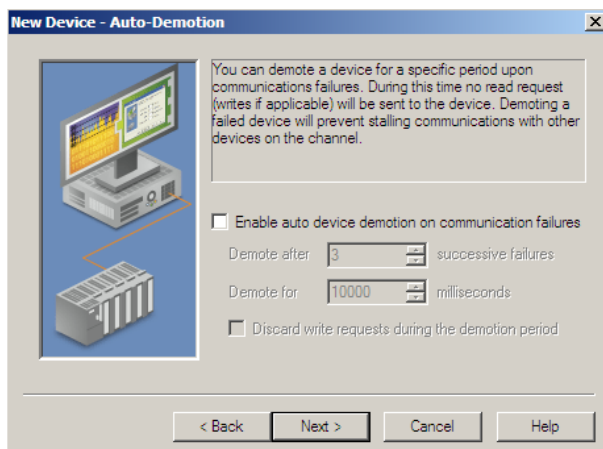
The 'New Device - Name' dialog box features a blue header bar with the title and a close button. On the left, there is a graphic of a computer monitor and a server rack. The main area contains two paragraphs of text: 'A device name can be from 1 to 256 characters in length.' and 'Names can not contain periods, double quotations or start with an underscore.' Below this is a 'Device name:' label followed by a text input field containing 'Device1'. At the bottom, there are four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Step 11 : Keep Scan mode as default .



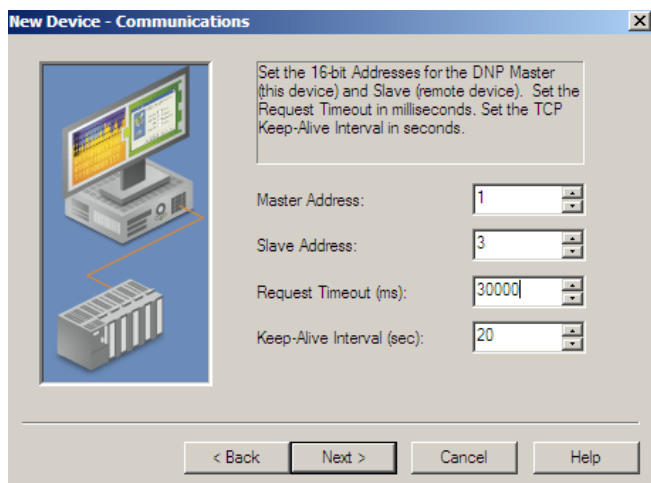
The 'New Device - Scan Mode' dialog box has a blue header bar with the title and a close button. It includes the same computer and server rack graphic on the left. The text area explains the scan mode options: 'Choose the first (default) option to use the scan rate requested by the client.', 'Set a lower limit on the requested rate by choosing the the second option.', and 'Force all tags to scan at the same rate by choosing the last option.' Below the text is a 'Scan Mode:' label and a dropdown menu currently set to 'Respect client specified scan rate'. The bottom features four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Step 12 : Keep Auto Demotion as default .



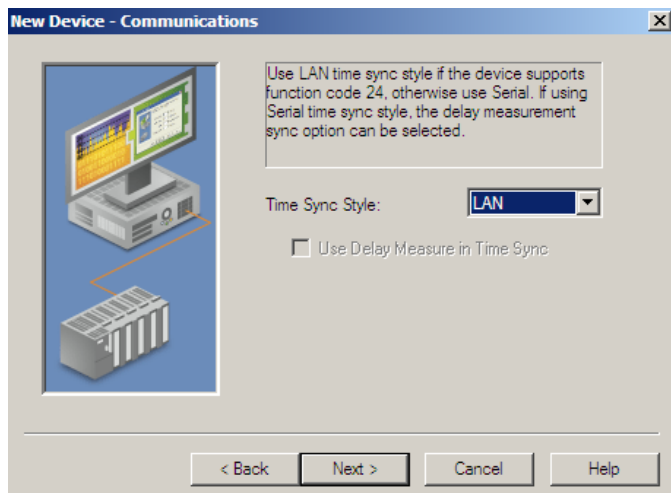
The 'New Device - Auto-Demotion' dialog box has a blue header bar with the title and a close button. It features the computer and server rack graphic on the left. The text area states: 'You can demote a device for a specific period upon communications failures. During this time no read request (writes if applicable) will be sent to the device. Demoting a failed device will prevent stalling communications with other devices on the channel.' Below this are three options: a checkbox for 'Enable auto device demotion on communication failures' (which is checked), a 'Demote after' field set to '3' with the unit 'successive failures', a 'Demote for' field set to '10000' with the unit 'milliseconds', and a checkbox for 'Discard write requests during the demotion period' (which is unchecked). The bottom has four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Step 13 : Write DNP3 Slave and Master Address and Keep-Alive Timer to 20 sec .



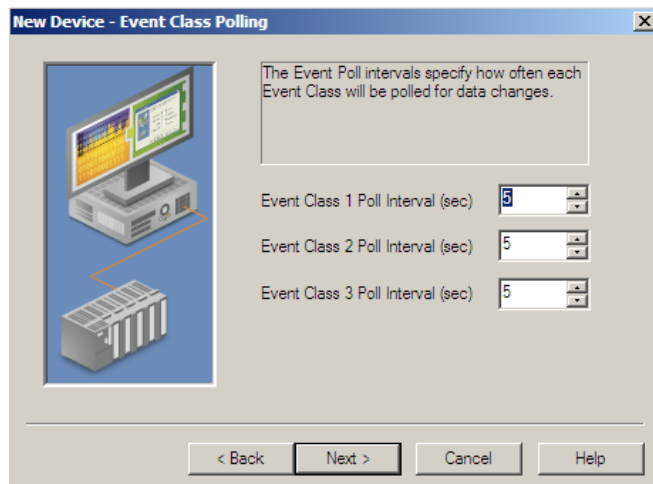
The 'New Device - Communications' dialog box contains a text box with instructions: 'Set the 16-bit Addresses for the DNP Master (this device) and Slave (remote device). Set the Request Timeout in milliseconds. Set the TCP Keep-Alive Interval in seconds.' Below this are four input fields: 'Master Address' (1), 'Slave Address' (3), 'Request Timeout (ms)' (30000), and 'Keep-Alive Interval (sec)' (20). At the bottom are buttons for '< Back', 'Next >', 'Cancel', and 'Help'.

Step 14 : Keep Time synchronization as LAN .



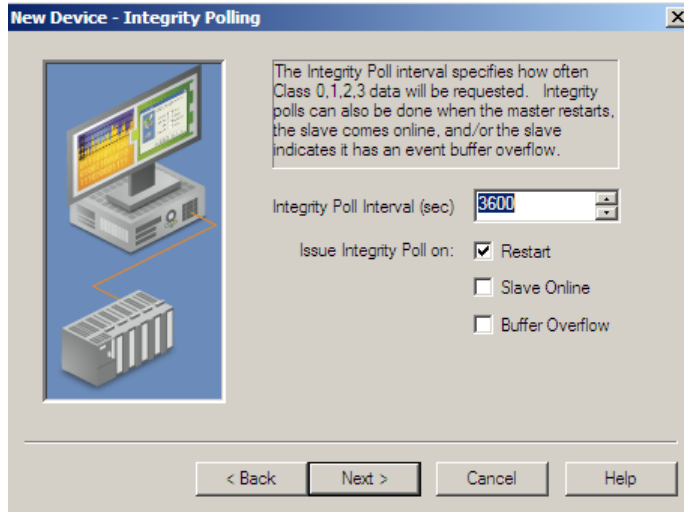
The 'New Device - Communications' dialog box shows a text box with instructions: 'Use LAN time sync style if the device supports function code 24, otherwise use Serial. If using Serial time sync style, the delay measurement sync option can be selected.' Below this is a 'Time Sync Style' dropdown menu set to 'LAN' and an unchecked checkbox labeled 'Use Delay Measure in Time Sync'. At the bottom are buttons for '< Back', 'Next >', 'Cancel', and 'Help'.

Step 15 : Set read class poll to 5 sec for reading changes from RTU every 5 Sec.

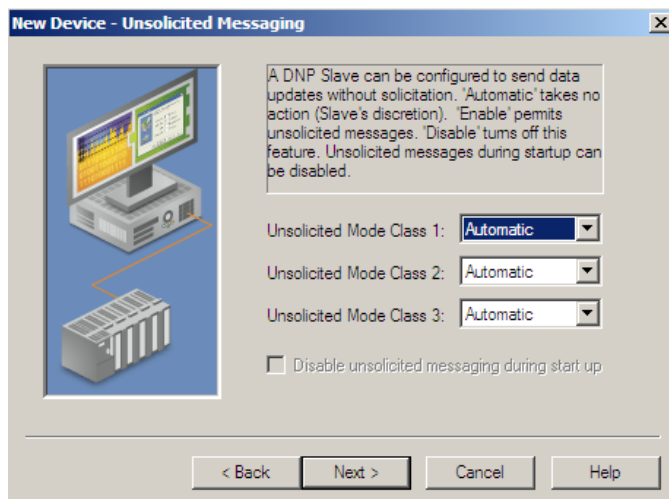


The 'New Device - Event Class Polling' dialog box contains a text box with instructions: 'The Event Poll intervals specify how often each Event Class will be polled for data changes.' Below this are three input fields: 'Event Class 1 Poll Interval (sec)' (5), 'Event Class 2 Poll Interval (sec)' (5), and 'Event Class 3 Poll Interval (sec)' (5). At the bottom are buttons for '< Back', 'Next >', 'Cancel', and 'Help'.

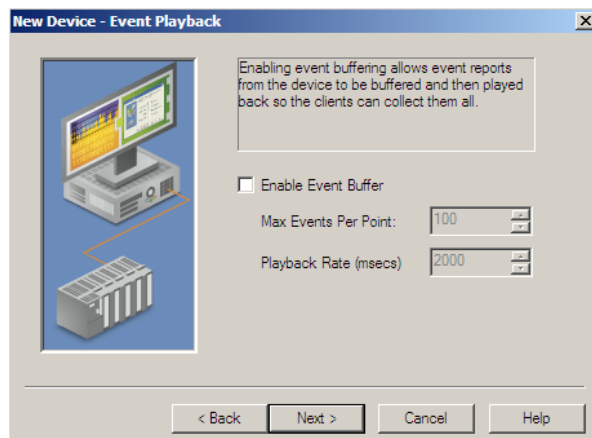
Step 16 : Set Send Integrity poll Parameter as following :



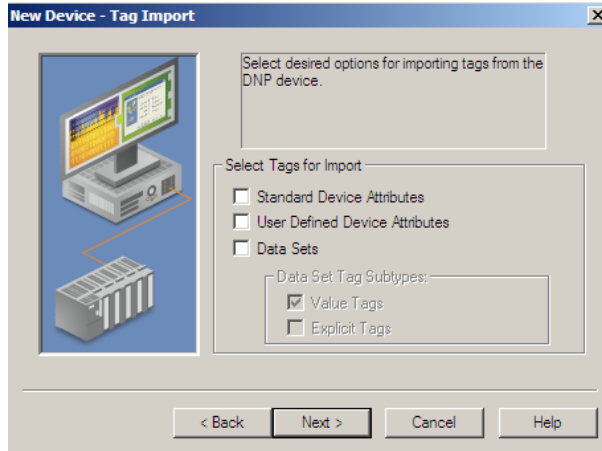
Step 17 : set Unsolicited communication to automatic as following :



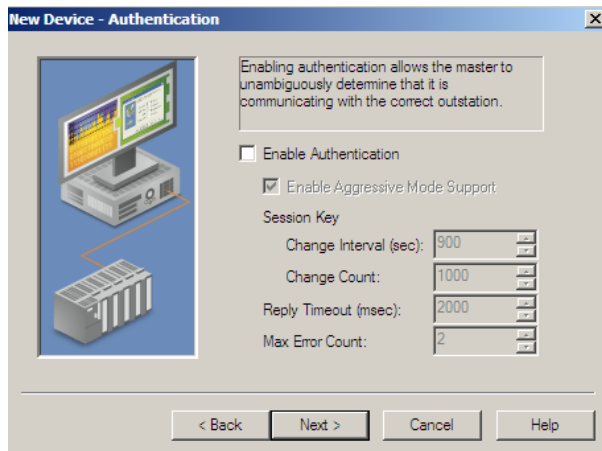
Step 18 : Keep Play Back Event to disable as default .



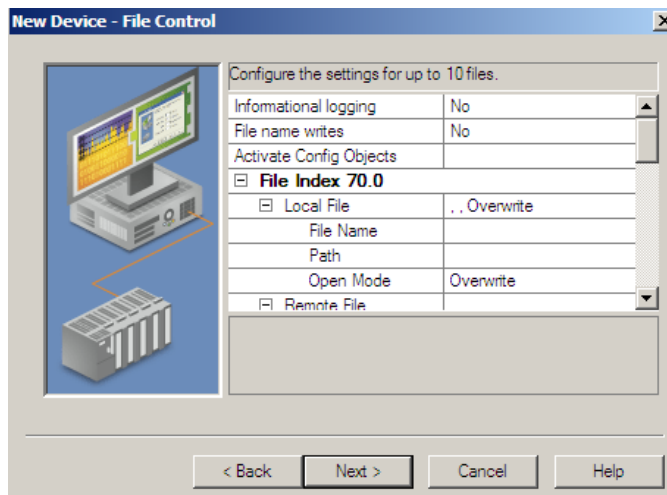
Step 19 : No need to import tags we will define tags manually.



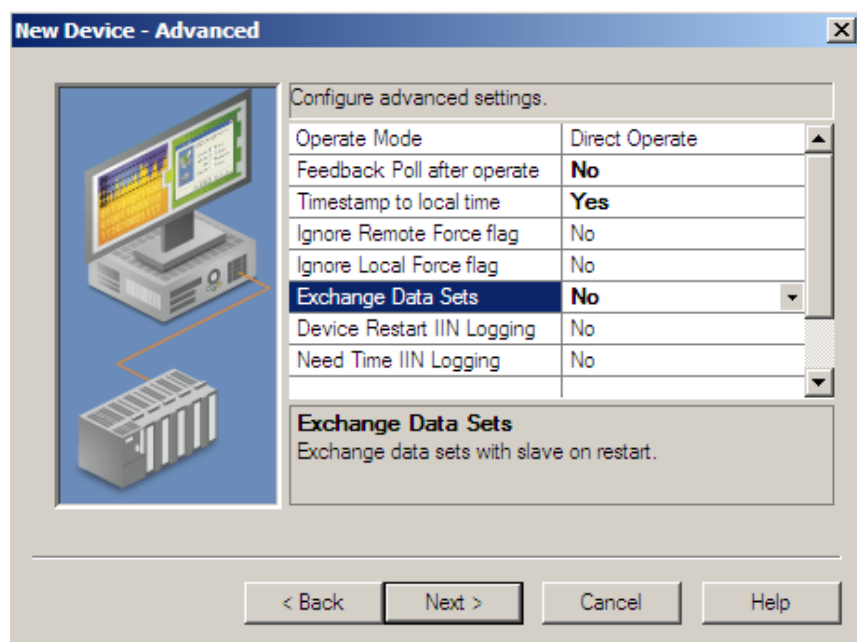
Step 20 : No Need to define Authentication .pbsSoftLogic DNP3 driver still not supports DNP3 Authentication .



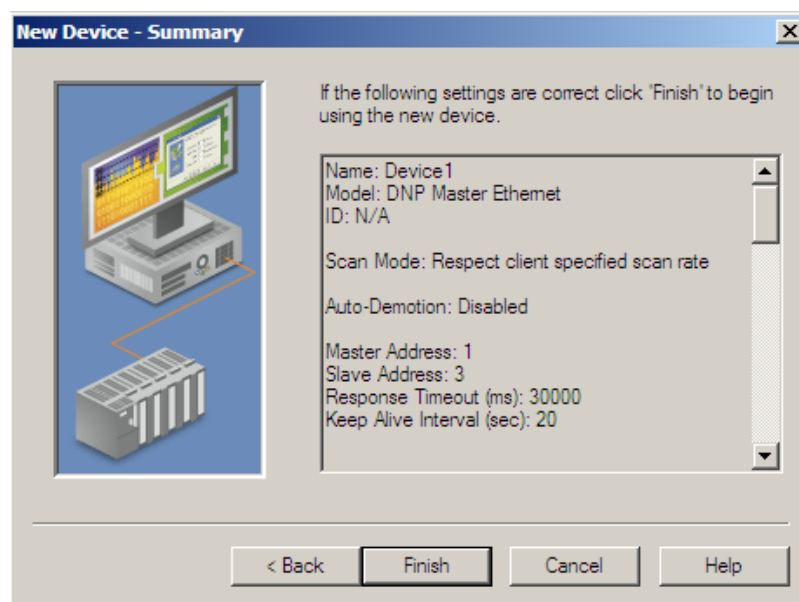
Step 21 : Pass File control page . pbsSoftLogic DNP3 driver still not supports File Transfer functionality .



Step 22: Set Advanced parameters as following :

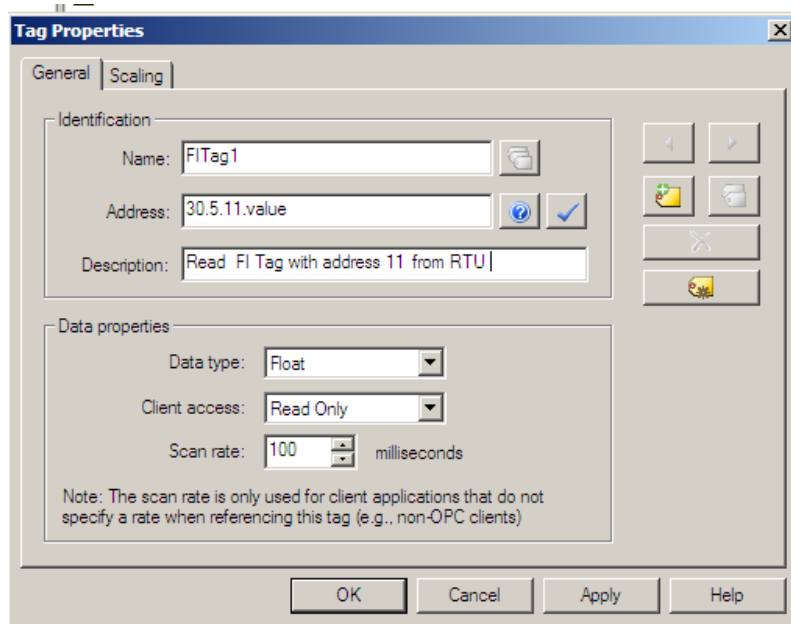


Step 23 : you finished Device configuration in this stage .Click on Finish Button .



Step 24 : In this stage you should define DNP3 Tags for OPC Server.

Reading Float Signal value from RTU by OPC Server define one float tag as following:



Tag address in OPC : DNPGroupNumber.DNPVariation.DNPTagAddress.Value

DNP3 Object Library		Group:	30
Group	Name: Analog Input	Variation:	5
Variation	Name: Single-precision, floating-point with flag	Type:	Static
		Parsing Codes:	Table 12-14

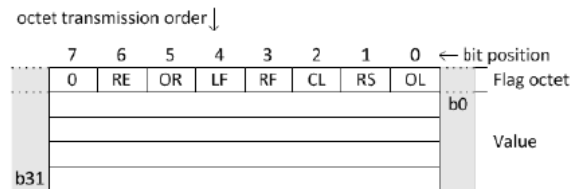
A.14.5.1 Description

Object group 30, variation 5 is used to report the current value of an analog input point. See 11.9.1 for a description of an Analog Input Point Type.

Variation 5 objects contain a flag octet and a single-precision, floating-point value.

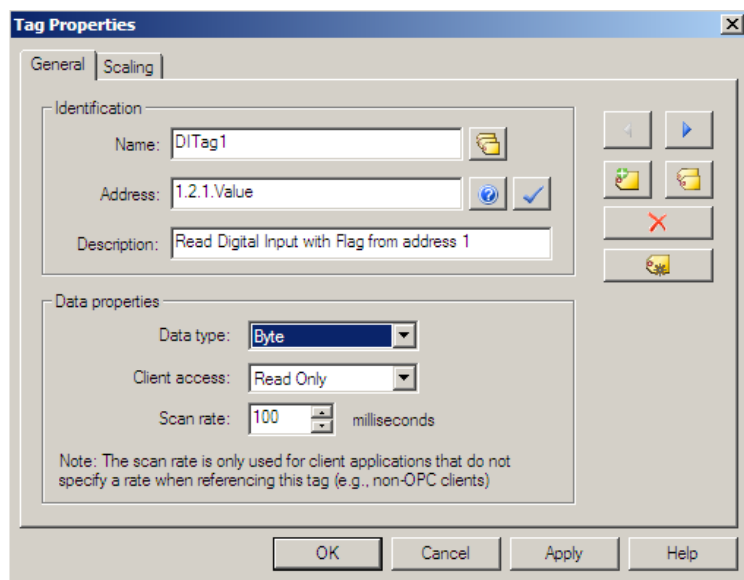
A.14.5.2 Coding

A.14.5.2.1 Pictorial



Item ID	Data Type	Value	Timestamp	Quality
Channel1.Device1._Integr...	DWord	3600	01:46:15.132	Good
Channel1.Device1._Maste...	DWord	1	01:46:15.132	Good
Channel1.Device1._Protocol	Byte	0	01:46:15.132	Good
Channel1.Device1._Slave...	DWord	3	01:46:15.132	Good
Channel1.Device1._Sourc...	Word	0	01:46:15.132	Good
Channel1.Device1.FITag1	Float	23.4948	01:46:04.049	Good

Reading DNP3 Digital Input Tag from RTU by OPC Server: Define an OPC tag as following:



A.2.2 Binary input—with flags

DNP3 Object Library

Group		Group:	1
Name:	Binary Input	Variation:	2
Variation		Type:	Static
Name:	With flags	Parsing	Table
		Codes:	12-2

A.2.2.1 Description

Object group 1, variation 2 is used to report the current value of a binary input point. See 11.9.8 for a description of a Binary Input Point Type.

Variation 2 objects contain a status octet that includes the state of the binary input.

A.2.2.2 Coding

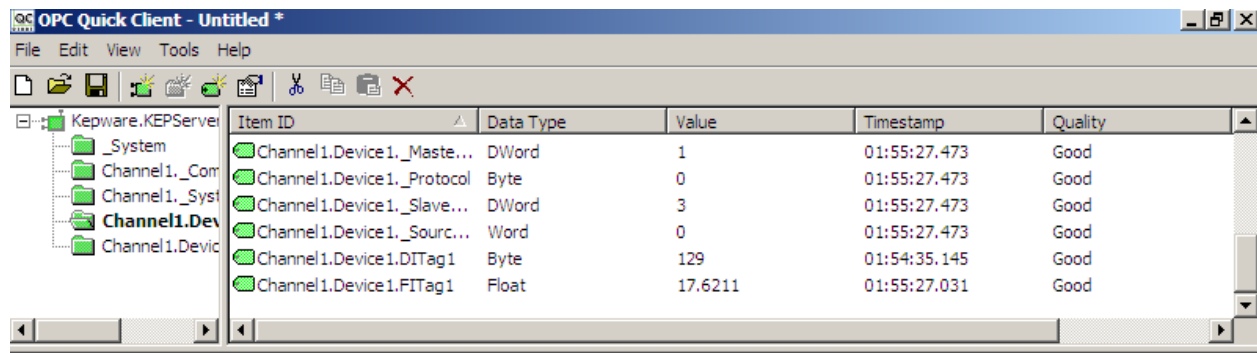
A.2.2.2.1 Pictorial

octet transmission order ↓

	7	6	5	4	3	2	1	0	← bit position
	ST	0	CF	LF	RF	CL	RS	OL	State and flag octet

A.2.2.2.2 Formal structure**BSTR8: Flag Octet**

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	CHATTER_FILTER
Bit 6:	Reserved, always 0
Bit 7:	STATE—Has a value of 0 or 1, representing the state of the physical or logical input.

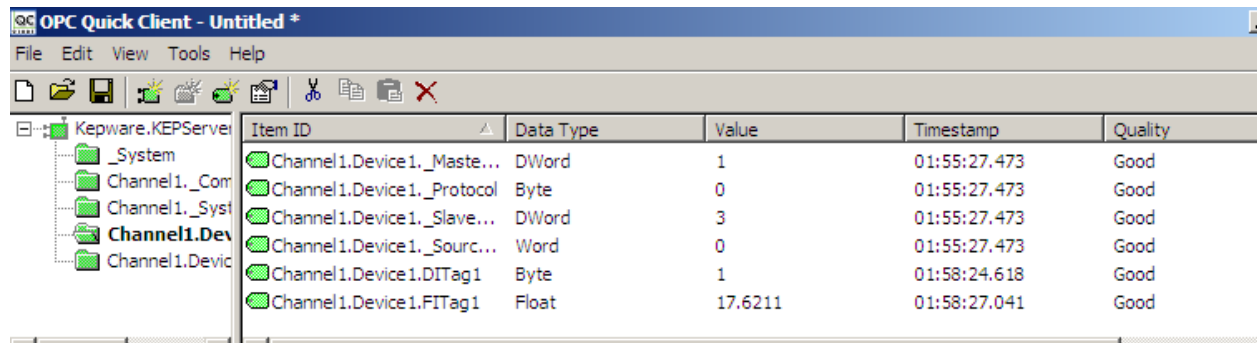


Item ID	Data Type	Value	Timestamp	Quality
Channel1.Device1._Maste...	DWord	1	01:55:27.473	Good
Channel1.Device1._Protocol	Byte	0	01:55:27.473	Good
Channel1.Device1._Slave...	DWord	3	01:55:27.473	Good
Channel1.Device1._Sourc...	Word	0	01:55:27.473	Good
Channel1.Device1.DITag1	Byte	129	01:54:35.145	Good
Channel1.Device1.FITag1	Float	17.6211	01:55:27.031	Good

When Value of Signal is 129 in OPC Server , it means that :

Bit 0 is Online

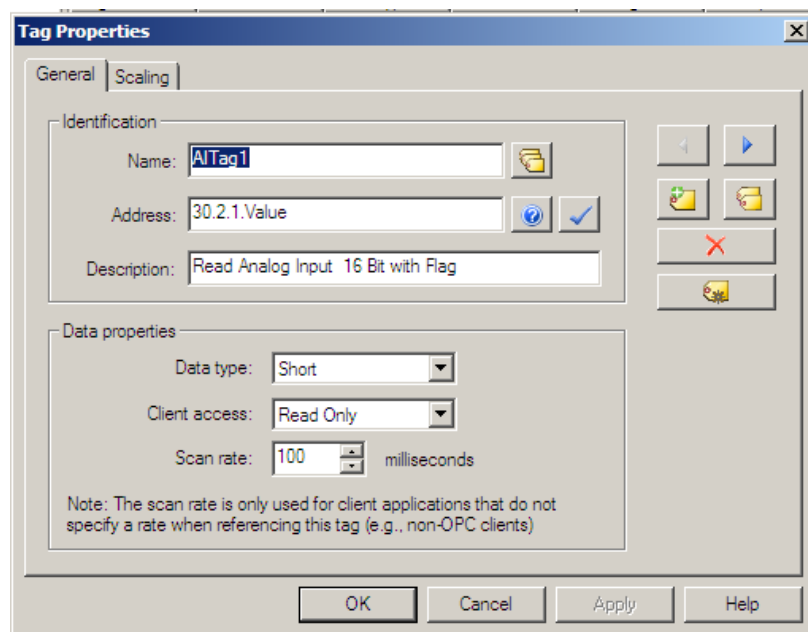
Bit 7 is 1 (Signal Value is 1)



Item ID	Data Type	Value	Timestamp	Quality
Channel1.Device1._Maste...	DWord	1	01:55:27.473	Good
Channel1.Device1._Protocol	Byte	0	01:55:27.473	Good
Channel1.Device1._Slave...	DWord	3	01:55:27.473	Good
Channel1.Device1._Sourc...	Word	0	01:55:27.473	Good
Channel1.Device1.DITag1	Byte	1	01:58:24.618	Good
Channel1.Device1.FITag1	Float	17.6211	01:58:27.041	Good

When Value of signal is 1 It means Bit 0 is online and Bit 7 (Signal Value) is 0

Reading Analog Input 16 Bit with Flag :



For reading Analog Input tags from RTU you need to define OPC Tag with Group 30 .

For Reading as 16 Bit Value, use Variation 2.

OPC Quick Client - Untitled *

File Edit View Tools Help

Item ID	Data Type	Value	Timestamp	Quality
Channel1.Device1._Slave...	DWord	3	09:41:58.185	Good
Channel1.Device1._Sourc...	Word	0	09:41:58.185	Good
Channel1.Device1.AITag1	Short	42	09:45:01.724	Good
Channel1.Device1.DITag1	Byte	129	09:41:59.090	Good
Channel1.Device1.DOB1	Boolean	0	09:41:59.090	Good
Channel1.Device1.FITag1	Float	21.8166	09:45:02.027	Good

A.14.2 Analog input—16-bit with flag**DNP3 Object Library**

Group		Group:	30
Name:	Analog Input	Variation:	2
Variation		Type:	Static
Name:	16-bit with flag	Parsing Codes:	Table 12-14

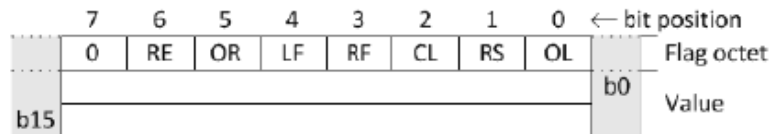
A.14.2.1 Description

Object group 30, variation 2 is used to report the current value of an analog input point. See 11.9.1 for a description of an Analog Input Point Type.

Variation 2 objects contain a flag octet and a 16-bit, signed integer value.

A.14.2.2 Coding**A.14.2.2.1 Pictorial**

octet transmission order ↓

**A.14.2.2.2 Formal structure**

BSTR8: Flag Octet

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	OVER_RANGE
Bit 6:	REFERENCE_ERR
Bit 7:	Reserved, always 0.

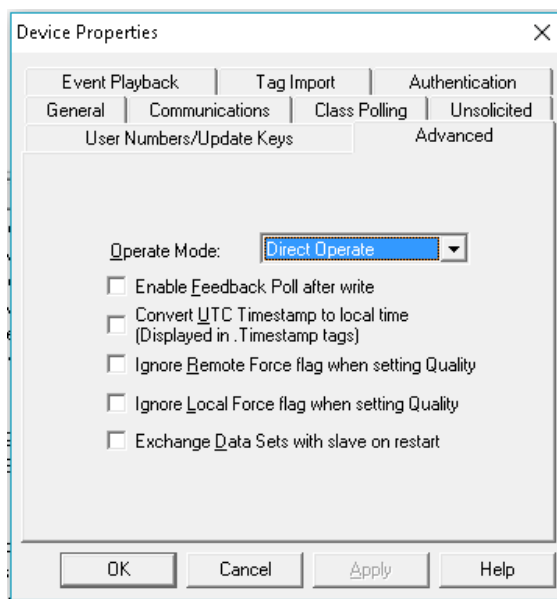
INT16: Value

This is the most recently measured, obtained, or computed value.

Range is -32 768 to +32 767.

Important Points for AOB and DOB Tag Writing by kepware DNP3 OPC Server to RTU:

- Always for any AOB or DOB Tags you MUST define AO and DO Tags with same addressing. Otherwise Kepware AOB or DOB tags will not get online status.
- In Device Advanced Properties , remove check mark for " Enable Feed Back Poll after Write"



DNP3 Driver is only supported Short and Long Write for AOB Tags.

So you should define AOB or DOB tags as following:

AOBTag1	40.1.1.Value	Long
AOBTag2	40.2.2.Value	Short
DOBTag1	10.0.1.Value	Boolean

40 : AO Tag Write

1: Long Write

2:Short Write

12 – 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.

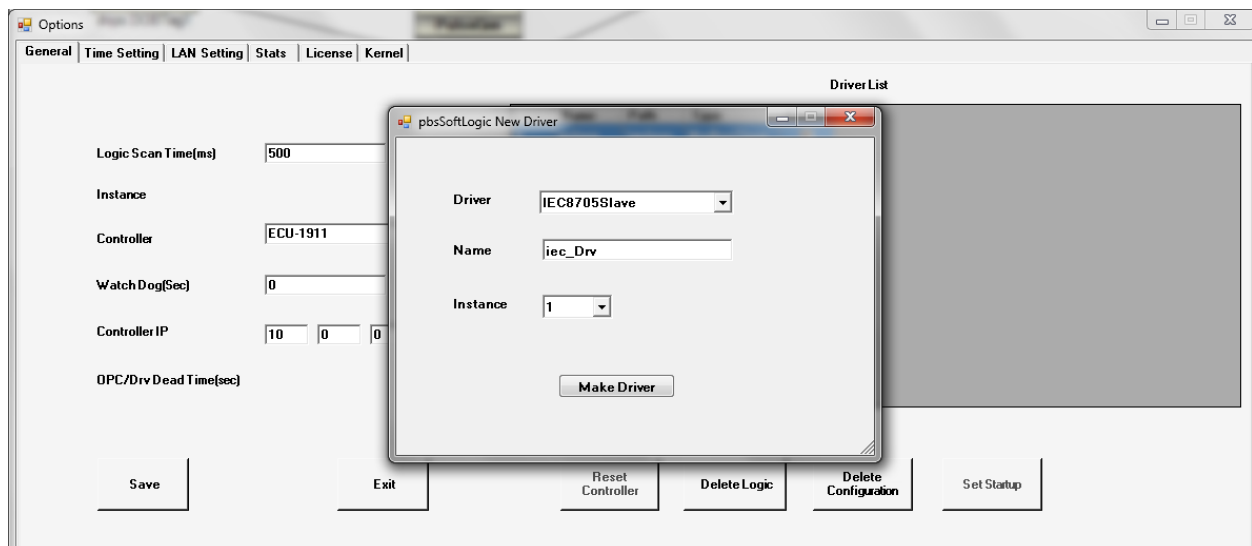
IEC870-5-101 is communicating over RS232 and IEC870-5-104 is communicating over TCP.

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 receive sequence number to send state variable</Desc>

<Value>12</Value>

</Node>

<Node>

<Name>WParam</Name>

<Desc>WParameter 1~ 32767 Latest ACK after receiving 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="D0Tag1" Type="DO" Class="1" Init="0" Address="1" Log="0" />
```

IEC101/104 State Tag:

Based on IEC101 /104 standards each Tag has a quality descriptor field.

Quality descriptor shows tag status.

7.2.6.3 Quality descriptor (separate octet)

The quality descriptor consists of five defined quality bits which may be set independently from each other. The quality descriptor provides the controlling station with additional information on the quality of an information object.

QDS	:=	CP8{OV,RES,BL,SB,NT,IV}	
OV	:=	BS1[1]<0..1>	(Type 6)
<0>	:=	no overflow	
<1>	:=	overflow	
RES = RESERVE	:=	BS3[2..4]<0>	(Type 6)
BL	:=	BS1[5]<0..1>	(Type 6)
<0>	:=	not blocked	
<1>	:=	blocked	
SB	:=	BS1[6]<0..1>	(Type 6)
<0>	:=	not substituted	
<1>	:=	substituted	
NT	:=	BS1[7]<0..1>	(Type 6)
<0>	:=	topical	
<1>	:=	not topical	
IV	:=	BS1[8]<0..1>	(Type 6)
<0>	:=	valid	
<1>	:=	invalid	

OV = OVERFLOW/NO OVERFLOW

The value of the INFORMATION OBJECT is beyond a predefined range of value (mainly applicable to analog values).

BL = BLOCKED/NOT BLOCKED

The value of the INFORMATION OBJECT is blocked for transmission; the value remains in the state that was acquired before it was blocked. Blocking and deblocking may be initiated for example by a local lock or a local automatic cause.

SB = SUBSTITUTED/NOT SUBSTITUTED

The value of the INFORMATION OBJECT is provided by the input of an operator (dispatcher) or by an automatic source.

NT = NOT TOPICAL/TOPICAL

A value is topical if the most recent update was successful. It is not topical if it was not updated successfully during a specified time interval or if it is unavailable.

IV = INVALID/VALID

A value is valid if it was correctly acquired. After the acquisition function recognizes abnormal conditions of the information source (missing or non-operating updating devices) the value is then marked invalid. The value of the INFORMATION OBJECT is not defined under this condition. The mark INVALID is used to indicate to the destination that the value may be incorrect and cannot be used.

Intermediate devices may modify the quality descriptors BL, SB, NT and IV.

BL: if an intermediate device blocks the transmission of an information object, it shall assert the quality descriptor BL. Otherwise it shall report the quality descriptor BL as reported from the lower level device.

SB: if an intermediate device substitutes the value of an information object, it shall assert the quality descriptor SB. Otherwise it shall report the quality descriptor SB as reported from the lower level device.

NT: if an intermediate device cannot obtain the value of an information object, it shall assert the quality descriptor NT. Otherwise it shall report the quality descriptor NT as reported from the lower level device.

IV: if an intermediate device identifies that an information object is not valid, it shall assert the quality descriptor IV. Otherwise it shall report the quality descriptor IV as reported from the lower level device.

Example 1

Suppose that the monitored status of a circuit-breaker is blocked because the field interface is in test mode. In this case, the quality descriptor (BL = 1 "blocked") will be transferred unchanged through all system levels from the field interface to the controlling station.

Example 2

A substituted value may be assigned automatically or manually to a measured value, for example when the data acquisition is disturbed. This substituted measured value is transmitted to the controlling station with the quality bit SB = 1 substituted.

If the value of an information object is automatically marked with a new quality descriptor due to specific conditions, the quality descriptor may be reset manually or automatically when the conditions change.

If a given information object is normally only reported spontaneously, every change of the quality descriptor initiates a spontaneous transmission. Information objects with a time tag are transmitted with the point of time at which the change of the quality descriptor occurred.

7.2.6.1 Single-point information (IEV 371-02-07) with quality descriptor

SIQ	:=	CP8{SPI,RES,BL,SB,NT,IV}
SPI	:=	BS1[1]<0..1>
	<0>	:= OFF
	<1>	:= ON
RES = RESERVE	:=	BS3[2..4]<0>
BL	:=	BS1[5]<0..1>
	<0>	:= not blocked
	<1>	:= blocked
SB	:=	BS1[6]<0..1>
	<0>	:= not substituted
	<1>	:= substituted
NT	:=	BS1[7]<0..1>
	<0>	:= topical
	<1>	:= not topical
IV	:=	BS1[8]<0..1>
	<0>	:= valid
	<1>	:= invalid

7.2.6.2 Double-point information (IEV 371-02-08) with quality descriptor

DIQ	:=	CP8{DPI,RES,BL,SB,NT,IV}	
DPI	:=	UI2[1..2]<0..3>	(Type 1.1)
	<0>	:= indeterminate or intermediate state	
	<1>	:= determined state OFF	
	<2>	:= determined state ON	
	<3>	:= indeterminate state	
RES = RESERVE	:=	BS2[3..4]<0>	(Type 6)
BL	:=	BS1[5]<0..1>	(Type 6)
	<0>	:= not blocked	
	<1>	:= blocked	
SB	:=	BS1[6]<0..1>	(Type 6)
	<0>	:= not substituted	
	<1>	:= substituted	
NT	:=	BS1[7]<0..1>	(Type 6)
	<0>	:= topical	
	<1>	:= not topical	
IV	:=	BS1[8]<0..1>	(Type 6)
	<0>	:= valid	
	<1>	:= invalid	

Defining Tag State

You need to define a state tag exactly after IEC tag definition in IECSTags.xml file.

```
<Tag Name="DITag1" Type="DI" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="DITag1.s" Type="DIS" Class="1" Init="0" Address="1" Log="0" />

<Tag Name="DITag2" Type="DI" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="DITag3" Type="DI" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="DITag4" Type="DI" Class="1" Init="0" Address="4" Log="0" />

<Tag Name="AITag1" Type="AI" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="AITag1.s" Type="AIS" Class="1" Init="0" Address="1" Log="0" />

<Tag Name="AITag2" Type="AI" Class="1" Init="0" Address="2" Log="0" />
<Tag Name="AITag2.s" Type="AIS" Class="1" Init="0" Address="2" Log="0" />

<Tag Name="AITag3" Type="AI" Class="1" Init="0" Address="3" Log="0" />
<Tag Name="AITag4" Type="AI" Class="1" Init="0" Address="4" Log="0" />
<Tag Name="AITag8" Type="AI" Class="1" Init="0" Address="8" Log="0" />

<Tag Name="FITag1" Type="FI" Class="1" Init="0" Address="1" Log="0" />
<Tag Name="FITag1.s" Type="FIS" Class="1" Init="0" Address="1" Log="0" />
```

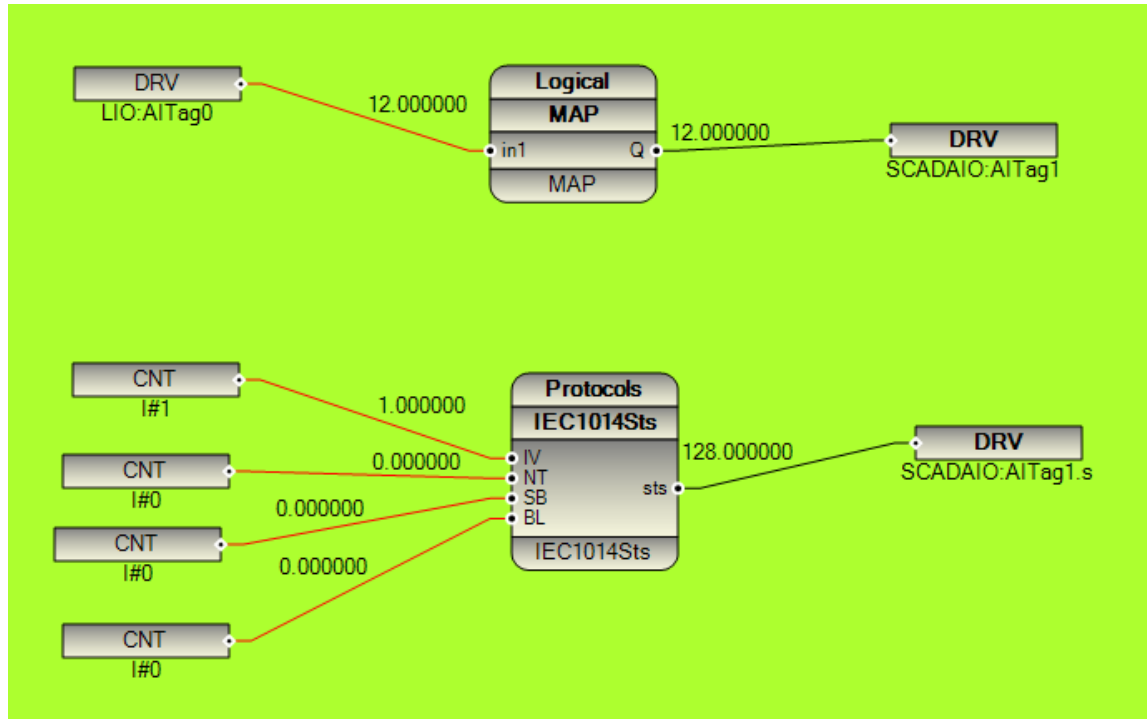
Note: State tag MUST be defined exactly after IEC Tag.

Look at Above example: DITag1 and DITag1.s Tag name MUST be different with IEC tag . You can add .s or _s to IEC as an example.

There are following State tag types:

- DIS : State Tag type for DI
- AIS : State Tag type for AI
- FIS : State Tag Type for FI
- DPIS : State Tag Type for DPI
- CNTS : State Tag Type for CNT (Counters)

You need to write State Tag in RTU Logic. You can use IEC1014Sts Function Block to make State Tag Signal in your logic. IV, NT, SB and BL are defined in the IEC101/104 standard.



Note: IEC1014Sts is a Lua Function block and you must transfer it to controller for proper operation of your logic. Please look at Lua User defined FB for detail information.

IEC1014Sts is used for DIS, DPIS, AIS and FIS State Tags.

For counters, you need to set only IV filed.

- IV = 0 Counter value is valid
- IV = 1 Counter value is not valid

So value of State tag for counter is only 0 or 1.

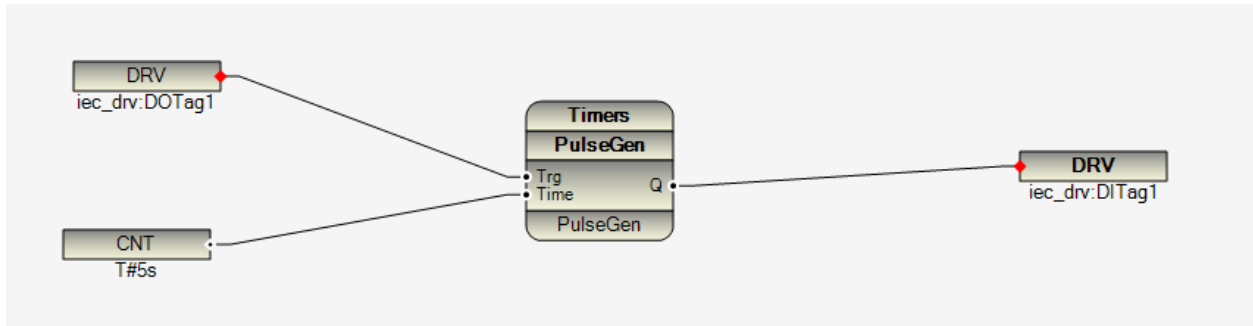
IEC 870-5 Slave driver Operation:

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

13 – S7 Communication Driver Configuration

pbsSoftLogic supports Siemens S7-Communication protocol for Read/Write Tags from/To all S7 Series PLCs .

pbssoftLogic supports Client side for S7-Communication protocol .

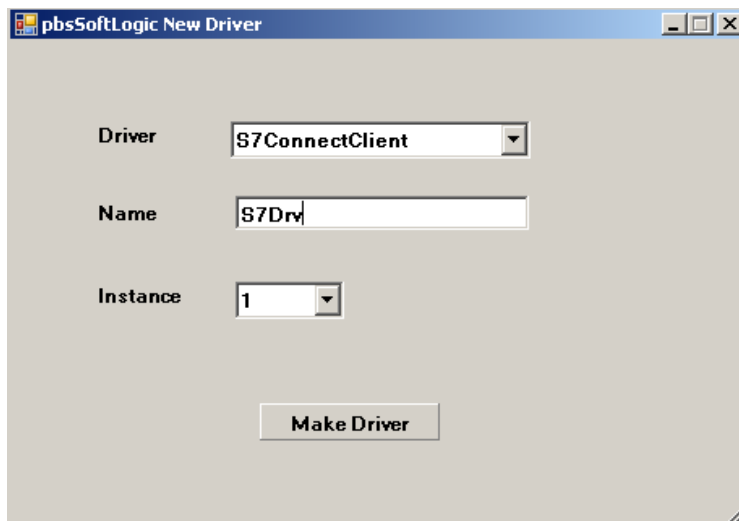
S7-communication works on TCP connection and your Siemens PLC should have IP address.

S7-Connect driver is not working with MAC ID of PLC.



Defining S7 Client in pbsSoftLogic :

- Define a new driver and select S7ConnectClient



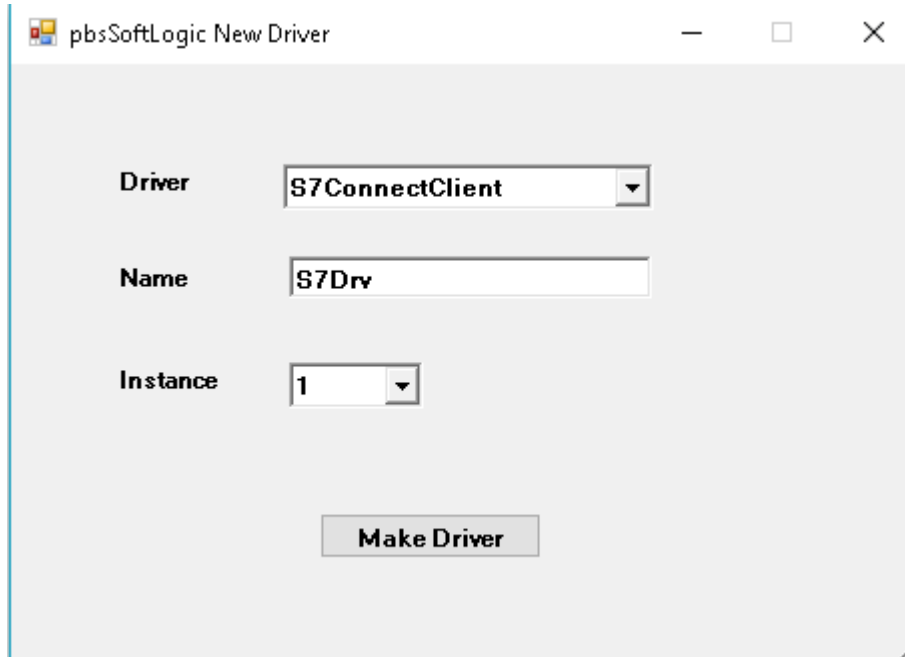
Select a unique name for driver and select Driver Instance.

If you want to connect to more than one S7 PLC, define a new S7ConnectClient for each PLC. Click on Make Driver, pbsSoftLogic will make a new directory (Driver name) and make default configuration file there.

You need to enable S7communication protocol on your PLC. pbsSoftLogic S7 Driver can read/write following type of information from S7 Series PLCs:

- Data Block
- Process Input / Output
- Memory

For defining a new S7 Driver for your project, in new Driver Page select s7ConnectClient Driver.



Select a unique name for driver and select Instance. When you have more than one PLC you need to define one S7Connect Driver for each PLC with different instance number.

You can connect maximum 8 PLC to pbsSoftLogic based RTU.

When you make a new driver, pbsSoftlogic will make default configuration files in your project.



Options.xml file:

```
<Options>
  <Node>
    <Name>PLCIP</Name>
    <Desc>PLC IP address</Desc>
    <Value>192.168.10.81</Value>
  </Node>
  <Node>
    <Name>Rack</Name>
    <Desc>PLC Rack</Desc>
    <Value>0</Value>
  </Node>
  <Node>
    <Name>Slot</Name>
    <Desc>PLC Slot</Desc>
    <Value>1</Value>
  </Node>
  <Node>
    <Name>BlockScanTime</Name>
    <Desc>BlockScanTime(ms) </Desc>
    <Value>50</Value>
  </Node>
  <Node>
    <Name>Instance</Name>
    <Desc>Instance</Desc>
    <Value>1</Value>
  </Node>
</Options>
```

PLCIP: IP address of S7 PLC

RACK, Slot: you need to find these numbers from your PLC Configuration application.

	Rack	Slot	
S7 300 CPU	0	2	Always
S7 400 CPU	Not fixed		Follow the hardware configuration.
WinAC CPU	Not fixed		Follow the hardware configuration.
S7 1200 CPU	0	0	Or 0, 1
S7 1500 CPU	0	0	Or 0, 1
CP 343	0	0	Or follow Hardware configuration.
CP 443	Not fixed		Follow the hardware configuration.
WinAC IE	0	0	Or follow Hardware configuration.

BlockScanTime: Scan time of reading/Writing to PLC In millisecond

S7Tags.xml file:

You need to define S7 Tags in this file. pbsSoftlogic will define for you all different supported blocks as sample for you when you define a new driver .

SYS block:

```
<Block Name="StatusBlock" Type="SYS" DBNum="0" Start="0" Channels="4">
  <S7Tags>
    <Tag Name="Online" Address="0" Init="0" />
    <Tag Name="ReadNum" Address="1" Init="0" />
    <Tag Name="WriteNum" Address="2" Init="0" />
    <Tag Name="ErrorNum" Address="3" Init="0" />
  </S7Tags>
</Block>
```

This block has 4 Tags: Online and ErrorNum is not used in driver.

ReadNum : Shows number of Read Operation from PLC (from 0 to 32000)

WriteNum: Shows number of Write Operation to PLC (from 0 to 32000)

You need to always check Read or Write Number In your logic. If for any reason RTU lost connection with PLC these number will not changed. And if you detect for example for 5 Sec these numbers are not changed, you will find that PLC connection is Offline and need to restart RTU.

S7Driver can only read/Write in byte format to S7 PLC.

You can define maximum 32 Blocks and in each block 64 Bytes can be read/Write. But total number of tags should not be more than 1024.

DBR Block:

```
<Block Name="Block1" Type="DBR" DBNum="1" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="DB1_0" Address="0" Init="0" />
    <Tag Name="DB1_1" Address="1" Init="0" />
    <Tag Name="DB1_2" Address="2" Init="0" />
    <Tag Name="DB1_3" Address="3" Init="0" />
    <Tag Name="DB1_4" Address="4" Init="0" />
    <Tag Name="DB1_5" Address="5" Init="0" />
    <Tag Name="DB1_6" Address="6" Init="0" />
    <Tag Name="DB1_7" Address="7" Init="0" />
  </S7Tags>
</Block>
```

DBR is using for reading S7 PLC Data Blocks.

DBNum = S7 Data Block Number

Start = Start Byte from Block

Channels: Number of Bytes to be read

In above sample RTU is reading 8 bytes from beginning of Block from DB Number 1.

DBW Block:

```
<Block Name="Block2" Type="DBW" DBNum="2" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="DB2_0" Address="0" Init="0" />
    <Tag Name="DB2_1" Address="1" Init="0" />
    <Tag Name="DB2_2" Address="2" Init="0" />
    <Tag Name="DB2_3" Address="3" Init="0" />
    <Tag Name="DB2_4" Address="4" Init="0" />
    <Tag Name="DB2_5" Address="5" Init="0" />
    <Tag Name="DB2_6" Address="6" Init="0" />
    <Tag Name="DB2_7" Address="7" Init="0" />
  </S7Tags>
</Block>
```

DBW Block is using for writing to S7 PLC Data blocks.

DBNum = S7 Data Block Number

Start = Start Byte from Block

Channels: Number of Bytes to be writes

In above sample RTU is Writing 8 bytes from beginning of Block from DB Number 2.

PI Block: process input

```

<Block Name="Block3" Type="PI" DBNum="0" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="PI_0" Address="0" Init="0" />
    <Tag Name="PI_1" Address="1" Init="0" />
    <Tag Name="PI_2" Address="2" Init="0" />
    <Tag Name="PI_3" Address="3" Init="0" />
    <Tag Name="PI_4" Address="4" Init="0" />
    <Tag Name="PI_5" Address="5" Init="0" />
    <Tag Name="PI_6" Address="6" Init="0" />
    <Tag Name="PI_7" Address="7" Init="0" />
  </S7Tags>
</Block>

```

PI Block will use for reading PLC Input Process Data. Suppose you map 32 digital Inputs and 4 Analog input channels in PLC Process Input Area from Address 10 to 22. Then you can read these channels by following PI block definition:

```

<Block Name="Block3" Type="PI" DBNum="0" Start="10" Channels="12">
  <S7Tags>
    <Tag Name="PI_0" Address="0" Init="0" />
    <Tag Name="PI_1" Address="1" Init="0" />
    <Tag Name="PI_2" Address="2" Init="0" />
    <Tag Name="PI_3" Address="3" Init="0" />
    <Tag Name="PI_4" Address="4" Init="0" />
    <Tag Name="PI_5" Address="5" Init="0" />
    <Tag Name="PI_6" Address="6" Init="0" />
    <Tag Name="PI_7" Address="7" Init="0" />
    <Tag Name="PI_8" Address="8" Init="0" />
    <Tag Name="PI_9" Address="9" Init="0" />
    <Tag Name="PI_10" Address="10" Init="0" />
    <Tag Name="PI_11" Address="11" Init="0" />
  </S7Tags>
</Block>

```

PI_0 , PI_1 , PI_2 and PI_3 has value of 32 digital input channel .

PI_4 /PI_5 = Analog input 1

PI_6 /PI_7 = Analog input 2

PI_8 /PI_9 = Analog input 3

PI_10 /PI_11 = Analog input 4

DBNum = Not used for PI Block.

POR, POW: POR = Read Process Output Status, POW = Write process output

```
<Block Name="Block4" Type="POR" DBNum="0" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="POR_0" Address="0" Init="0" />
    <Tag Name="POR_1" Address="1" Init="0" />
    <Tag Name="POR_2" Address="2" Init="0" />
    <Tag Name="POR_3" Address="3" Init="0" />
    <Tag Name="POR_4" Address="4" Init="0" />
    <Tag Name="POR_5" Address="5" Init="0" />
    <Tag Name="POR_6" Address="6" Init="0" />
    <Tag Name="POR_7" Address="7" Init="0" />
  </S7Tags>
</Block>
<Block Name="Block5" Type="POW" DBNum="0" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="POW_0" Address="0" Init="0" />
    <Tag Name="POW_1" Address="1" Init="0" />
    <Tag Name="POW_2" Address="2" Init="0" />
    <Tag Name="POW_3" Address="3" Init="0" />
    <Tag Name="POW_4" Address="4" Init="0" />
    <Tag Name="POW_5" Address="5" Init="0" />
    <Tag Name="POW_6" Address="6" Init="0" />
    <Tag Name="POW_7" Address="7" Init="0" />
  </S7Tags>
</Block>
```

POR Block will use for reading PLC Output Process Data.

POW Block will use for Writing PLC Output Process Data.

DBNum = Not used for POR, POW Blocks.

Start = Start Address of Process output

Channels: Number of Bytes to Write/Read

In above example RTU is reading Output process area of PLC from address 0 for 8 bytes by Block4 and will write on same area by Block5.

MBR , MBW Blocks : MBR = Read memory Area , MBW = Write memory Area

```
<Block Name="Block6" Type="MBR" DBNum="0" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="MBR_0" Address="0" Init="0" />
    <Tag Name="MBR_1" Address="1" Init="0" />
    <Tag Name="MBR_2" Address="2" Init="0" />
    <Tag Name="MBR_3" Address="3" Init="0" />
    <Tag Name="MBR_4" Address="4" Init="0" />
    <Tag Name="MBR_5" Address="5" Init="0" />
    <Tag Name="MBR_6" Address="6" Init="0" />
    <Tag Name="MBR_7" Address="7" Init="0" />
  </S7Tags>
</Block>
<Block Name="Block7" Type="MBW" DBNum="0" Start="0" Channels="8">
  <S7Tags>
    <Tag Name="MBW_0" Address="0" Init="0" />
    <Tag Name="MBW_1" Address="1" Init="0" />
    <Tag Name="MBW_2" Address="2" Init="0" />
    <Tag Name="MBW_3" Address="3" Init="0" />
    <Tag Name="MBW_4" Address="4" Init="0" />
    <Tag Name="MBW_5" Address="5" Init="0" />
    <Tag Name="MBW_6" Address="6" Init="0" />
    <Tag Name="MBW_7" Address="7" Init="0" />
  </S7Tags>
</Block>
```

MBR Block will use for reading PLC memory Area.

PBW Block will use for Writing PLC memory area.

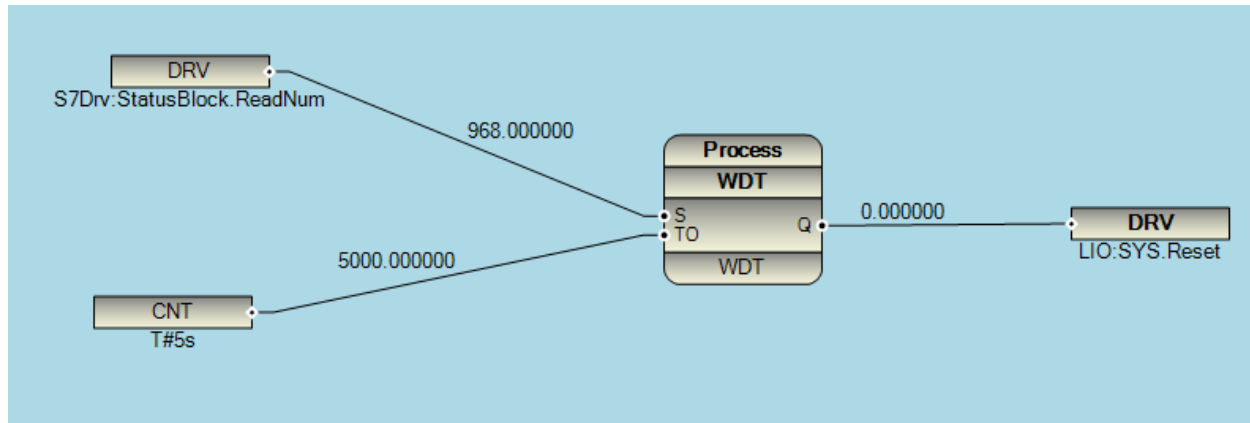
DBNum = Not used for POR, POW Blocks.

Start = Start Address of Memory area

Channels: Number of Bytes to Write/Read

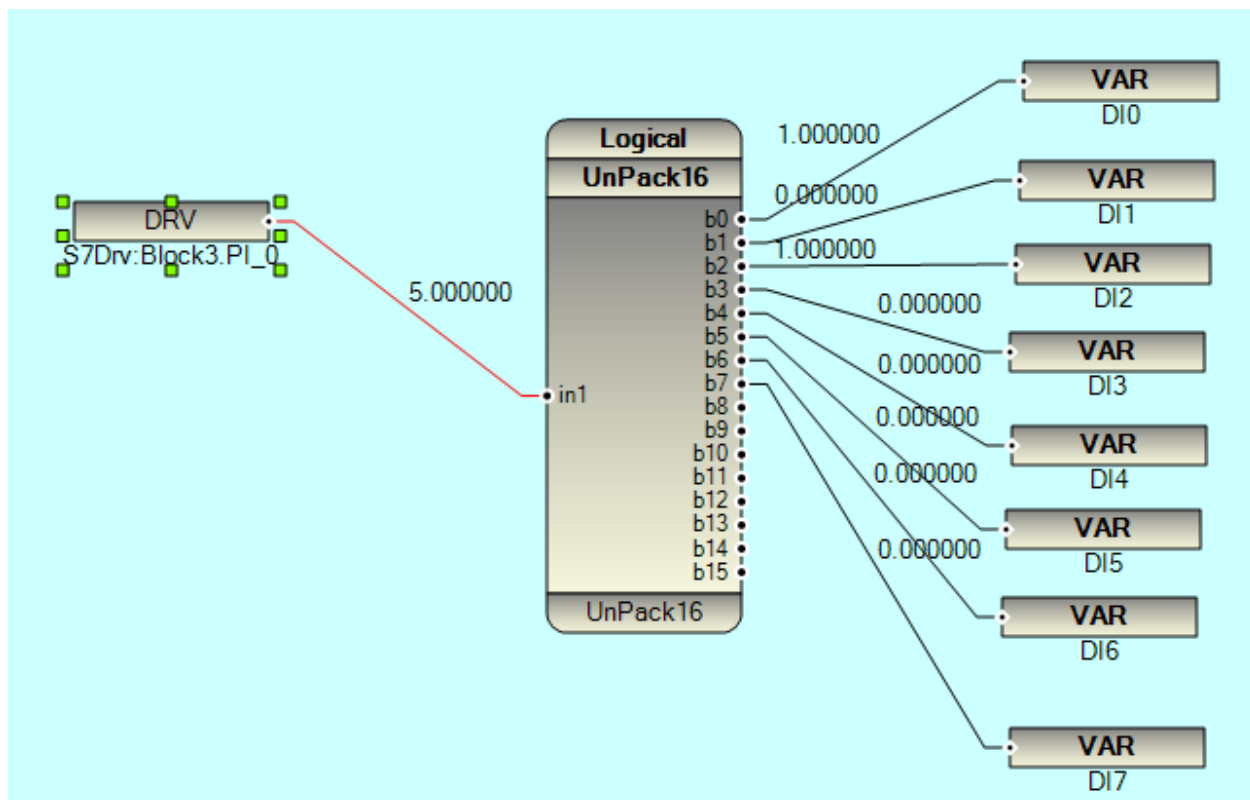
In above example RTU is reading Memory area of PLC from address 0 for 8 bytes by Block6 and will write on same area by Block7.

pbsSoftLogic is used Snap7 project for communication with S7PLC . Please refer to <http://snap7.sourceforge.net/>

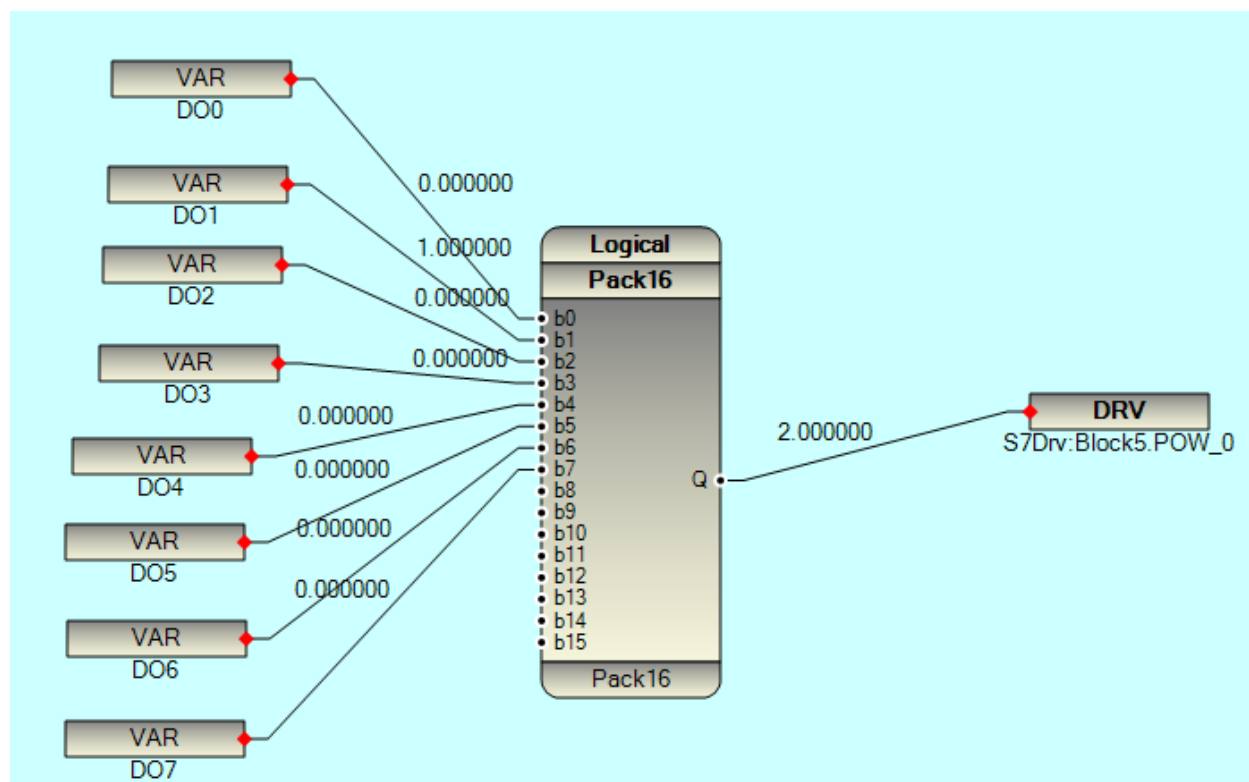


In above example RTU is checking PLC Connection status by watching ReadNum parameter. WDT function (in Process Group) it will watch S input, if S is changing in less than TO Sec, It will keep Q to 0 . But if S is not changed in TO sec, Q will change to 1.

In About Example WDT.Q Output is connected to RTU Reset Signal. It means that if RTU couldn't read Data from PLC in 5 Sec, RTU will restart Automatic for a new PLC connection.



In above example One byte of PI Block is converted to 8 Digital Input Signal (Internal Variable = VAR) to be used in Logic.

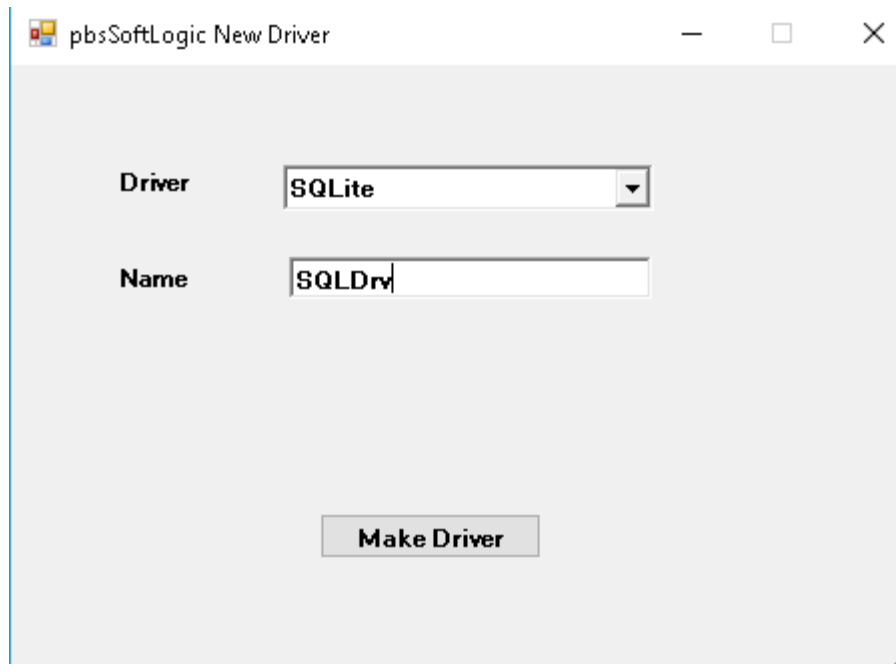


In above example 8 Internal Variable (DO0 to DO 7) is packed as one byte and written to POW_0 byte.

14 – SQLite Configuration, RTU local data Archiving and Automatic synchronization with MS SQL Server

pbsSoftLogic supports SQLite Driver for RTU local data archiving . For detail information about SQLite please refer to <https://www.sqlite.org/> you can synchronize SQLite in RTU with MS SQL Server in Control room automatically with help of this driver.

For adding SQLite to you project , use SQLite Driver to add to project Driver list .



You can define one instance of SQLite driver to project.

In pbsSoftLogic Directory you can see sqlite directory with following contents :

> This PC > Windows7_OS (C:) > pbsControl > PSLE > Sqlite	
Name	Date modified
sqliteprj.sqbpro	12/1/2015 9:37 AM
sqllog.db	11/29/2015 5:26 PM
SQLQuery6.sql	12/18/2015 3:53 AM

You need to transfer sqllog.db file to RTU by Filezilla. You can copy it in SD card of RTU .

For ADAM-3600 RTU External SD card path is /media/mmcbk1p1

When you make driver, pbsSoftLogic will make default configuration for you in driver directory.



options.xml



SQLiteTags.xml

Options.xml file:

```

<Node>
  <Name>SQLSyncUrl</Name>
  <Desc>Web Service Path in Master SCADA for synchronization of Sqlite and RDBMS in Master SCADA</Desc>
  <Value>/>
</Node>
<Node>
  <Name>RTUName</Name>
  <Desc>UniqueRTUName</Desc>
  <Value>RTU1</Value>
</Node>
<Node>
  <Name>OfflinePath</Name>
  <Desc>Save Data in this path when couldnt synchronized with master SCADA </Desc>
  <Value>/home/sqlsynclog/</Value>
</Node>
<Node>
  <Name>OfflinePath2</Name>
  <Desc>Save Data in this path when couldnt synchronized with master SCADA2 </Desc>
  <Value>/home/sqlsynclog2/</Value>
</Node>
<Node>
  <Name>LogCyclicRtMin</Name>
  <Desc>Log Data exactly at each 1 Min</Desc>
  <Value>1</Value>
</Node>
<Node>
  <Name>LogChanges</Name>
  <Desc>Log Signal when its value changes</Desc>
  <Value>0</Value>
</Node>
<Node>
  <Name>SQLDatabase</Name>
  <Desc>Path and Name of SQLite Database</Desc>
  <Value>/home/sqlldb/sqllog.db</Value>
</Node>
...

<Node>
  <Name>SQLServerName</Name>
  <Desc>Name or IP address of SQL server in master scada for direct insertion of data </Desc>
  <Value>192.168.1.100</Value>
</Node>
<Node>
  <Name>SQLServerName2</Name>
  <Desc>Name or IP address of Second SQL server in master scada2 for direct insertion of data </Desc>
  <Value>192.168.1.101</Value>
</Node>
<Node>
  <Name>SQLServerUser</Name>
  <Desc>User Name for SQL Server in master scada for direct insertion of data </Desc>
  <Value>sqlite</Value>
</Node>
<Node>
  <Name>SQLServerPassword</Name>
  <Desc>Password for SQL Server in master scada for direct insertion of data </Desc>
  <Value>psle</Value>
</Node>
<Node>
  <Name>SQLServerDBName</Name>
  <Desc>Database Name in SQL Server in master scada for direct insertion of data </Desc>
  <Value>pbsHMI</Value>
</Node>
<Node>
  <Name>SQLiteRTULogEnable</Name>
  <Desc>1 = Data Will log to SQLite File inside RTU , 0 = Data is not log to SQLite inside RTU and only will send data to Master SCADA </Desc>
  <Value>1</Value>
</Node>

```

SQLSyncUrl : you can synchronize data by Master SCADA Database by two solution :

- Using Web Service to use any type of DBMS in Control room
- Direct Synchronization with MS SQL Server

When SQLSyncUrl is blank , Web service Method is disabled .

RTUName : Each RTU in the Network should has unique Name for archiving data in control room . RTU name is also used to archive data in Control Room Database directly in pbsHMI database format.

When you are using pbsHMI in Control room , each tag has a prefix in pbsHMI . Suppose your tag name in RTU is Tag1 and in pbsHMI with modbus TCP driver its name is ModbusTags.Device1.Tag1 . Then you need to use ModbusTags.Device1 as RTUName option . When SQLite driver is sending data to Control room , it will use {RTUName}+"."+{TagName } as signal name . So pbsHMI can easily use data that is send by SQLite driver for backfilling and reporting . It is not affect Instant data value in pbsHMI because SQLite Driver is directly archive data in Database of pbsHMI .

OfflinePath : when RTU is offline and there is no communication with master SCADA to synchronize data between RTU and SQL Server in Control room , RTU will archive Data in this Path . To be sure that Path is exist in RTU . Suppose you will set this path to /home/data then data directory must be in home directory for proper operation of driver.

OfflinePath2 : SQLite driver can send data to two master SCADA in the same time . This Path is for second Control room. Operation is same as OfflinePath

LogCyclicAtMin : Will log data exactly at every LogCyclicAtMin . Suppose you set this value to 5 then SQLite driver will log data at xx:00 , xx:05 , xx:10 , xx:15,xx:20,xx:25,xx:30,xx:35,xx:40,xx:45,xx:50,xx:55,...

This data logging is not related to RTU is offline with master SCADA or it is online.

SQLDatabase : full path of sqlite database . Like /media/mmcblk1p1/sqllog.db

Logchanges : not used .

SQLServerName : IP address of SQL Server in Control room .

SQLServerName2 : IP address of second SQL Server in Control room . If blank disable.

SQLServerUser : User name of MS SQL server in control room .


SQLServerPassword: Password of MS SQL server in control room.

Please notice that If you are using two MS SQL Server to synchronize data , both should has same user name and password for proper operation of SQLite driver .

SQLServerDBName : Database name in SQL Server . you need to use pbsHMI database in control room .

SQLiteRTULogEnable : if set to 1 , Data will archive in RTU in SQLite format . If set to 0 , Data is only send to SQL Server in Control room and SQLite is not used in RTU . But Backfilling is working without problem in this Mode.

You can use pbsSoftLogic SQLite Editor to set all above parameters . for using SQLite Editor , in Driver List , Right Click and select Edit .

 C:\PIP2012\pbsSoftLogic\PSLE\VSLE\SqITest\SqITest\SQL1

SQLite Configuration Path
C:\PIP2012\pbsSoftLogic\PSLE\VSLE\SqITest\SqITest\SQL1

Params **Tags**

☒ SQLite RTU Logging Enable LogCyclic 1 min

SQLSyncUrl

RTUName RTU1

SQLite Database Path and Name /home/sqlldb/sqllog.db

MS SQL Server Options

MS SQLServerDBName pbsHMI

MS SQLServerUser sqlite

MS SQLServerPassword psle

MS SQL Multi Server Options

OfflinePath /home/sqlsynclog/ OfflinePath2 /home/sqlsynclog2/

SQLServerIP 192.168.1.100 SQLServerIP2 192.168.1.101

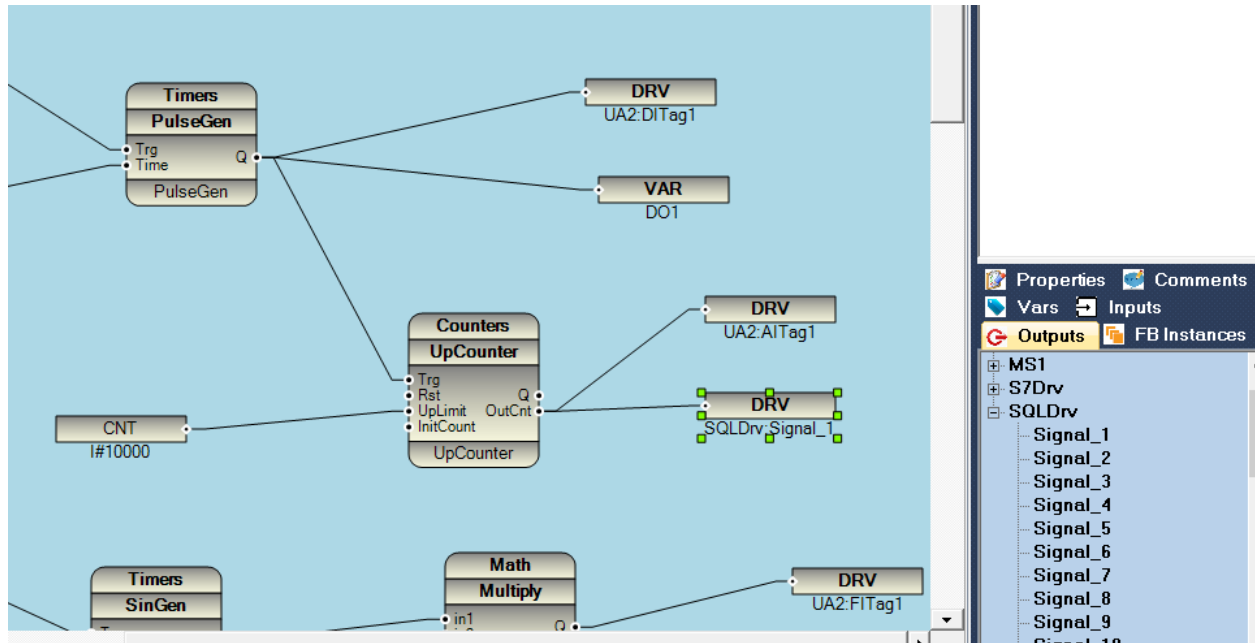
Instance 1

For saving configuration , right click on any point in SQLite Editor form and select Save menu .

SQLiteTags.xml: you can define Tags in this file. Only you need to define tag name. You can define maximum 1024 Tags for SQLite Driver.

```
<OPCSrvTags>  
  <Version>1.0.0</Version>  
  <Tag Name="Signal_1" />  
  <Tag Name="Signal_2" />  
  <Tag Name="Signal_3" />  
  <Tag Name="Signal_4" />  
  <Tag Name="Signal_5" />  
  <Tag Name="Signal_6" />  
  <Tag Name="Signal_7" />  
  <Tag Name="Signal_8" />  
  <Tag Name="Signal_9" />  
  <Tag Name="Signal_10" />  
  <Tag Name="Signal_11" />  
  <Tag Name="Signal_12" />  
  <Tag Name="Signal_13" />  
  <Tag Name="Signal_14" />  
  <Tag Name="Signal_15" />  
  <Tag Name="Signal_16" />  
  <Tag Name="Signal_17" />  
  <Tag Name="Signal_18" />  
  <Tag Name="Signal_19" />  
  <Tag Name="Signal_20" />  
  <Tag Name="Signal_21" />  
  <Tag Name="Signal_22" />  
  <Tag Name="Signal_23" />  
  <Tag Name="Signal_24" />  
  <Tag Name="Signal_25" />  
  <Tag Name="Signal_26" />  
  <Tag Name="Signal_27" />  
  <Tag Name="Signal_28" />  
  <Tag Name="Signal_29" />  
  <Tag Name="Signal_30" />  
  <Tag Name="Signal_31" />  
  <Tag Name="Signal_32" />  
</OPCSrvTags>
```

You should write all above signals in your logic. Like following sample:



At each cycle, logic will write Sqlite Signals value to Driver. Driver has internal memory for save all SQLite signals. When it is time for logging, driver will read all internal memory and save them on the SQLite database with current time value for all signals. SQLite driver is not buffering all signal changes and only will save signals vale at logging time.

There are many free SQLite browser and management utility in the market.

You can use following as sample:

SQLite DB Browser : <http://sqlitebrowser.org/>

SQLite Studio : <http://sqlitestudio.pl/>

SQLite Expert : <http://www.sqliteexpert.com/>

Database structure:

Sqllog.db has two main tables:

- TagIndex
- TagData

TagIndex Table Fields:

Index	Name	Declared Type	Type	Size	Precision	Not Null
1	ID	INTEGER	INTEGER	0	0	<input checked="" type="checkbox"/>
2	TagName	TEXT	TEXT	0	0	<input checked="" type="checkbox"/>

TagData Table Fields:

Index	Name	Declared Type	Type	Size	Precision	Not Null
1	ID	INTEGER	INTEGER	0	0	<input checked="" type="checkbox"/>
2	TagDT	NUMERIC	NUMERIC	0	0	<input checked="" type="checkbox"/>
3	TagValue	REAL	REAL	0	0	<input checked="" type="checkbox"/>

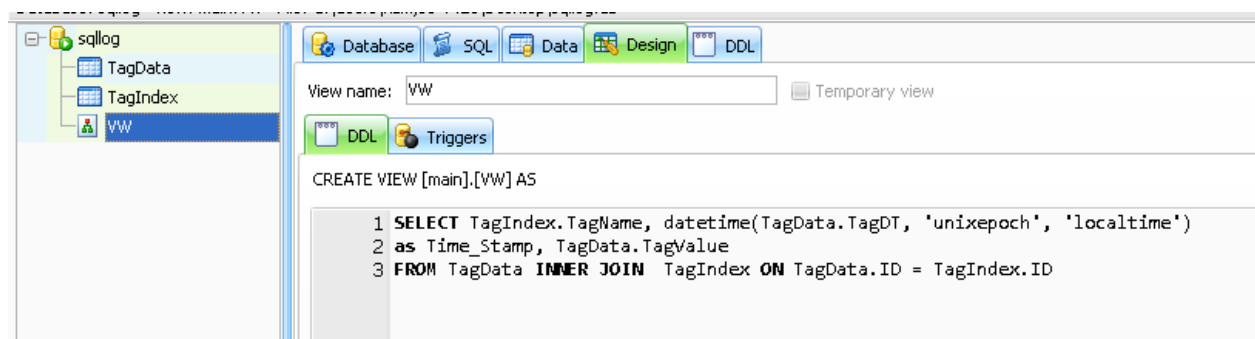
When Driver wants to insert one record to database , it will do following tasks :

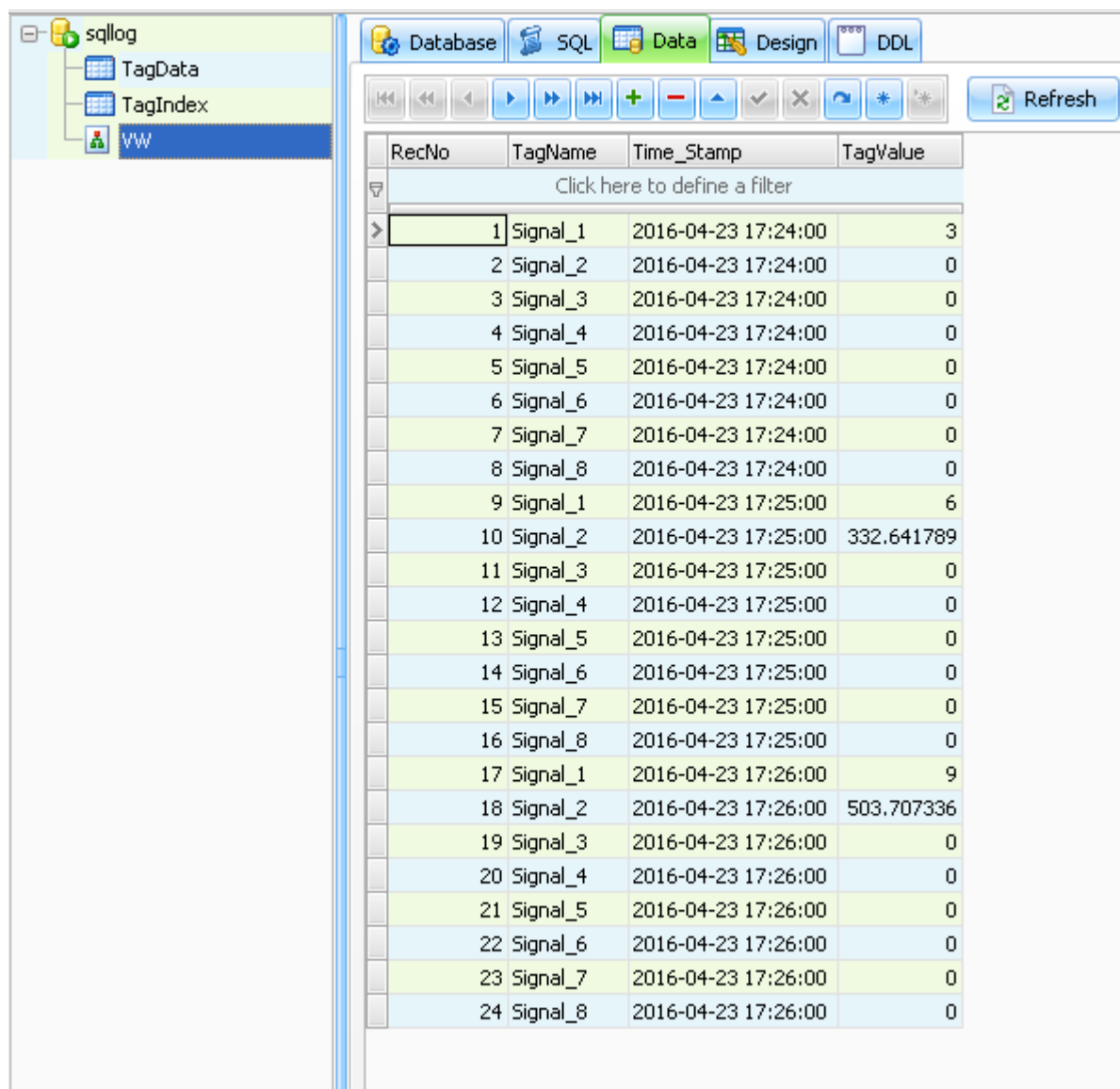
- Check in TagIndex Table , if Tag name is Inserted before it will get Tag ID
- If Tag Name is not inserted into TagIndex , It will insert TagName and will get new TagID
- With TagID , TagValue and TagDT(Data Time) , Driver will insert one record into TagData Table

In other words, TagIndex is Definition of tags (record number is equal to Number of Tags) and TagData is real data archiving in database. TagData will increase based on number of Records in the database.

TagDT is Elapsed Time which is seconds from 1/1/1970.

You can use following view to see readable view from database: (VW view is already in the sqllog.db)





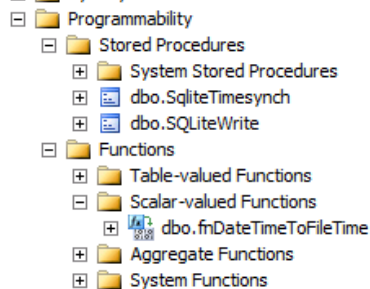
The screenshot shows a SQLite browser window with a tree view on the left containing 'sqllog', 'TagData', 'TagIndex', and 'VW'. The main pane displays a table with columns 'RecNo', 'TagName', 'Time_Stamp', and 'TagValue'. The table contains 24 rows of data, with the first row selected. A toolbar at the top includes buttons for Database, SQL, Data, Design, DDL, and a Refresh button.

RecNo	TagName	Time_Stamp	TagValue
1	Signal_1	2016-04-23 17:24:00	3
2	Signal_2	2016-04-23 17:24:00	0
3	Signal_3	2016-04-23 17:24:00	0
4	Signal_4	2016-04-23 17:24:00	0
5	Signal_5	2016-04-23 17:24:00	0
6	Signal_6	2016-04-23 17:24:00	0
7	Signal_7	2016-04-23 17:24:00	0
8	Signal_8	2016-04-23 17:24:00	0
9	Signal_1	2016-04-23 17:25:00	6
10	Signal_2	2016-04-23 17:25:00	332.641789
11	Signal_3	2016-04-23 17:25:00	0
12	Signal_4	2016-04-23 17:25:00	0
13	Signal_5	2016-04-23 17:25:00	0
14	Signal_6	2016-04-23 17:25:00	0
15	Signal_7	2016-04-23 17:25:00	0
16	Signal_8	2016-04-23 17:25:00	0
17	Signal_1	2016-04-23 17:26:00	9
18	Signal_2	2016-04-23 17:26:00	503.707336
19	Signal_3	2016-04-23 17:26:00	0
20	Signal_4	2016-04-23 17:26:00	0
21	Signal_5	2016-04-23 17:26:00	0
22	Signal_6	2016-04-23 17:26:00	0
23	Signal_7	2016-04-23 17:26:00	0
24	Signal_8	2016-04-23 17:26:00	0

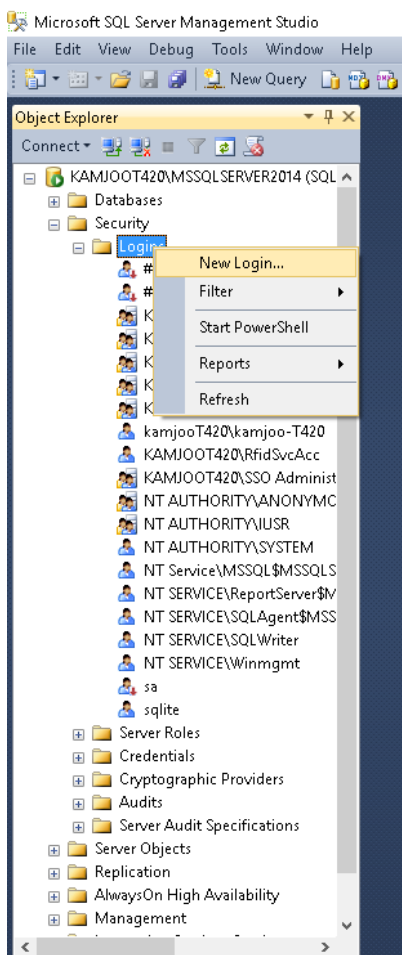
You can easily transfer sqllog.db to your server in control room and open it by any SQLite browser and analysis stored data.

MS SQL Server setting : you need to define pbsHMI Database in SQL Server at control room and define one User with password to use in SQLite Driver . Please do following steps :

- 1- Make a new database in SQL Server ,name it pbsHMI.
- 2- Use SQL Script that is in SQLite Directory of pbsSoftLogic or use Database directory of pbsHMI
- 3- Run SQLServerData.sql and SQLServerIndex.sql to make data and index tables in pbsHMI Database
- 4- Run fnDateTimeToFileTime.sql , SqliteTimesynch.sql and SQLiteWrite.sql to make all necessary Stored procure and Scalar functions in pbsHMI Database

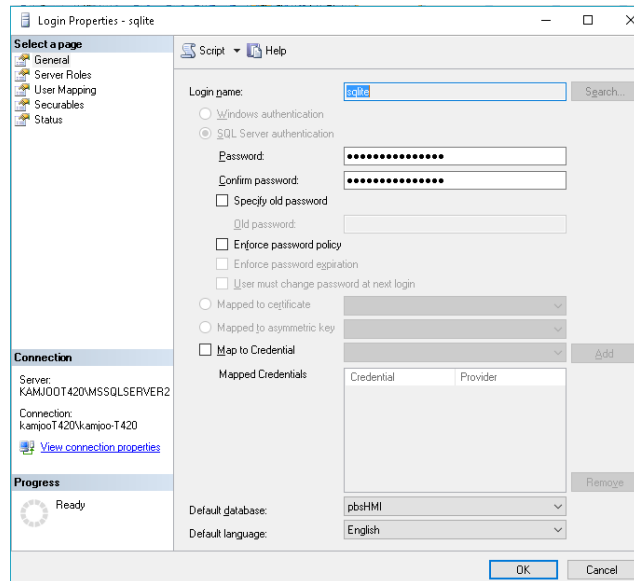


- 5- In SQL Server Management studio , Use "Security" item and open "Login" segment .
- 6- Right click in Login and select New Login

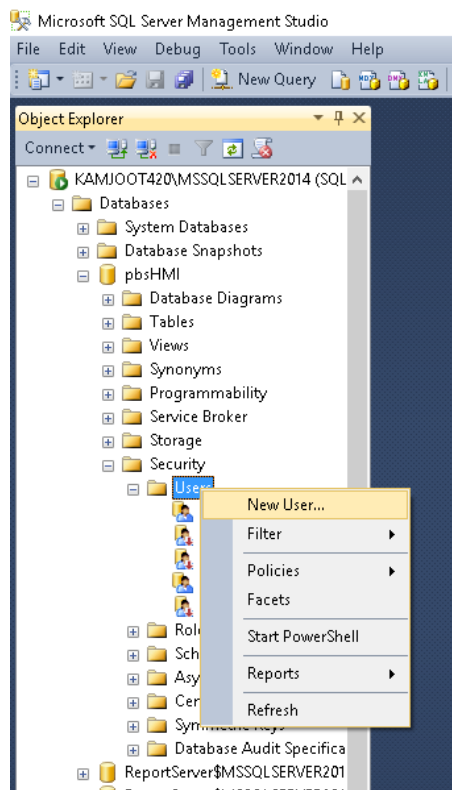


6 – In Login Properties page type user name for example “sqlite” and set password .Remove pass policy , expiration and user should change password in next login . you will use this user and password in SQLite driver options .

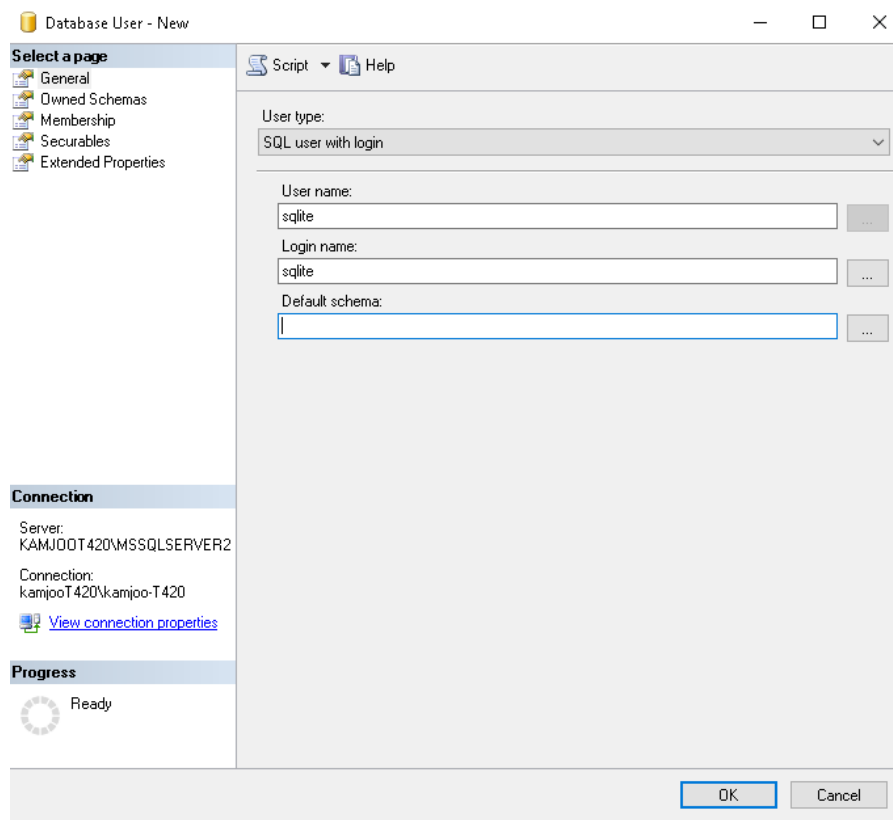
7 – select pbsHMI as default database .



8 – select pbsHMI Database , Select Security and open Users and right click on user to make new user.

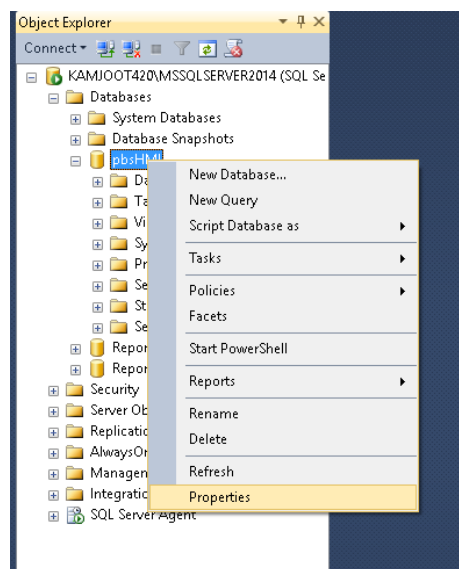


9 –use same user that you make for SQL server “sqlite” and redefine it here.

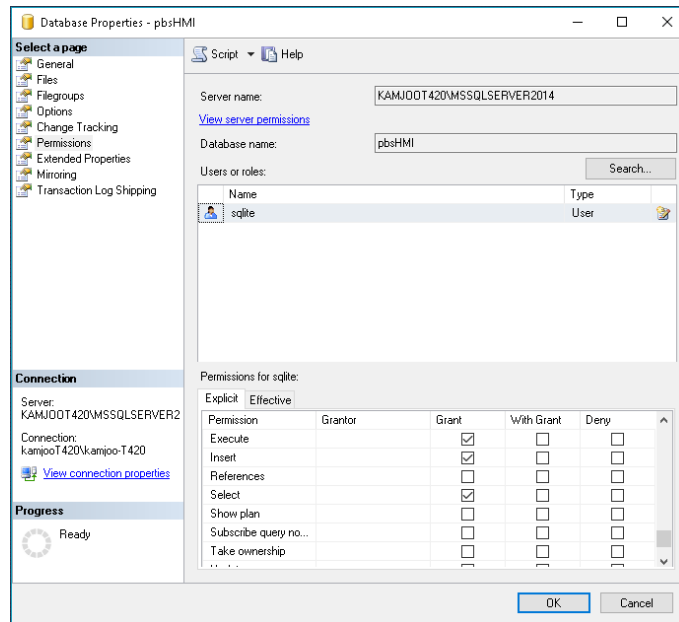


10 – Click on ok to define user .

11 – Select properties of pbsHMI Database

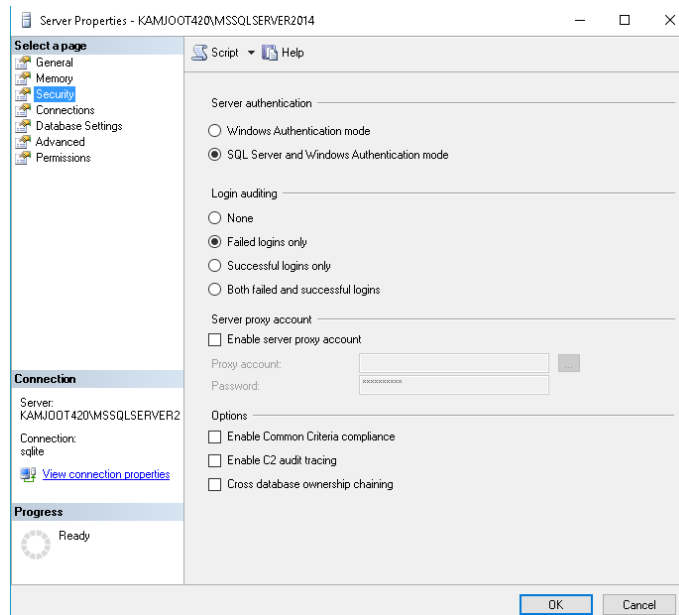


12 – select permissions Page and Grant Connect , Execute , Insert , Update , Delete and select functionality to sqlite user . In this stage SQL Server is ready for proper operation with SQLite driver .



13 – To be sure pbsHMI Database collation is not Arabic , Persian , ... and only Latin like SQL_Latin1_General_CP1_CI_AS will communicate with RTU

14 – To be Sure SQL Server Authentication is SQL Server and Windows

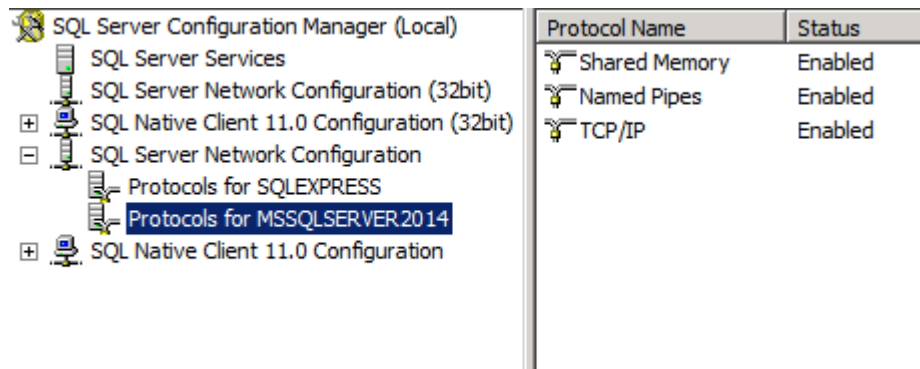


15 – Check in Windows Firewall. TCP Port 1433 should be allowed in both inbound and outbound rules.

16 – Check in Windows Firewall. SQLBrowser.exe utility should be in Allowed programs List .you can find SQLBrowser.exe Path from C:\Program Files (x86)\Microsoft SQL Server\90\Shared\sqlbrowser.exe

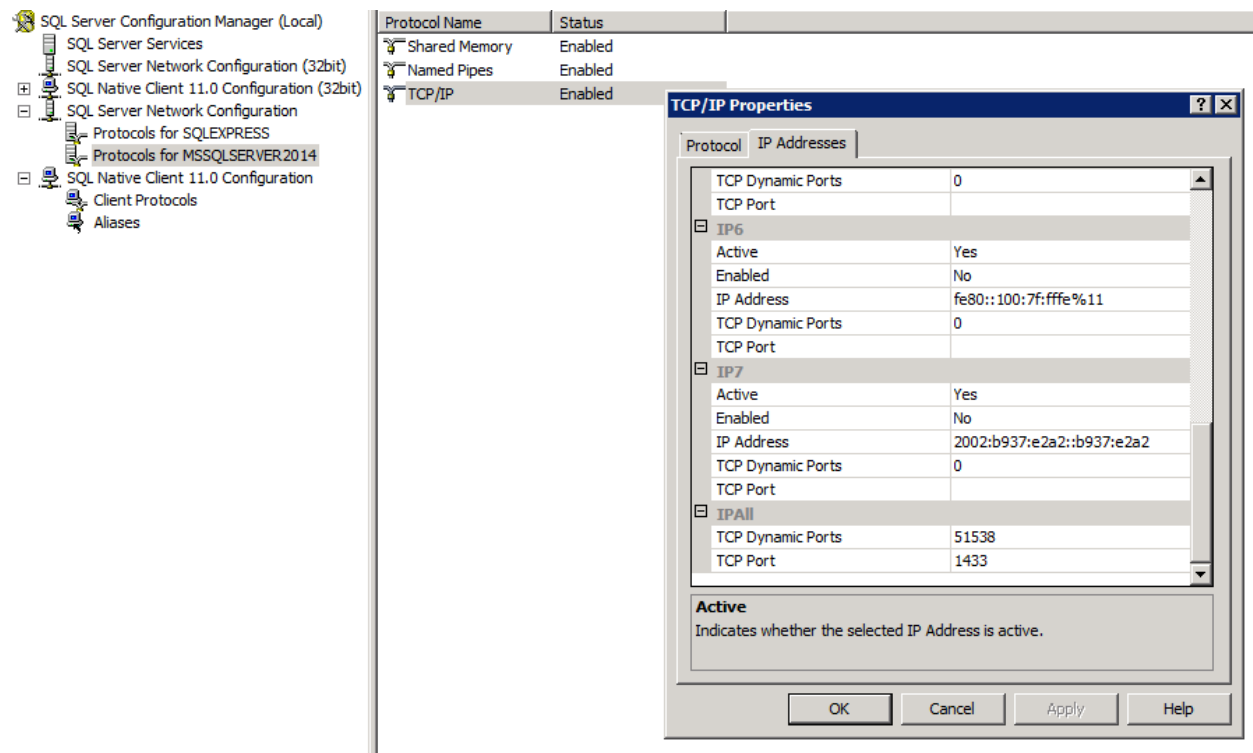
To be sure SQL Server and SQL Server Browser Services are started properly.

17 – Open SQL Server Configuration Utility and open SQLServer Network Configuration.



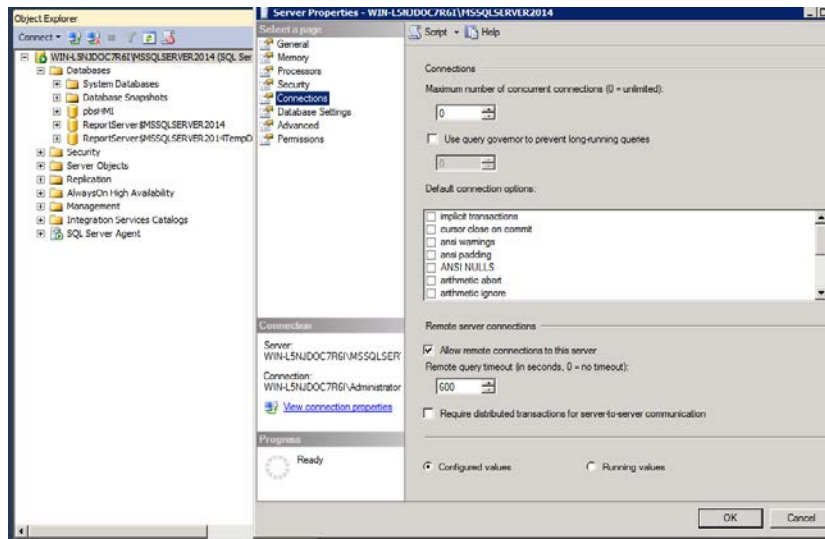
To be sure Named Pipes and TCP/IP protocols are enabled. You can Make Them Enable by Right Click on each Item and select Enable Option.

Double click on TCP/IP Protocol and select IP Tab.



Scroll Down to IPALL and write TCP Port 1433.

18 – Open property page of main SQL server instance and select Connections.



To be sure Allow Remote Connections to this server is checked.

19 – Restart SQL Server Service.

20 – In RTU edit /etc/freetds/freetds.conf file and find global part setting and do following changes :

```
# Global settings are overridden by those in a database
# server specific section
[global]
    # TDS protocol version
    port = 1433
    tds version = 7.0
```

20 – To be sure that Offline Path, Offline Path2 and SQLite Database are existing inside RTU.

SQLite Driver with SQL Server Automatic synchronization is only work on AMS-R3010 RTU , pbs2010GW ,pbs2012GW , ADAM-3600 and UNO1252G .

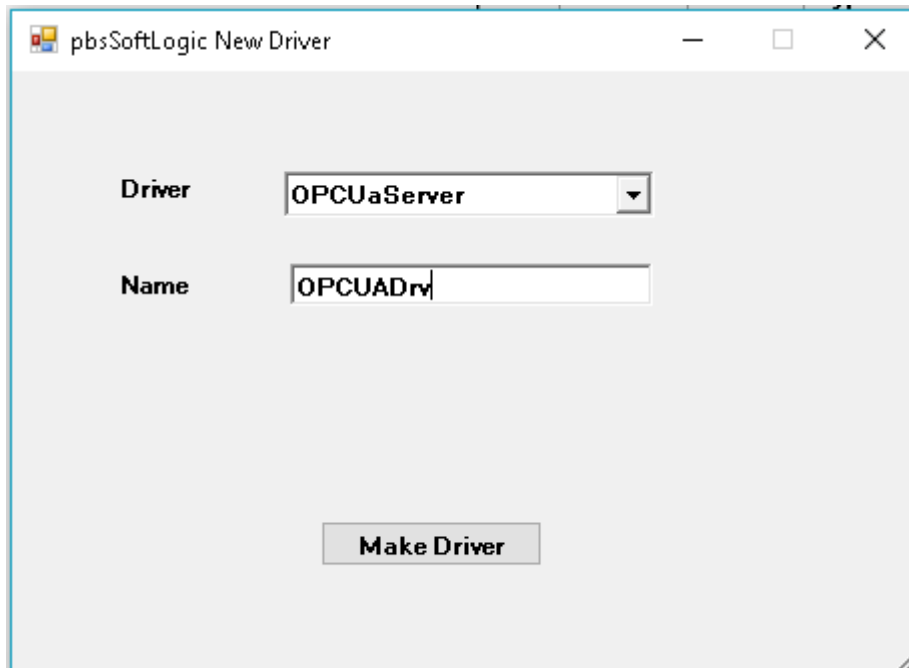
15 – OPC UA Server configuration for RTU

pbsSoftLogic is supporting OPC UA Server functionality for ADAM-3600 and AMS-R3010 RTU .

pbsSoftLogic is using OpenOPCUa (<http://www.openopcua.org>) technology for implementing OPC UA functionality .

For detail information about OPC-UA please refer to www.opcfoundation.org






For adding OPC UA Server driver to your project from new driver list page, select OPCUaServer Driver:



Select a unique name for your driver. You can only define one OPC UA Driver for a project.

Click on Make Driver button, pbsSoftLogic will make default configuration in driver directory.

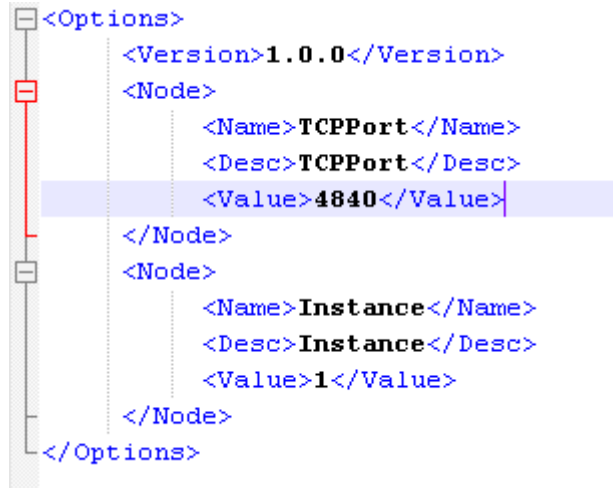
In Driver directory you can see following files:

-  ConfigOpenOpcUa.xml
-  OPCUaTags.xml
-  options.xml
-  pslemodel.xml
-  pslemodelsb.xml

You need to do configuration for options.xml and OPCUaTags.xml files. Other files are systematic and generated by pbsSoftLogic when you save project settings.

OpenOPCUaCore Server should install on your RTU and running automatic at boot time for proper operation of OPC UA Driver.

Options.xml file :



```
<Options>
  <Version>1.0.0</Version>
  <Node>
    <Name>TCPPort</Name>
    <Desc>TCPPort</Desc>
    <Value>4840</Value>
  </Node>
  <Node>
    <Name>Instance</Name>
    <Desc>Instance</Desc>
    <Value>1</Value>
  </Node>
</Options>
```

TCPPort = you need to assign not used TCP port for OPC UA Communication. OPCOpCua System is using TCP for communication in physical layer. The IANA registered port for "OPC UA /TCP" is 4840.

OPCUaTags.xml file

OPCUA Tags are defined in this file. In following sample file you can see different tag types. You can define maximum 1024 OPC UA Tags.

There are following Tag Types:

DI = Digital input = Boolean (Read by Client)

DO = Digital Output = Boolean (Read/Write by Client)

AI = Analog Input (2 Bytes Signed - Read by Client)

UAI = unsigned Analog Input (2 Bytes unsigned - Read by Client)

AO = Analog Output (2 Bytes Signed – Read/Write by Client)

FI = Float Input (4 Bytes Signed - Read by Client)

FO = Float Output (4 Bytes Signed – Read/Write by Client)

LI = Long Input (4 Bytes Signed - Read by Client)

ULI = unsigned Long Input (4 Bytes unsigned - Read by Client)

DBI = Double Input (8 Bytes Signed - Read by Client)

DBO = Double Output (8 Bytes Signed – Read/Write by Client)

```

]<OPCSrvTags>
  <Version>1.0.0</Version>
  <Tag Name="DITag1" Type="DI" Init="False" />
  <Tag Name="DITag2" Type="DI" Init="False" />

  <Tag Name="DOTag1" Type="DO" Init="False" />
  <Tag Name="DOTag2" Type="DO" Init="False" />

  <Tag Name="AITag1" Type="AI" Init="0" />
  <Tag Name="AITag2" Type="AI" Init="0" />

  <Tag Name="UAITag1" Type="UAI" Init="0" />
  <Tag Name="UAITag2" Type="UAI" Init="0" />

  <Tag Name="AOTag1" Type="AO" Init="0" />
  <Tag Name="AOTag2" Type="AO" Init="0" />

  <Tag Name="FITag1" Type="FI" Init="0" />
  <Tag Name="FITag2" Type="FI" Init="0" />

  <Tag Name="FOTag1" Type="FO" Init="0" />
  <Tag Name="FOTag2" Type="FO" Init="0" />

  <Tag Name="LITag1" Type="LI" Init="0" />
  <Tag Name="LITag2" Type="LI" Init="0" />

  <Tag Name="ULITag1" Type="ULI" Init="0" />
  <Tag Name="ULITag2" Type="ULI" Init="0" />

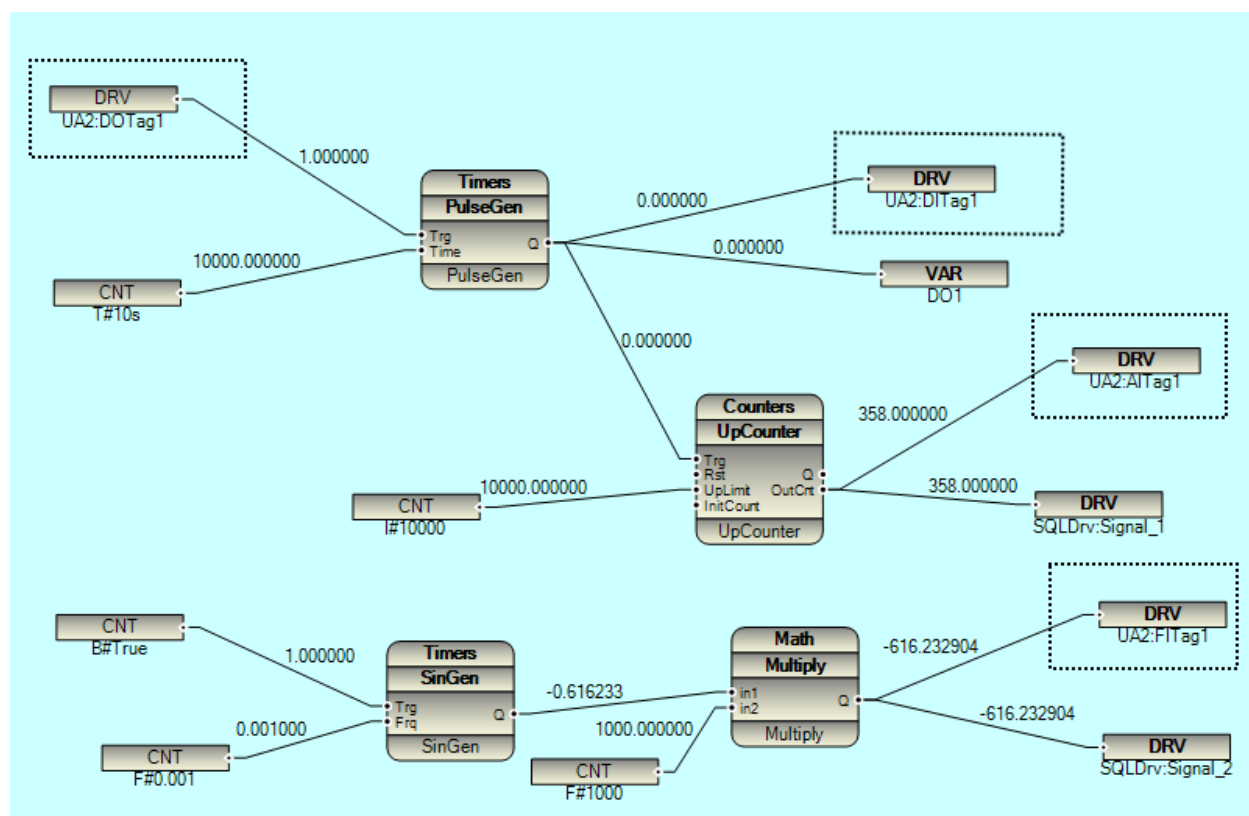
  <Tag Name="DBITag1" Type="DBI" Init="0" />
  <Tag Name="DBITag2" Type="DBI" Init="0" />

  <Tag Name="DBOTag1" Type="DBO" Init="0" />
  <Tag Name="DBOTag2" Type="DBO" Init="0" />
-</OPCSrvTags>

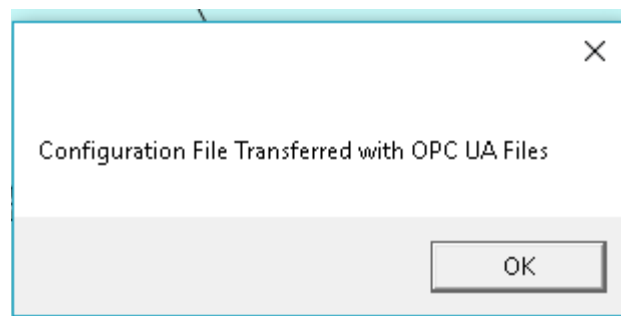
```

You need to write Input Tags (DI , AI , UAI , LI , ULI , FI ,DBI) In logic and read output Tags (DO , AO , FO , DBO)

In following sample, UA2:DOTag1 is connected to FB Input Side (Read) and UA2:DITag1, UA2:AITag1, UA2:FITag1 are connected to FB output side (Write)



When you transfer configuration to RTU, you will see following message that shows OPC UA Systematic files are also transferred to RTU.



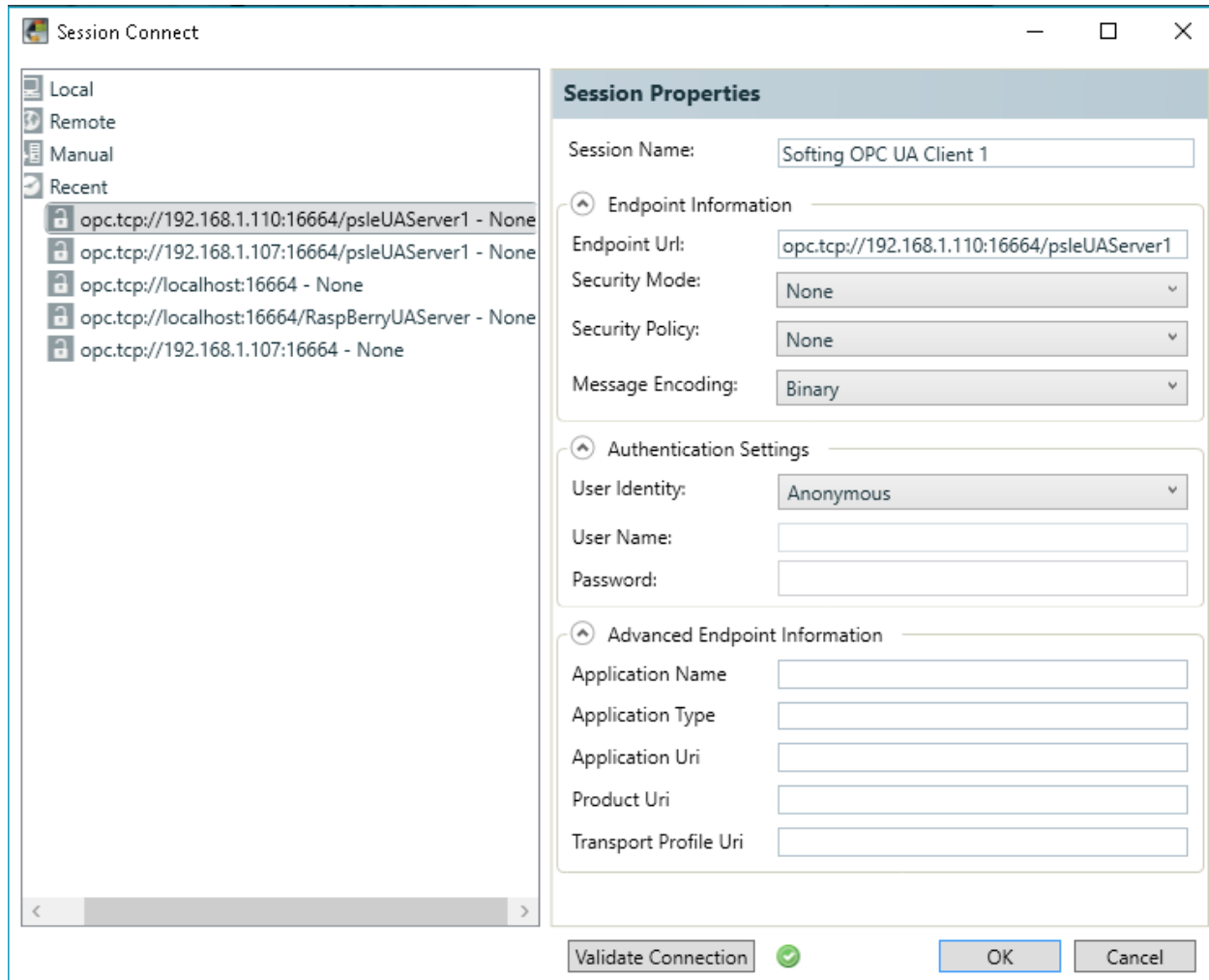
Reset RTU when you did modification in OPC UA Tags and configuration .

OPC UA Client Connection:

You can use any OPC UA Browser utility to connect to RTU by OPC UA protocol.

Softing OPC UA client: It is a free OPC UA Browser that you can download from www.softing.com

Create a new session with following parameters:



End Point URL has following format:

opc.tcp://{RTU IP address} :{ OPC UA TCP Port}/psleUAServer1

Example:

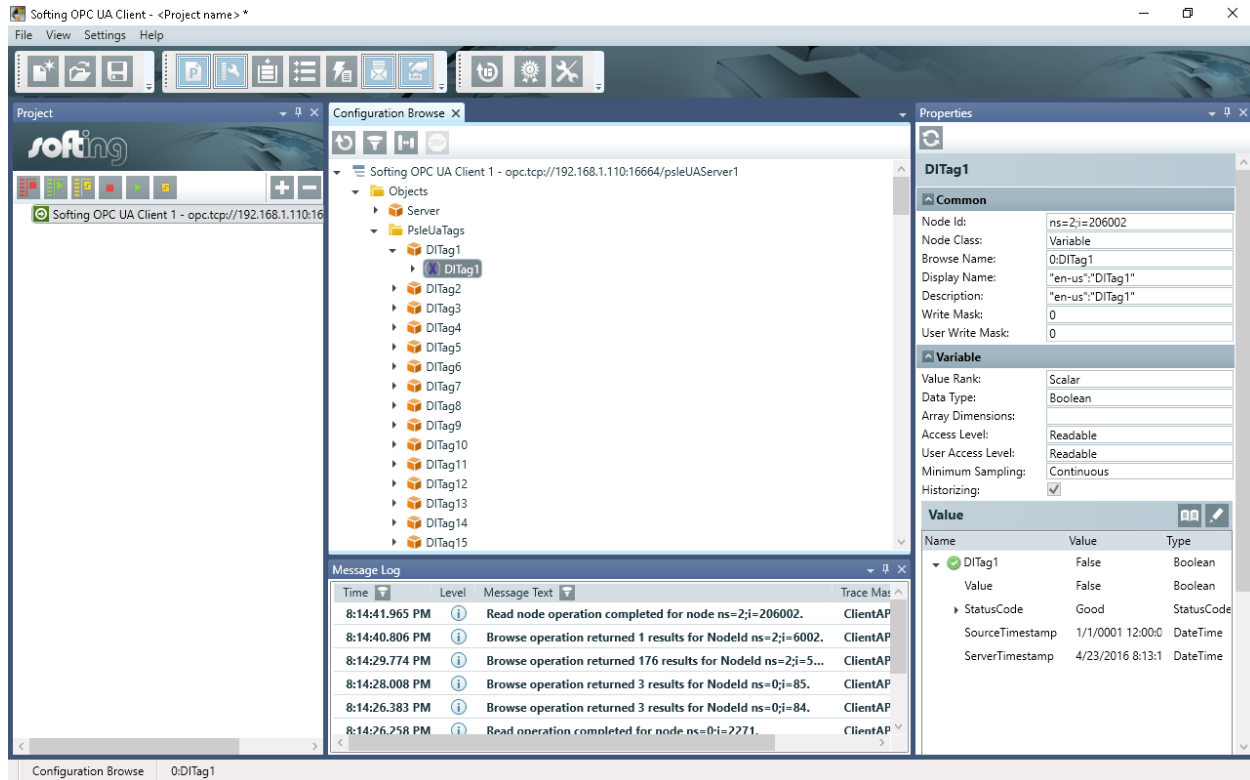
opc.tcp://192.168.1.110:16664/psleUAServer1

OPC UA supports different modes for security. OpenOPCUA supports all modes, but in pbsSoftLogic we only support Anonymous (No Security)

OPC UA TCP Port should be defined before for Network and firewall software's for proper communication between client and RTU.

Click on Validate connection, it should be changed to green check mark for correct validation.

Click on OK Button, you should see tag Data types, Objects, as following page:



pbsSoftLogic Tags will be found in Objects\PsleUaTags folder .

By double clicking on a tag, it will be included to subscribe tags, so you can get automatic tags changes in OPC UA Client application.

The screenshot displays the Softing OPC UA Client application interface. The main window is titled "Softing OPC UA Client - <Project name> *". It features a menu bar (File, View, Settings, Help) and a toolbar with various icons. The interface is divided into several panes:

- Project Pane:** Shows the project structure, including "Subscription 1" and a tree view of nodes: "Root\Objects\PseUaTags\FITag1", "Root\Objects\PseUaTags\AITag1", and "Root\Objects\PseUaTags\DOTag1".
- Configuration Browse Pane:** Displays a table of monitored items.

State	Display Name	Node Id	Data Type	Value	Server Timestamp	Source
...	...gs\FITag1\FITag1	ns=2;i=206082	Single	997.049	8:19:36.650 PM	12:00:0
...	...s\AITag1\AITag1	ns=2;i=206034	Int16	1	8:19:31.648 PM	12:00:0
...	...DOTag1\DOTag1	ns=2;i=206018	Boolean	True	8:19:28.645 PM	12:00:0
- Properties Pane:** Shows the "Monitored Items Properties" for the selected item.
 - Display Name: Root\Objects\PseUaTags\DOTag1\DOTag1
 - Item Id: 2874453319
 - Node Id: ns=2;i=206018
 - Attribute: Value
 - Sampling Interval (ms): 1000
 - Queue Size: 1
 - Index Range: (empty)
 - Discard Oldest: ☐
 - Sampling in Connected State: ☐
 - Log Values: ☐
 - Status: Reporting
- Filter Pane:** Shows filter settings.
 - Data Change Trigger: StatusValue
 - Deadband Type: None
 - Deadband Value: 0
- Read/Write Node Pane:** Shows the node details for the selected item.

Name	Value	Type
...DOTag1\DOTag1	True	Boolean
Value	True	Boolean
StatusCode	Good	StatusCode
SourceTimestamp	1/1/0001 12:00:0	DateTime
ServerTimestamp	4/23/2016 8:18:3	DateTime
- Message Log Pane:** Displays a log of messages.

Time	Level	Message Text	Trace Masks	Thread Id
8:20:54.592 PM	Info	Write operation completed for 1 values(s).	ClientAPI	16
8:20:40.747 PM	Info	Write operation completed for 1 values(s).	ClientAPI	6
8:20:20.653 PM	Info	Write operation completed for 1 values(s).	ClientAPI	4
8:20:03.121 PM	Info	Read operation completed for 3 values(s).	ClientAPI	4
8:20:01.761 PM	Info	Read node operation completed for node ns=2;i=206018.	ClientAPI	16
8:19:53.855 PM	Info	Read node operation completed for node ns=2;i=206018.	ClientAPI	8

The status bar at the bottom indicates "Data Access Root\Objects\PseUaTags\DOTag1\DOTag1, Active".

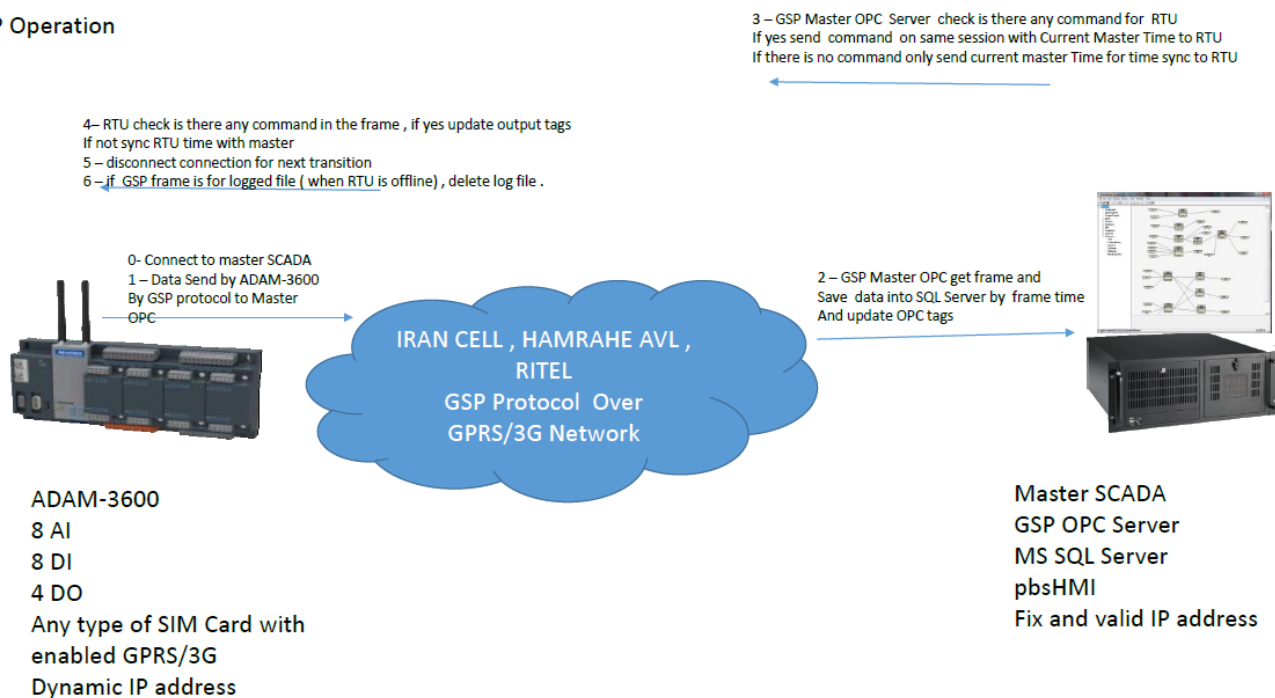
16 – GSP Client Driver Configuration

User should know GSP protocol concepts for proper using of GSP Driver. For detail Information about GSP protocol please refer to GSP specification documentation. http://pbscontrol.com/pdf/gsp_spec.pdf

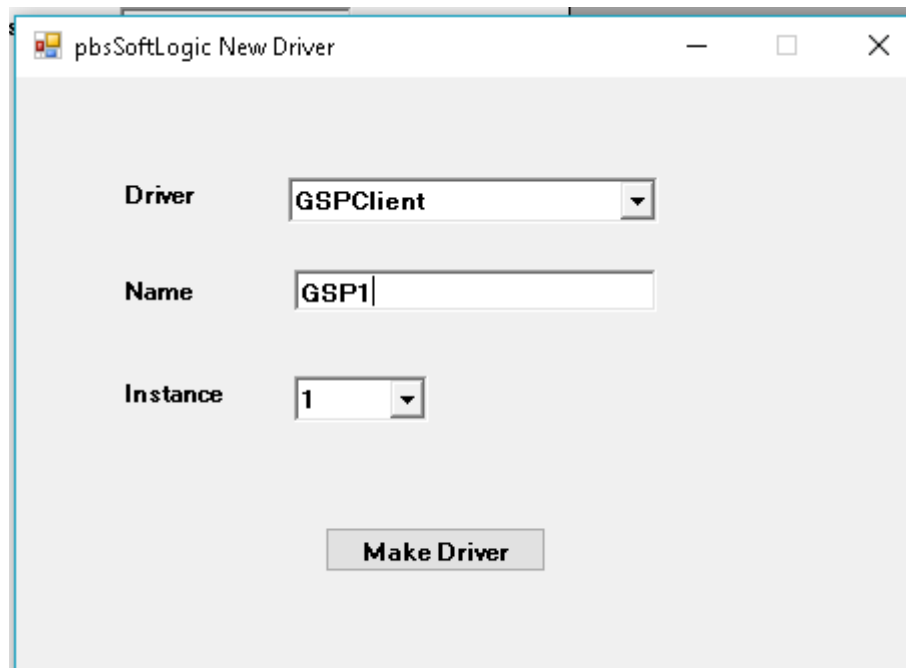
GSP (GPRS/3G for SCADA Projects) protocol is developed for SCADA systems which has following specification :

- There is no stable communication media between RTUs and Master SCADA.
- RTU has dynamic IP address.
- There is only one Valid and static IP address for master SCADA.
- Time Synchronization is required between master SCADA and RTUS.
- Local Data buffering in RTU is required when Communication is offline and sending logged data to SCADA master after establishing connection.
- Number of total I/O for transition between RTU and Master SCADA is not more than 600 Tags
- There are many RTUs in Project. GSP is designed for SCADA with huge number of RTUs.
- Target project for GSP is power and gas distribution SCADA.
- RTU has low Resources (No need for powerful CPU and too much RAM)

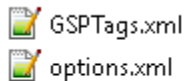
GSP Operation



Define a new Driver for your project and select GSPClient Driver.



Select a unique name for driver. Click on make driver. pbsSoftLogic will make default configuration for you . pbsSoftLogic will make two configuration file in driver directory .



Options file:

MasterIPAddress : Master IP address for sending GSP frame to SCADA server . We developed Free GSP OPC Server for master SCADA which you need to install it at master.

RTUID : Each RTU should has unique Numeric ID (1 , 2, 3, ..) for communication with master SCADA . Maximum ID can be 65535.

TCPPort : TCP Communication port . It should be same value for all RTUs and Master OPC server. You need to unblock this port number in your network for proper communication. Please refer to your IT team to assign one TCP port for your project.

SendGSPPeriod : RTU will send data based on this time . If value is less than 60 , it will send every n sec , but if it is more than 60 , it will send exactly on Every n Min . Suppose you set this parameter to 300. Then it will send exactly on following time xx:0 ,xx:5 , xx:10 , xx:15 ,xx:20 ,xx:25 , xx:30 , xx:35 , xx:40 , xx:45 ,xx:50 , xx:55 ,xx+1:0,...

SendbyChanges : RTU Will send data when SendByChange SYS tag is triggered to high .

LogPath : if RTU couldn't send GSP frame to master SCADA for 3 times retry , then it will save GSP frame in a local binary file which is located In LogPath . For example /home/gspdata/

Every time communication established to master SCADA, GSP will send all saved file to Master OPC Server. This Directory should be created in RTU.

OfflineSec : if RTU couldn't communicate with master SCADA in OfflineSec , then CommOnline Signal will change to 0 . When RTU has connection with Master SCADA CommOnline has value 1 .

You can restart RTU when CommOnline is dropped to 0 especially when GPRS/3G is using for communication with master SCADA.

GSPTags.xml file

pbsSoftLogic supports GSP client driver , so it is like Slave drivers . you need to define Tags and Write input tags and read Output Tags in your logic.

In following figure you can see different types of GSP tags :

```
<?xml version="1.0"?>
<OPCSrvTags>
  <Version>1.0.0</Version>
  <Tag Name="SendByChange" Type="SYS" Init="0" Address="0" Log="0" />
  <Tag Name="CommOnline" Type="SYS" Init="0" Address="1" Log="0" />
  <Tag Name="SaveByChange" Type="SYS" Init="0" Address="2" Log="0" />
  <Tag Name="NumOfLogFiles" Type="SYS" Init="0" Address="3" Log="0" />
  <Tag Name="DITag1" Type="DI" Address="1" Init="0" Log="0" />
  <Tag Name="DITag2" Type="DI" Address="2" Init="0" Log="0" />
  <Tag Name="DITag3" Type="DI" Address="3" Init="0" Log="0" />
  <Tag Name="DITag9" Type="DI" Address="9" Init="0" Log="0" />
  <Tag Name="AITag1" Type="AI" Address="1" Init="0" Log="0" />
  <Tag Name="AITag2" Type="AI" Address="2" Init="0" Log="0" />
  <Tag Name="AITag3" Type="AI" Address="3" Init="0" Log="0" />
  <Tag Name="AITag4" Type="AI" Address="4" Init="0" Log="0" />
  <Tag Name="AITag8" Type="AI" Address="8" Init="0" Log="0" />
  <Tag Name="AITag9" Type="AI" Address="9" Init="0" Log="0" />
  <Tag Name="AITag10" Type="AI" Address="10" Init="0" Log="0" />
  <Tag Name="FITag1" Type="FI" Address="1" Init="0" Log="0" />
  <Tag Name="FITag2" Type="FI" Address="2" Init="0" Log="0" />
  <Tag Name="FITag3" Type="FI" Address="3" Init="0" Log="0" />
  <Tag Name="FITag4" Type="FI" Address="4" Init="0" Log="0" />
  <Tag Name="LITag1" Type="LI" Address="1" Init="0" Log="0" />
  <Tag Name="LITag2" Type="LI" Address="2" Init="0" Log="0" />
  <Tag Name="LITag3" Type="LI" Address="3" Init="0" Log="0" />
</OPCSrvTags>
```

First four tags are system tags. Please do not change name, type and address.

SendByChange : When Change from 0 to 1 in your logic , GSP Driver provide GSP frame and start to send to server .

CommOnLine : When GSP Driver is sending Frames to Master and getting answer from master , CommOnLine is set to 1 . If OfflineSec is passed and GSP driver couldn't send any frame to master , CommOnline will change to 0 .

SaveByChange : When Change from 0 to 1 in logic , GSP Driver is reading current value of GSP Tags and save them in RTU Flash for sending to Master based on scheduling that is set for driver . SaveByChange only save data as back fill file and not start communicating with master.

NumOfLogFiles : shows number of Back fill Files in RTU . You can send this value to Master and use it in Modem Off/ON Logic . Suppose when number of Back Fill files are more than 20 , then make 3G Modem On until all Back fill files are transferred to Master .

There are following Tag types in GSP:

DI = Digital Input

AI = Analog Input (2 byte Signed)

FI = Float Input (4 Bytes)

LI = Long Input (4 Bytes Signed)

ULI = Unsigned long (4 Bytes Unsigned)

DO = Digital Output

AO = Analog Output (2 Bytes, Signed)

Each Tag Type has an address which is start from 1 and end with 64. So you can define maximum 64 tags from each type. Please notice that GSP is designed for distribution SCADA projects which has a lot of RTU but with small number of signals for communications.

Log: if you set Log to 1, last value of Signal will log to RTU flash (Persistent Tag) .

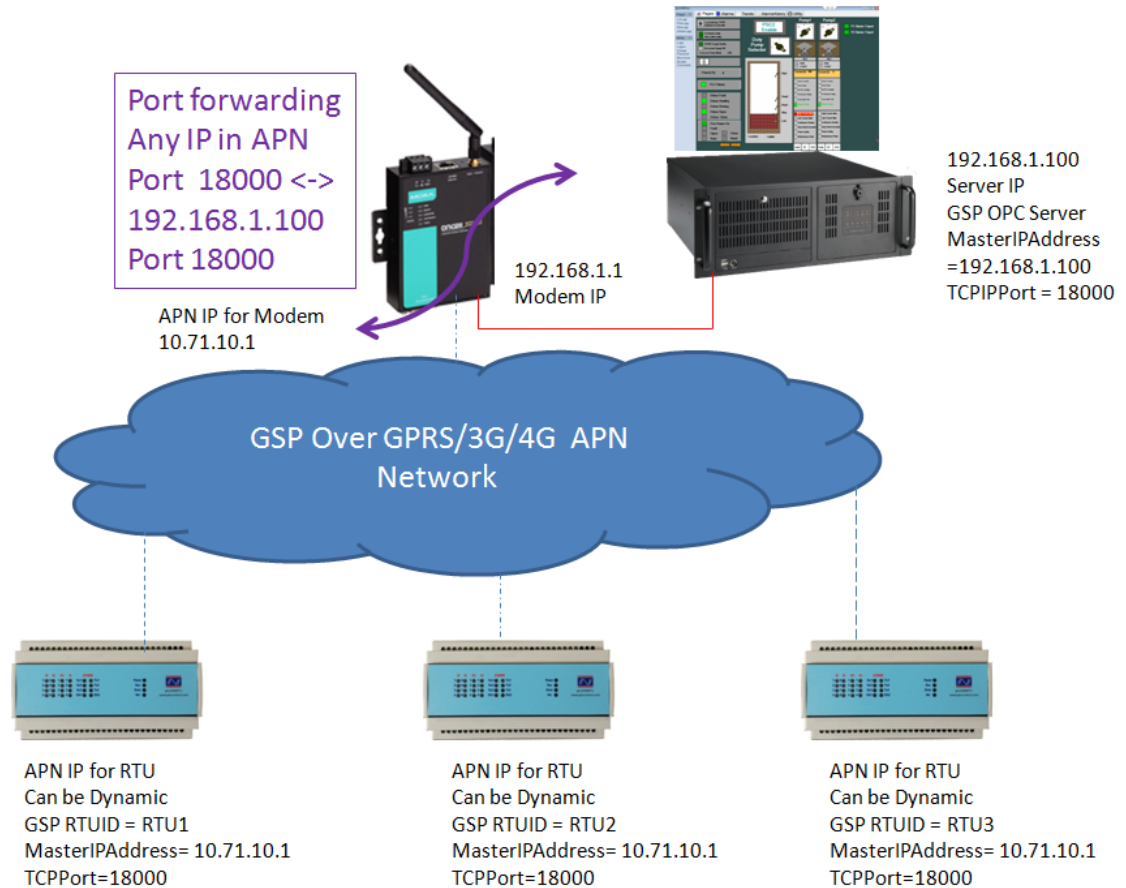
GSP Operation: when Driver detects Communication time, it will first connect to Master SCADA. If RTU can connect, it will send all tags to master by GSP frame format. When Master OPC server got frame, it will send server time and check is operator forced any Do or AO signals? If there is Forced Output signal, Master Will sends Time and DO and AO commands to RTU. If there is no any Output command, Master is only send Time for synchronization between Master and RTU.

When RTU received correct answer, it will close connection with master.

Because each transaction starts from RTU, So we don't need Fix IP address in RTU. Only RTU ID is enough.

Configuration of Sample system based on 3G Network

- Suppose you are using APN for SCADA Communication
- In master SCADA you have 3G Modem Like MOXA OnCell3151.
- There are 3 AMS-R3010 RTU at site and need to communicate with Master SCADA by GSP Protocol . As following Diagram



- Suppose you are using TCP port number 18000 for GSP Communication . Set GSP Parameter "TCPPort" to 18000
- When Setting GSP Driver parameter , you need to set APN IP of MOXA 3G Modem for each RTU as "MasterIPAddress" parameter .
- Give unique name for each RTU in "RTUID" parameter.
- In MOXA OnCell3151 enable DHCP Server functionality.

The screenshot shows a web-based configuration interface for network settings. It is divided into two main sections: 'Basic Network Settings' and 'DHCP Server Configuration'. The 'Basic Network Settings' section includes fields for IP configuration (Static), IP address (192.168.127.254), Netmask (255.255.255.0), Gateway, WINS function (Enable), WINS server, and LAN speed (Auto). The 'DHCP Server Configuration' section includes fields for DHCP server (Enable), DNS relay (Enable), Start IP address, Maximum dynamic users (2), Client lease time (1 day), and Static IP mapping (Disable). A 'Submit' button is located at the bottom of the form.

Basic Network Settings	
IP configuration	Static
IP address	192.168.127.254
Netmask	255.255.255.0
Gateway	
WINS function	<input checked="" type="radio"/> Enable <input type="radio"/> Disable
WINS server	
LAN speed	Auto

DHCP Server Configuration	
DHCP server	<input checked="" type="radio"/> Enable <input type="radio"/> Disable
DNS relay	<input checked="" type="radio"/> Enable <input type="radio"/> Disable
Start IP address	
Maximum dynamic users	2
Client lease time	1 (1~10 days)
Static IP mapping	<input type="radio"/> Enable <input checked="" type="radio"/> Disable

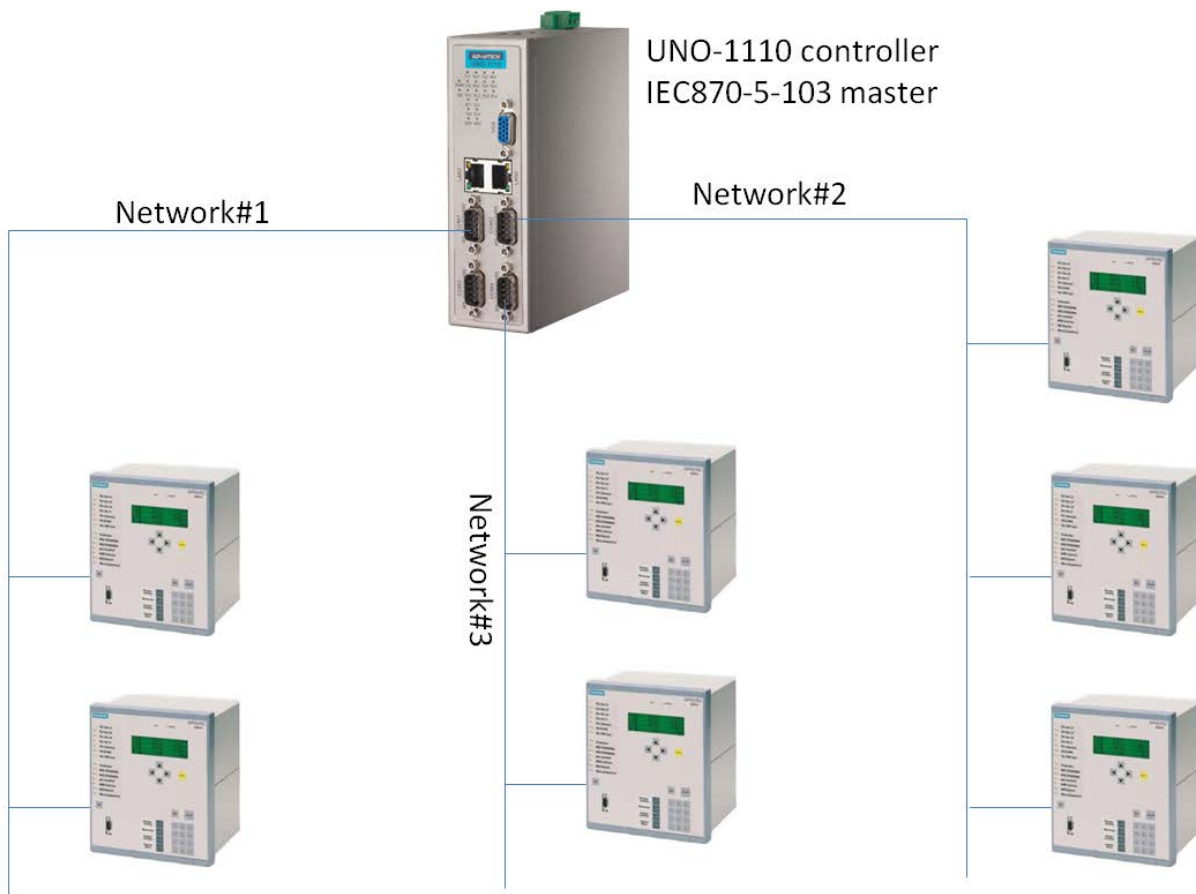
Submit

-
- Set DHCP Server , Start IP Address and Maximum dynamic Users .
- Enable Static IP mapping and Configure Modem to give always static IP address Like 192.168.1.100 to SCADA Server .
- Define port forwarding in MOXA Modem , forward all TCP frames that is coming from APN with port Number 18000 will map to SCADA Server (IP = 192.168.1.100) by same port 18000 .
- Download and install GSP Master OPC Server from www.pbscontrol.com on SCADA Server .
- When you configure GSP OPC Server set IP address of Server in "MasterIPAddress" parameter(192.168.1.100) and set "TCPIPPort" to 18000 .

17 – IEC870-5-103 Master Driver Configuration

pbsSoftlogic version 1.6.5 supports IEC870-5-103 master protocol for communication with protection relays.

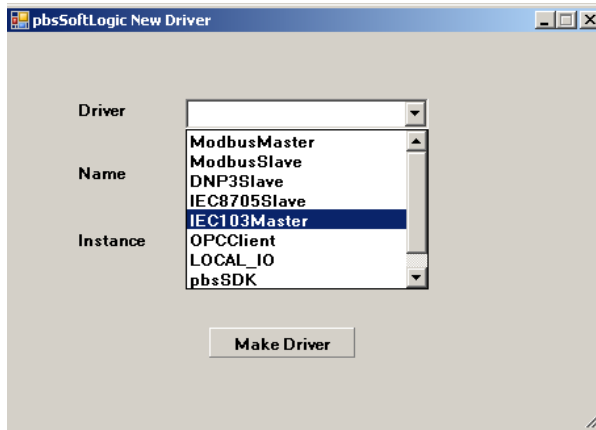
Up to 8 networks can be configured for a controller to communicate with different protection relays.



For each network, you can define up to 1024 IEC tags.

Defining IEC870-5-103 master driver in pbsSoftLogic :

- open project option page
- right click on Driver list and select IEC103Master Driver



- Write unique driver name and select Instance. Instance number is between 1 and 8 and shows IEC103 Network number.
- Click on make driver button.
- pbsSoftLogic will make default configuration files at driver directory for current project .
 - o options.xml communication parameters settings
 - o IEC103Devices.xml IEC103 slave device configuration and Tags

Options.xml file content:

```
<Options>
  <Node>
    <Name>COMPort</Name>
    <Desc>Serial Port for Communication 1,2,3,4,5,...</Desc>
    <Value>3</Value>
  </Node>
  <Node>
    <Name>BaudRate</Name>
    <Desc>9600,19200,36400,52700,115200</Desc>
    <Value>9600</Value>
  </Node>
  <Node>
    <Name>DataBit</Name>
    <Desc>7,8</Desc>
    <Value>8</Value>
  </Node>
  <Node>
    <Name>StopBit</Name>
    <Desc>1,2</Desc>
    <Value>1</Value>
  </Node>
  <Node>
    <Name>Parity</Name>
    <Desc>None,Even,Odd</Desc>
    <Value>Even</Value>
  </Node>
  <Node>
    <Name>Instance</Name>
    <Desc>Instance</Desc>
    <Value>1</Value>
  </Node>
</Options>
```

IEC103Devices.xml file content

```

<Devices>
  <Device Name="Relay8" Address="8" ResetLinkFC="0" PollPeriod="300" CommTimeOutSecSP="15" SendGIPeriod="600">
    <IEC103Tags>
      <Tag Name="TimeTagMsg1" TypeInd="1" FunType="40" InfNum="1" Position="0" />
      <Tag Name="TimeTagMsg2" TypeInd="1" FunType="40" InfNum="2" Position="0" />
      <Tag Name="TimeTagMsg3" TypeInd="1" FunType="40" InfNum="3" Position="0" />
      <Tag Name="TimeTagMsg4" TypeInd="1" FunType="40" InfNum="4" Position="0" />
      <Tag Name="TimeTagMsg5" TypeInd="1" FunType="40" InfNum="5" Position="0" />
      <Tag Name="TimeTagMsg6" TypeInd="1" FunType="40" InfNum="6" Position="0" />
      <Tag Name="TimeTagMsg7" TypeInd="1" FunType="40" InfNum="7" Position="0" />
      <Tag Name="TimeTagMsg8" TypeInd="1" FunType="40" InfNum="8" Position="0" />
      <Tag Name="MeasurandI1" TypeInd="3" FunType="160" InfNum="145" Position="0" />
      <Tag Name="MeasurandI2" TypeInd="3" FunType="160" InfNum="145" Position="1" />
      <Tag Name="MeasurandI3" TypeInd="3" FunType="160" InfNum="145" Position="2" />
      <Tag Name="MeasurandI4" TypeInd="3" FunType="160" InfNum="145" Position="3" />
      <Tag Name="MeasurandI5" TypeInd="3" FunType="160" InfNum="145" Position="4" />
      <Tag Name="MeasurandI6" TypeInd="3" FunType="160" InfNum="145" Position="5" />
      <Tag Name="MeasurandI7" TypeInd="3" FunType="160" InfNum="145" Position="6" />
      <Tag Name="MeasurandI8" TypeInd="3" FunType="160" InfNum="145" Position="7" />
      <Tag Name="MeasurandII1" TypeInd="9" FunType="134" InfNum="146" Position="0" />
      <Tag Name="MeasurandII2" TypeInd="9" FunType="134" InfNum="146" Position="1" />
      <Tag Name="MeasurandII3" TypeInd="9" FunType="134" InfNum="146" Position="2" />
      <Tag Name="MeasurandII4" TypeInd="9" FunType="134" InfNum="146" Position="3" />
      <Tag Name="MeasurandII5" TypeInd="9" FunType="134" InfNum="146" Position="4" />
      <Tag Name="MeasurandII6" TypeInd="9" FunType="134" InfNum="146" Position="5" />
      <Tag Name="MeasurandII7" TypeInd="9" FunType="134" InfNum="146" Position="6" />
      <Tag Name="MeasurandII8" TypeInd="9" FunType="134" InfNum="146" Position="7" />
    </IEC103Tags>
  </Device>
</Devices>

```

For each protection relay on a network, you should define one <Device> Tag.

<Device> Name Attribute: for each Relay in a network, you should consider unique name

<Device> address: Data Link layer and ASDU Address of relay.

<Device> ResetLinkFC : shows which Function code is use for resting communication with relay . 0 or 7

<Device> PollPerid : shows period of sending class1 or class2 request to relay in msec . IEC103 driver is sending one class1 request for reading events and one class2 request for reading analog signals periodically.

<Device> CommTimeoutSecSP : shows after how many sec , driver will reset and reinitialized communication with relay when there is no communication with relay .

<Device>SendGIPeriod : Shows after how many min , IEC103 driver will send GI command to relay .

<IEC103Tags><Tag> Name : Unique name for a tag in a device .

<IEC103Tags><Tag> TypeInd :

- TYPE IDENTIFICATION 1: Time-tagged message
- TYPE IDENTIFICATION 2: Time-tagged message with relative time
- TYPE IDENTIFICATION 3: Measurands I
- TYPE IDENTIFICATION 9: Measurands II

<IEC103Tags><Tag>FunType: Function Type for tag. Please refer to relay IEC103 documentation for detail function types.

<IEC103Tags><Tag>InfNum: Information Number for tag. Please refer to relay IEC103 documentation for detail Information Number.

<IEC103Tags><Tag> Position : Position of analog signal in MEASURANDS I and II . Starting from 0.

Sample IEC103 Tags from Siemens 7SK80 Series Relay:

No.	Description	Function	IEC 60870-5-103					Configurable in Matrix		
			Type	Information Number	Compatibility	Data Unit	Position	CFC	Control Display	Default Display
-	Number of TRIPs= (#of TRIPs=)	Statistics	-	-	-	-	-	CFC		
-	Operating hours greater than (OpHour>)	SetPoint(Stat)	-	-	-	-	-	CFC		
601	Ia (Ia =)	Measurement	134	157	No	9	1	CFC		
602	Ib (Ib =)	Measurement	160	145	Yes	3	1	CFC		
			134	157	No	9	2			
603	Ic (Ic =)	Measurement	134	157	No	9	3	CFC		
604	In (In =)	Measurement	134	157	No	9	4	CFC		
605	I1 (positive sequence) (I1 =)	Measurement	-	-	-	-	-	CFC		
606	I2 (negative sequence) (I2 =)	Measurement	-	-	-	-	-	CFC		
621	Va (Va =)	Measurement	134	157	No	9	6	CFC		
622	Vb (Vb =)	Measurement	134	157	No	9	7	CFC		
623	Vc (Vc =)	Measurement	134	157	No	9	8	CFC		
624	Va-b (Va-b=)	Measurement	160	145	Yes	3	2	CFC		
			134	157	No	9	9			
625	Vb-c (Vb-c=)	Measurement	134	157	No	9	10	CFC		
626	Vc-a (Vc-a=)	Measurement	134	157	No	9	11	CFC		
627	VN (VN =)	Measurement	134	118	No	9	1	CFC		

pbssoftLogic Tag definition for some above relay signals :

<Tag Name="Ia" TypeInd="9" FunType="134" InfNum="157" Position="0" />

<Tag Name="Ib" TypeInd="3" FunType="160" InfNum="145" Position="0" />

<Tag Name="Ic" TypeInd="9" FunType="134" InfNum="157" Position="2" />

<Tag Name="Va" TypeInd="9" FunType="134" InfNum="157" Position="5" />

<Tag Name="VN" TypeInd="9" FunType="134" InfNum="118" Position="0" />

Sample IEC103 tags from Siemens 7sk80 series relay:

3.2.1 Automatic reclosure status

ASDU	Function type	Information number	Name	Description	Obj. - Adr.
1	40	1	>79 ON	>79 ON; Automatic reclosure ON; ON = 1, OFF = 0	2701
1	40	2	>79 OFF	>79 OFF; ON = 1, OFF = 0	2702
1	40	3	> BLOCK 79	>BLOCK 79; ON = 1, OFF = 0	2703
2	40	15	>Start 79 Gnd	>Start 79 Ground programm; ON = 1, OFF = 0	2715
2	40	16	>Start 79 Ph	>Start 79 Phase programm; ON = 1, OFF = 0	2716
1	40	20	>Enable ANSI#-2	>Enable 50/67-(N)-2 (override 79 blk); ON = 1, OFF = 0	2720
1	40	30	>CB Ready	>Circuit breaker READY for reclosing; ON = 1, OFF = 0	2730
1	40	81	79 OFF	79 Auto recloser is switched OFF; ON = 1, OFF = 0	2781
2	40	85	79 DynBlock	79 – Auto-reclose is dynamically BLOCKED; ON = 1, OFF = 0	2785
1	40	101	79 in progress	79 – in progress; ON = 1, OFF = 0	2801
1	40	162	79 Successful	79 – cycle successful; ON = 1, OFF = 0	2862
2	40	163	79 Lockout	79 – Lockout; ON = 1, OFF = 0	2863
2	40	180	79 L_N Sequence	79-A/R single phase reclosing sequence; Program earthfault is running = 1, Program is deactivated = 0	2878
2	40	181	79 L-L Sequence	79-A/R multi-phase reclosing sequence; ON = 1, OFF = 0	2879
1	160	16	79 ON	79 Auto recloser is switched ON; ON = 1, OFF = 0	2782
1	160	128	79 Close	79 – Close command; ON = 1	2851

pbssoftLogic Tag definition for some above relay signals :

```
<Tag Name="79_ON" TypeInd="1" FunType="40" InfNum="1" Position="0" />
```

```
<Tag Name="79_OFF" TypeInd="1" FunType="40" InfNum="2" Position="0" />
```

```
<Tag Name="79_Block" TypeInd="1" FunType="40" InfNum="3" Position="0" />
```

```
<Tag Name="79_Start_Gn" TypeInd="2" FunType="40" InfNum="15" Position="0" />
```

```
<Tag Name="79_Successful" TypeInd="1" FunType="40" InfNum="162" Position="0" />
```

IEC103 master driver runtime operation:

When IEC103 master driver starts, it will do following sequence:

- send INIT command to relay (FC = 0 or FC = 7)
- Sending Class1 request until getting Identification message ASDU = 5 , COT = resetCU or ResetFCB
- Send Time Synchronization command to relay TYPE ID = 6 , COT = 8
- Send GI Command
- Sending Class1 Request until getting ASDU = 8 , COT = End of GI
- Send Class 2 for reading analog signals
- Send Class 1 for reading Events
- Send Class 2 for reading analog signals
- Send Class 1 for reading Events
- If it is time for sending GI command , send GI command
- If there is no communication from relay side after COMMMTimeOutSecSP Sec , make relay offline and try to initialize relay and repeat above sequence

Note : In current version of IEC103 master driver , Sending Set points and commands are not supported. So you can use IEC103 driver just for reading events and analog signals for monitoring purpose from relay.

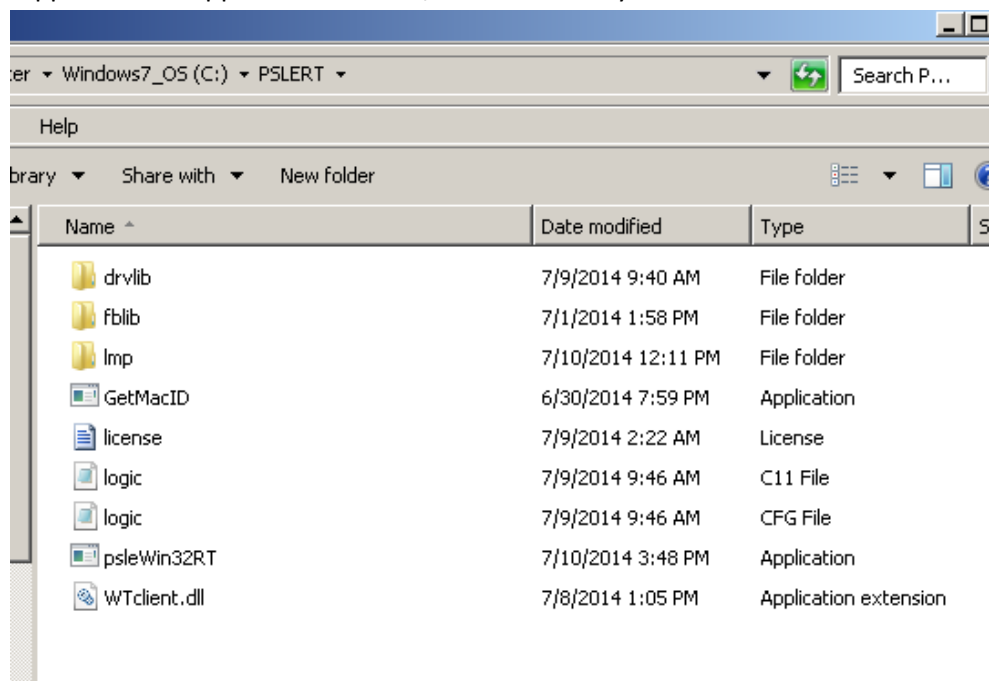
18 – OPC Client Driver Configuration for Win32 Target

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

There are two windows32 runtime kernels for pbsSoftLogic:

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

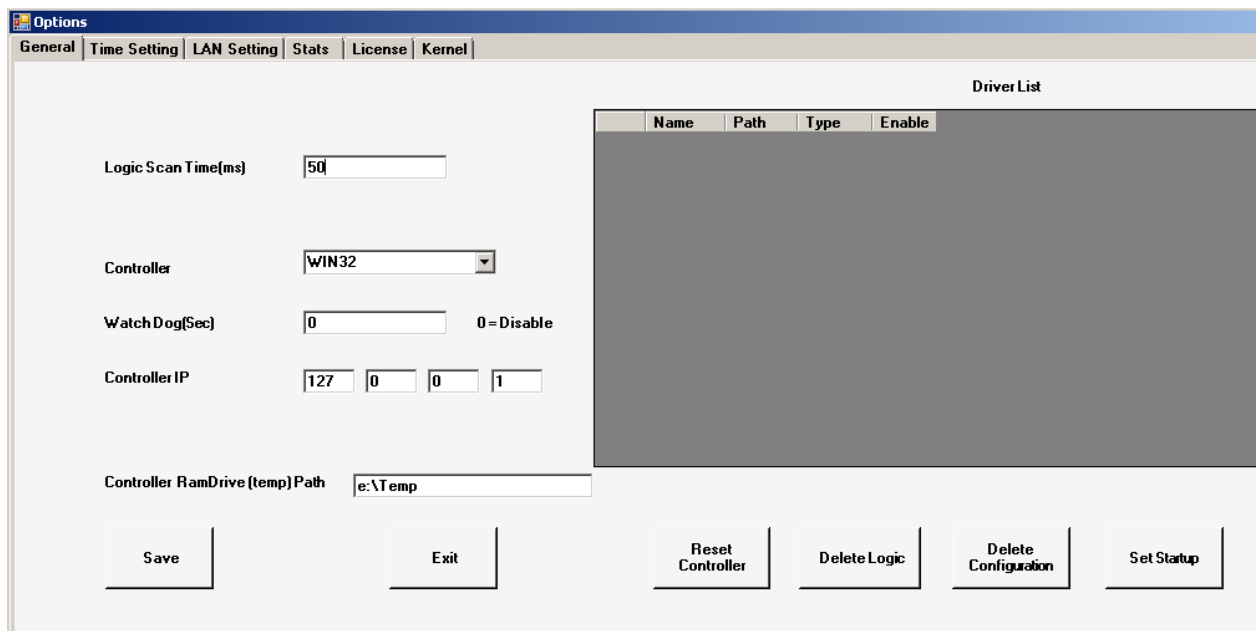
Download latest Win32 target from www.pbscontrol.com . Unzip it on any drive in your controller. Suppose we unzipped kernel on C:\PSLERT Directory.



- 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
- Fbllib : 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:

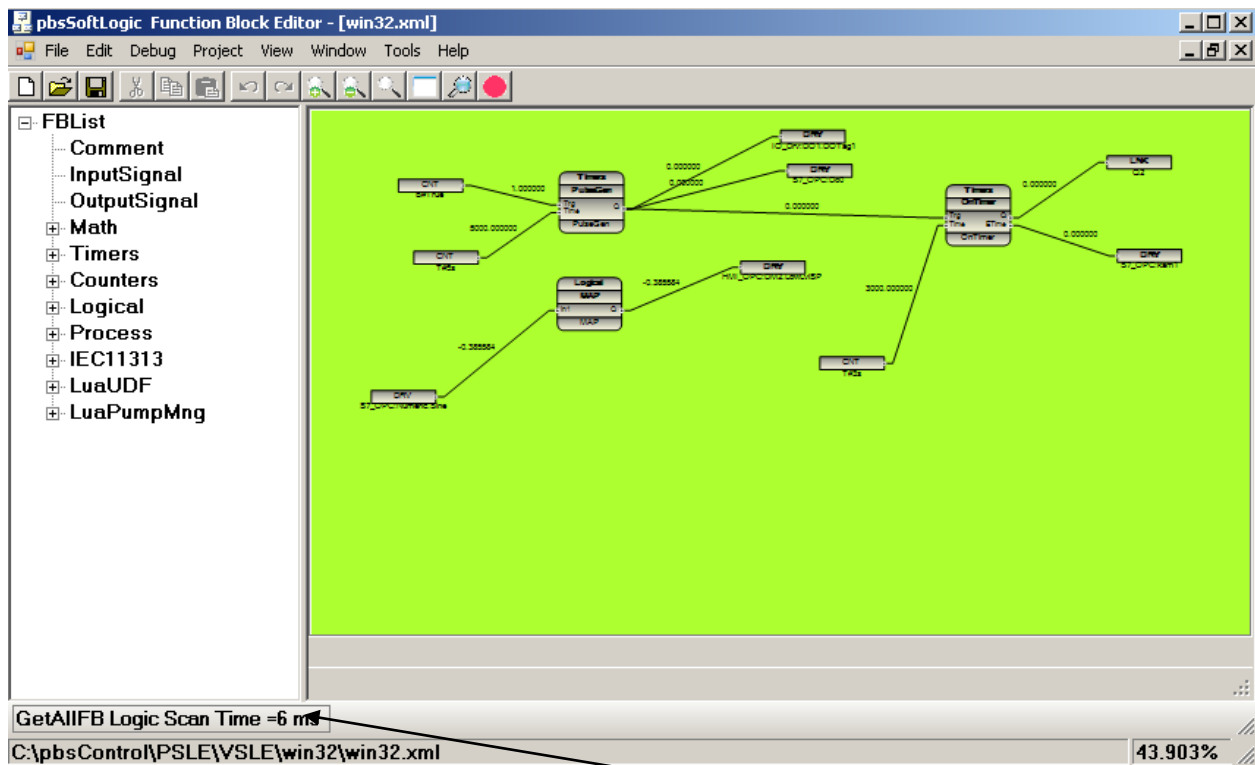


Controller Type is Win32.

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.



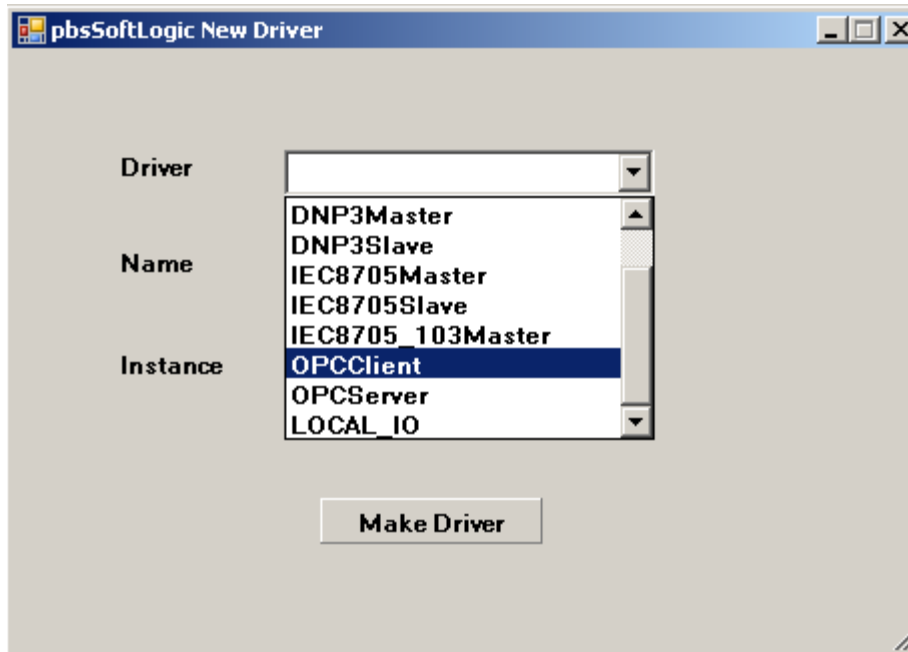
In above sample, we used two UNO-1170 as controllers and one station as programming station.

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

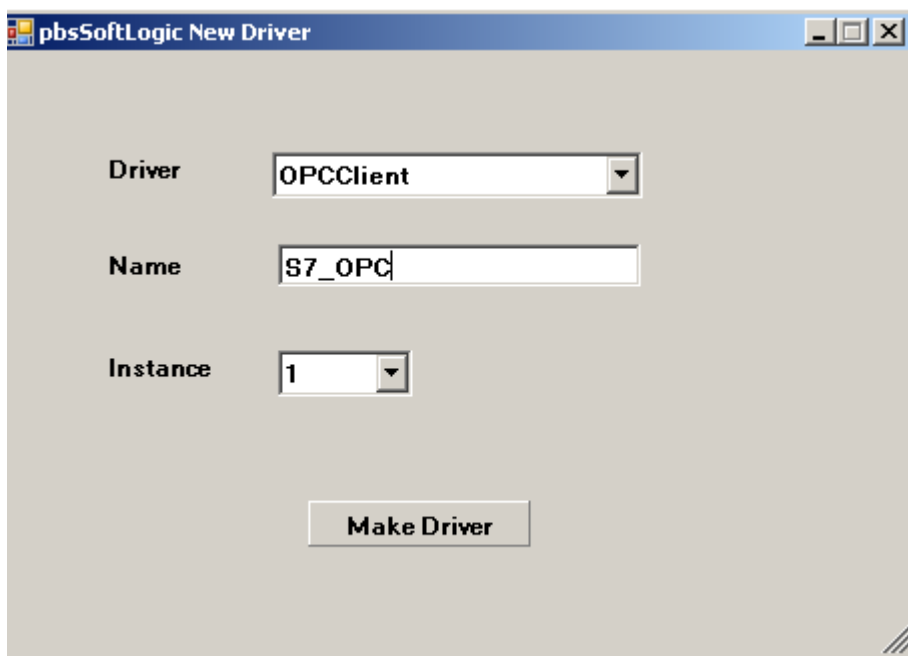
If you need to pass data between controllers, then you can define modbus-TCP master on one controller and Modbus-TCP slave on another controller. You can also use DNP3 over TCP and IEC870-5-104 for communicating between two controllers.

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 Instance ID.

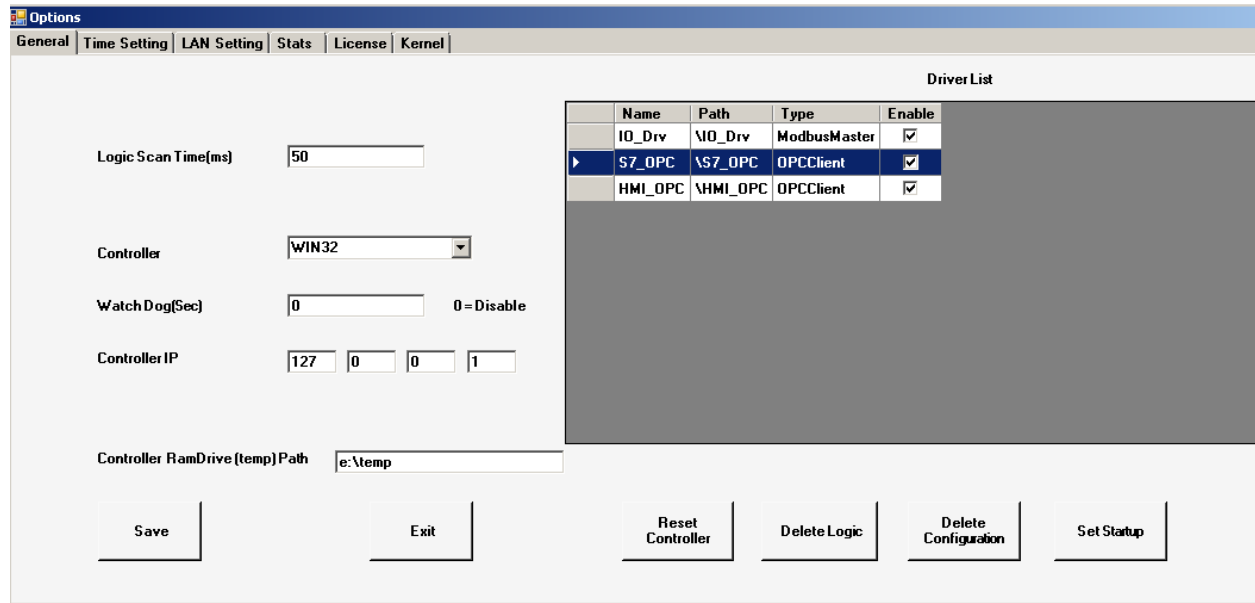


Click on “Make Driver” Button. pbsSoftLogic will make basic definition in your project .

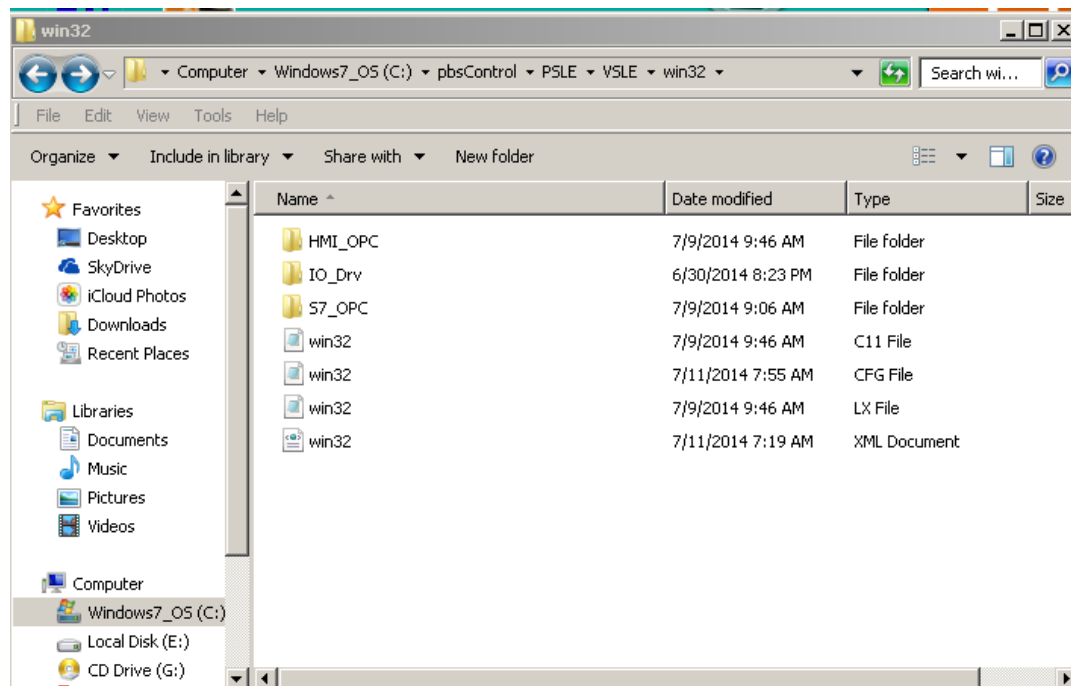
OPC client Driver : pbsSoftLogic runtime kernel will connect to other OPC servers DA2.0

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.

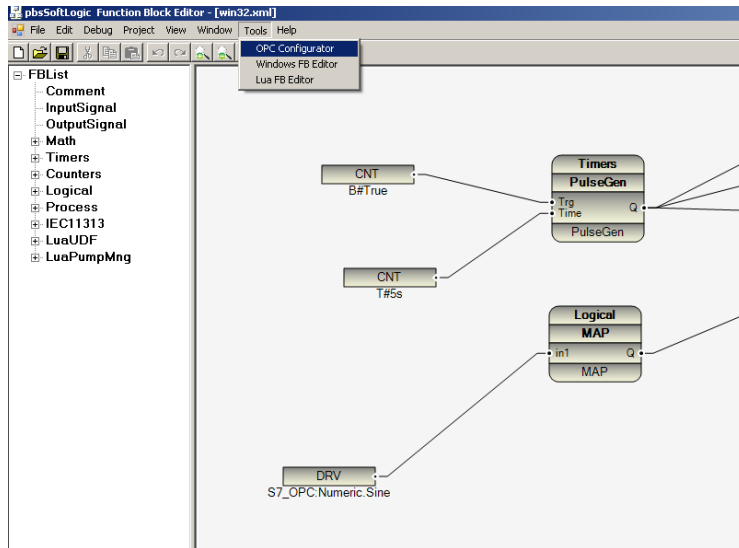


pbsSoftLogic will make a new folder in project directory with same driver name .



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

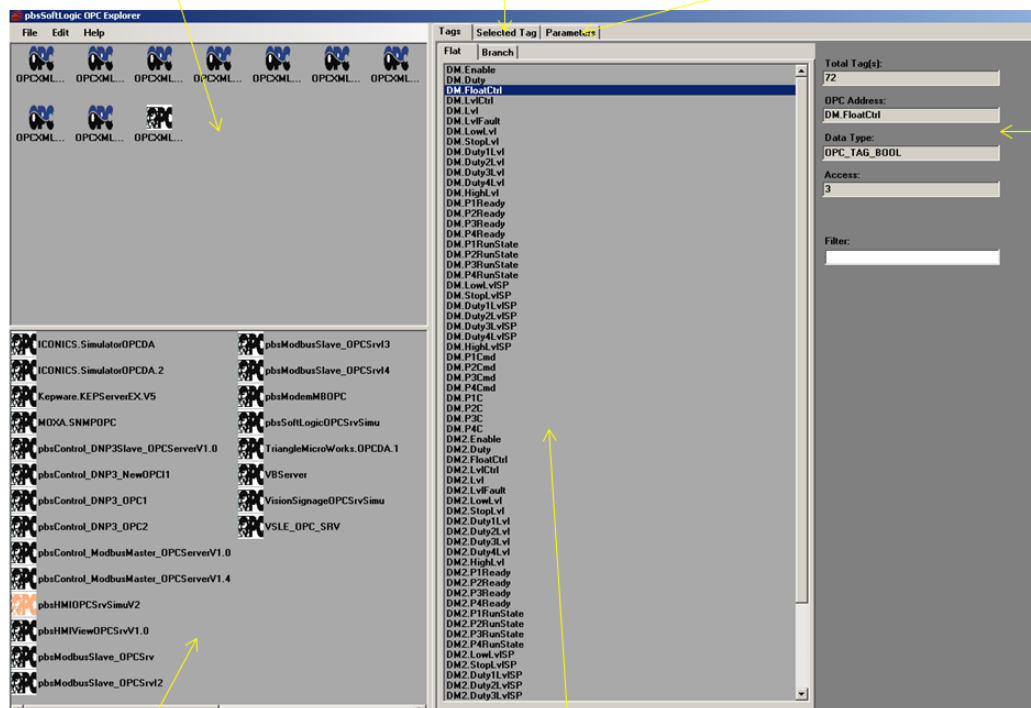
For making OPCTags.xml you should use pbsSoftlogic OPC configurator utility at Tools menu .



Defined OPC configuration files

Selected OPC server Tags

OPC configuration parameters



OPC Tag properties

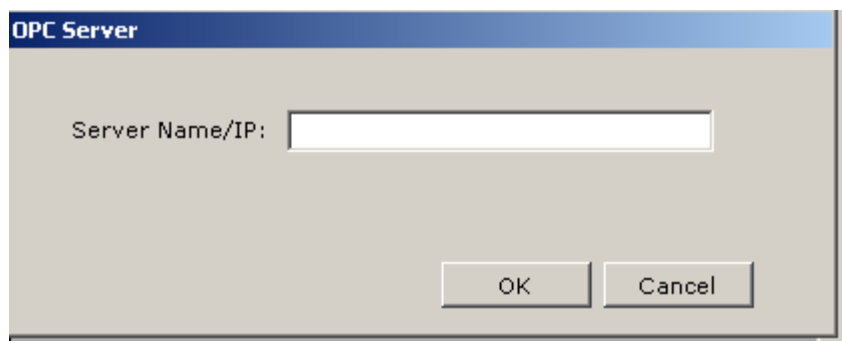
Installed OPC servers on this PC
or network PC

Connected OPC server Tags

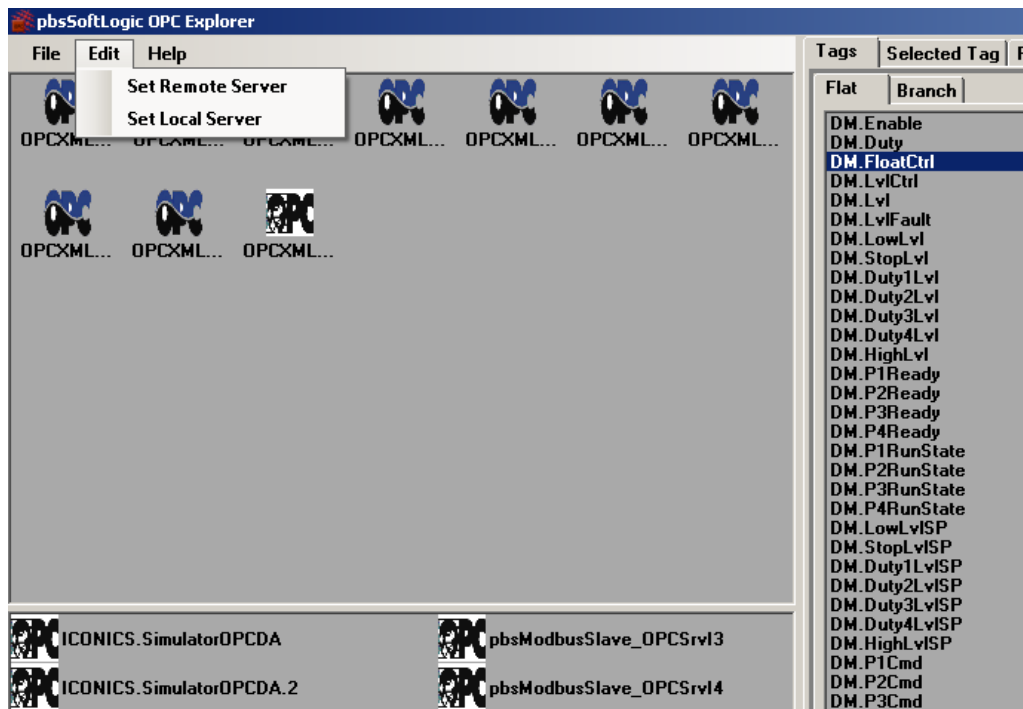
Defined OPC configuration files: At Top left panel you can see all defined OPC configuration files. These files are located at \PSLE\OPC folder.

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.

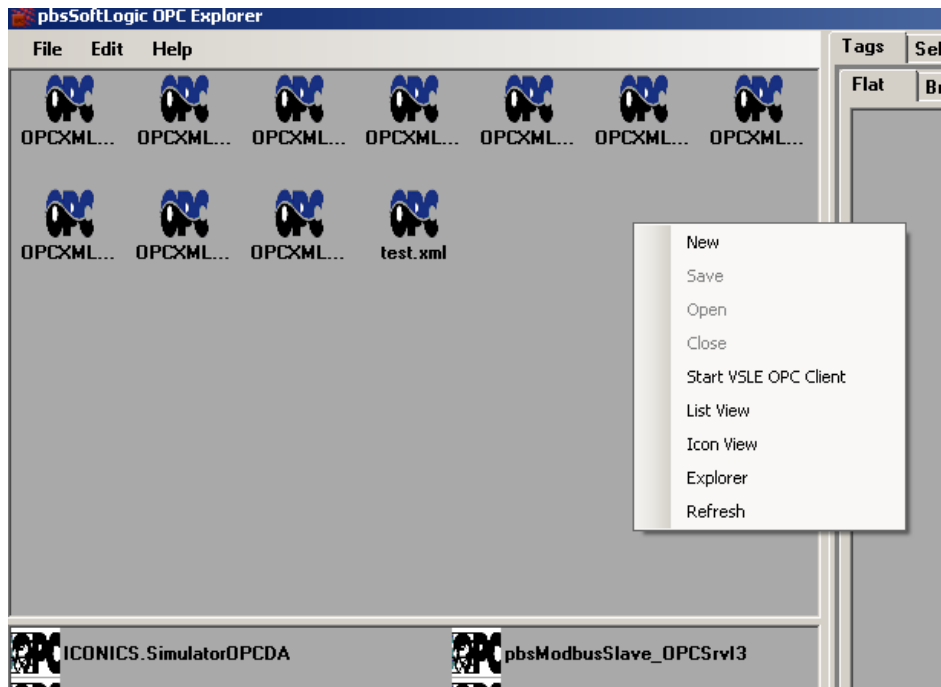


For changing to local Machine, from Edit menu select "Set Local Server".



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

And select “new” menu.

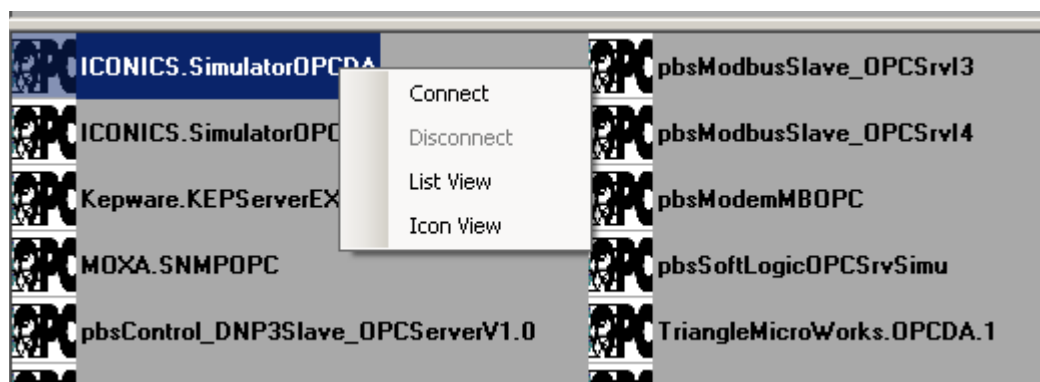


OPC explorer will look at \psle\OPC directory and find all files with OPCXMLn.xml name format and will make a new file with OPCXML{n+1}.xml name when n is max number in the OPC directory .

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

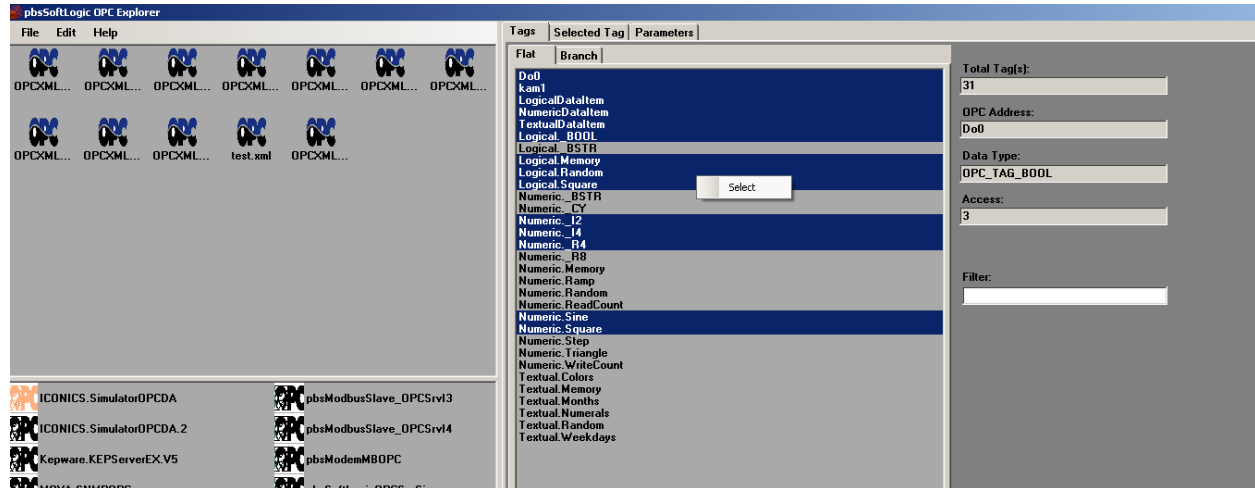
By running refresh Menu, Defined OPC configuration file panel will be refreshed with new names.

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.

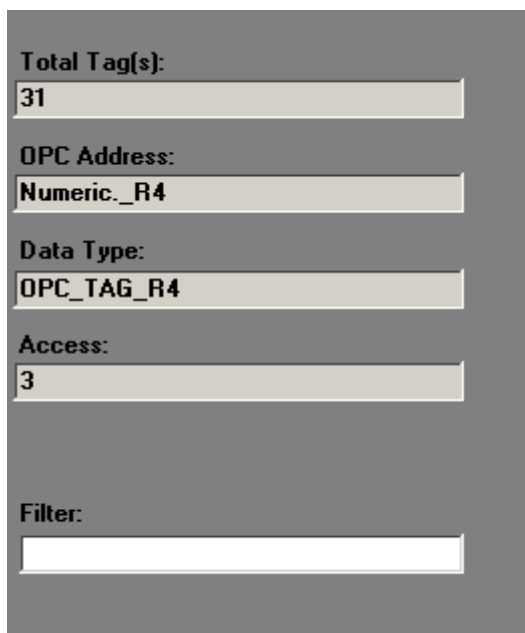


String and date data types are not supported in pbsSoftLogic for OPC client Driver.

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:



The screenshot shows a configuration panel for OPC tag properties. It contains the following fields:

- Total Tag(s):** 31
- OPC Address:** Numeric_R4
- Data Type:** OPC_TAG_R4
- Access:** 3
- Filter:** (empty text box)

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

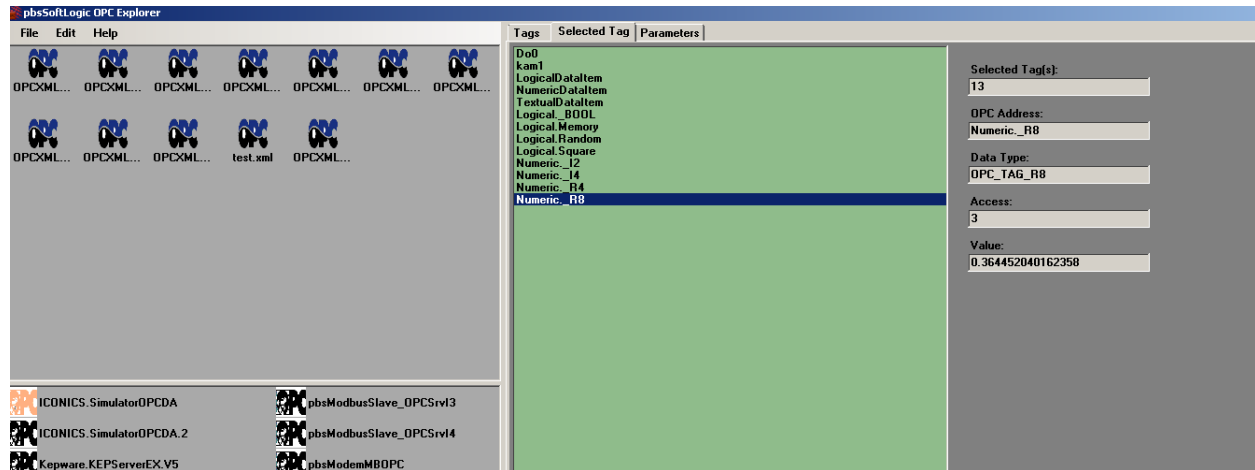
VT_BOOL has different definition in OPC:

VT_BOOL True=-1, False=0

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

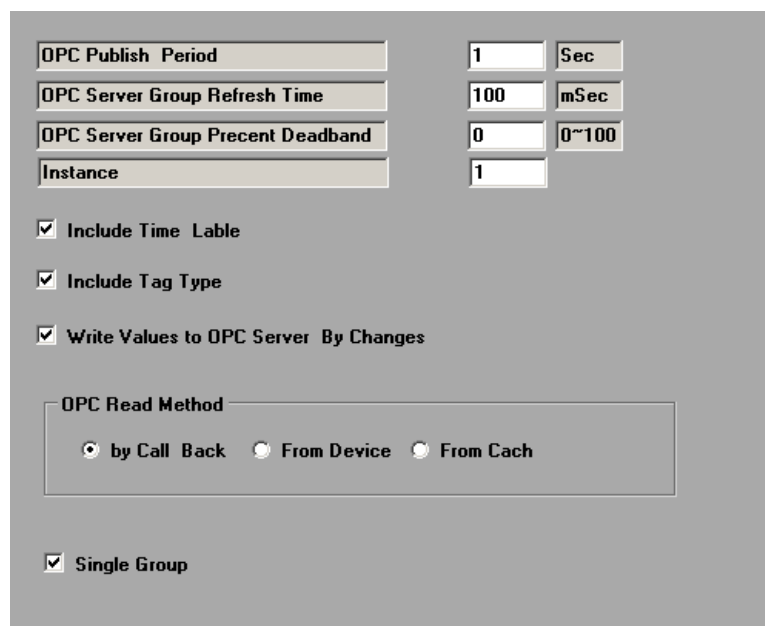
VT_BOOL True=1, False=0

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.

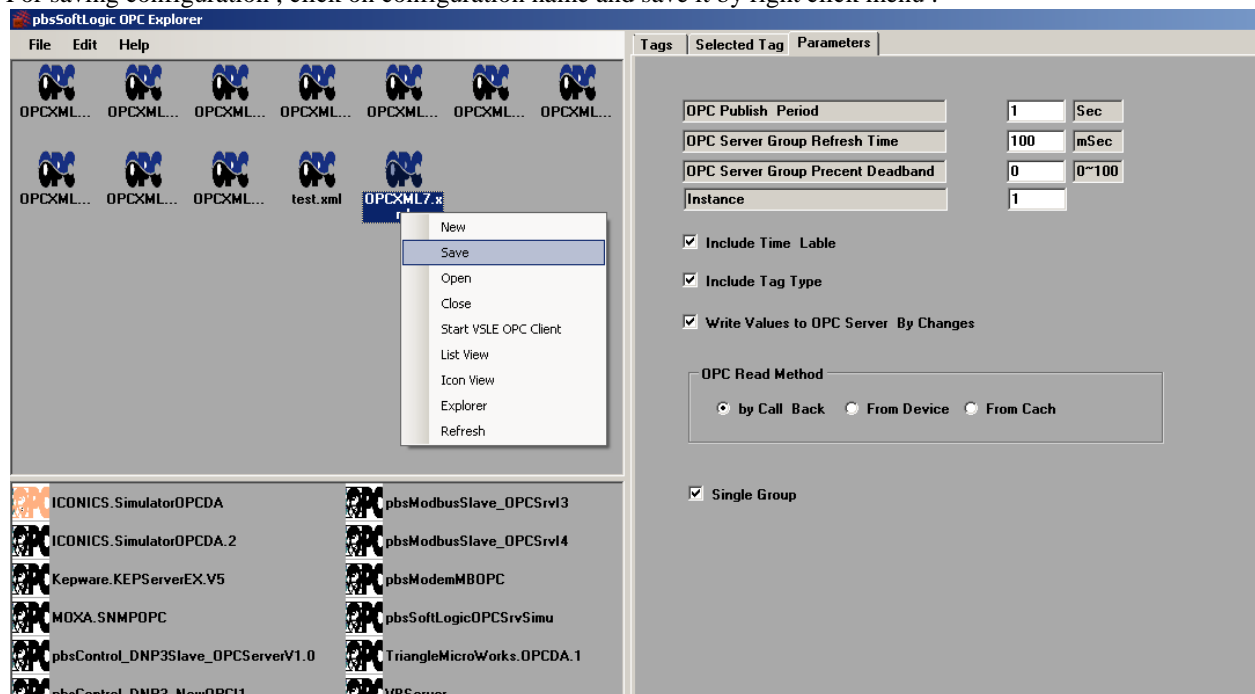
OPC Server Group refresh Time definition from OPC standard:

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

Instance:

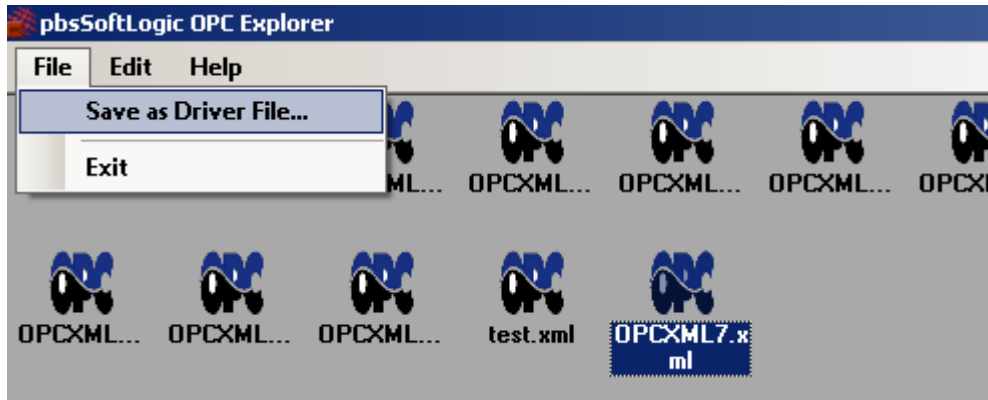
Instance number is same as driver instance number.

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



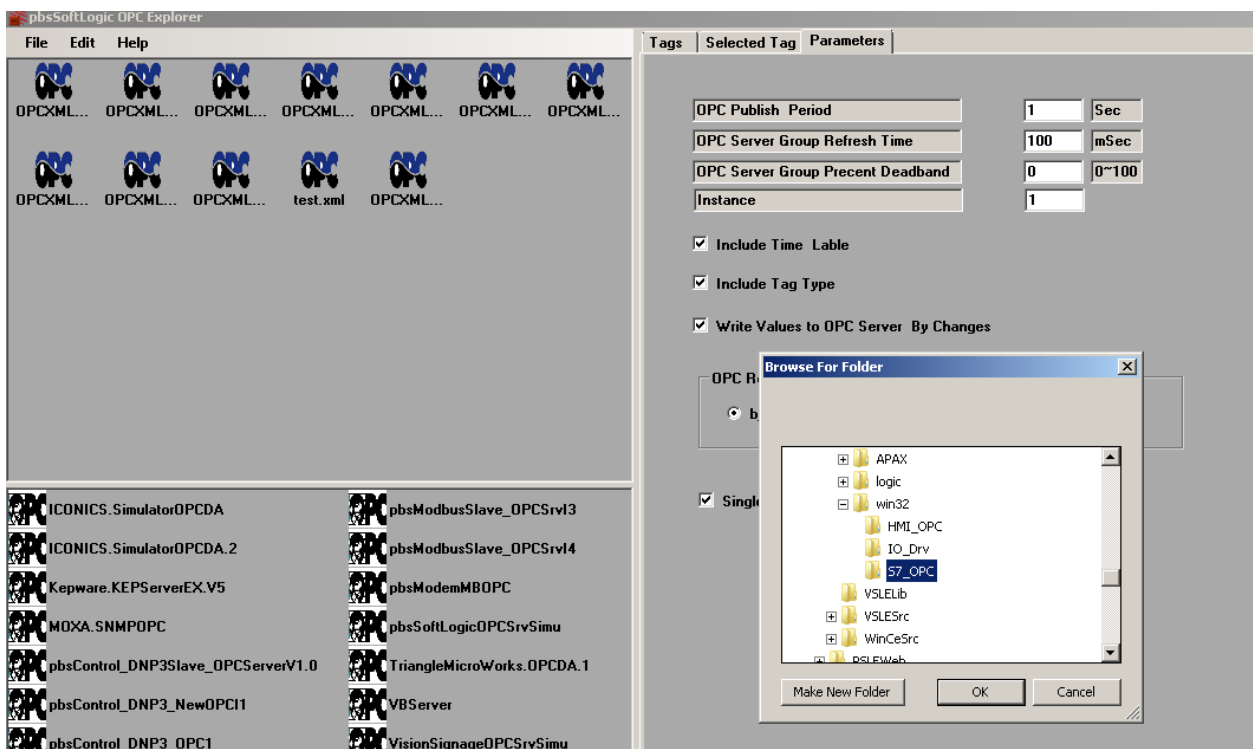
OPC configuration file will be saved at \psle\OPC directory. This file is same file that is used in OPC client driver configuration.

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.

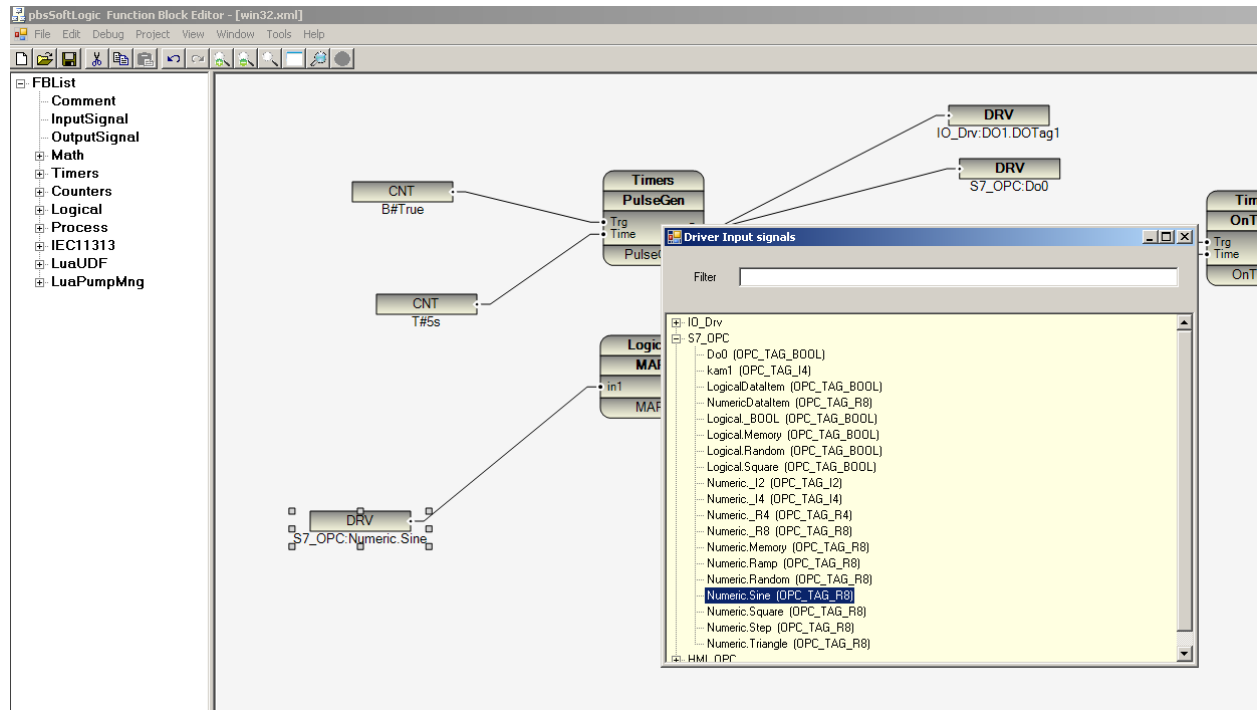
You should select same folder that is made by pbssoftlogic when you define OPC Client driver in project setting page.



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

19 – User defined function block by Lua Scripting and C Language

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.

You need to use Lua scripting language for writing UDF code and with help of XML file you can define UDF body .

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 .

Three steps are required for adding UDF to pbsSoftLogic:

- 1- Defining FB Input / Output structure
- 2- Writing FBD Inside by Lua scripting

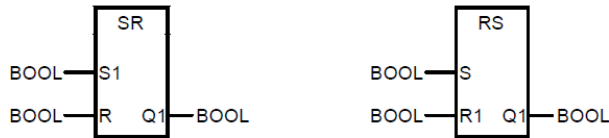
C Source code of all internal FB are included in pbsSoftLogic. You can use these source codes to make new FB and expand platform.

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.

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
```

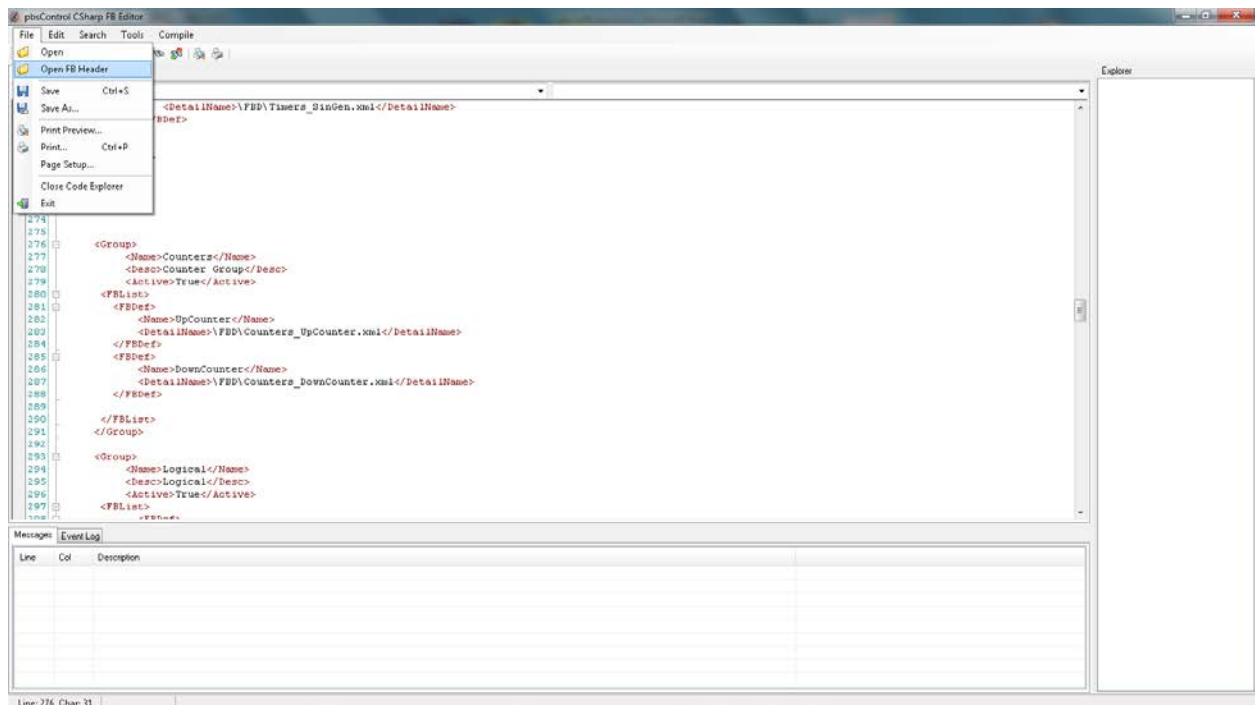
FUNCTION_BLOCK **RS** (* flip flop reset dominant *)

```
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:

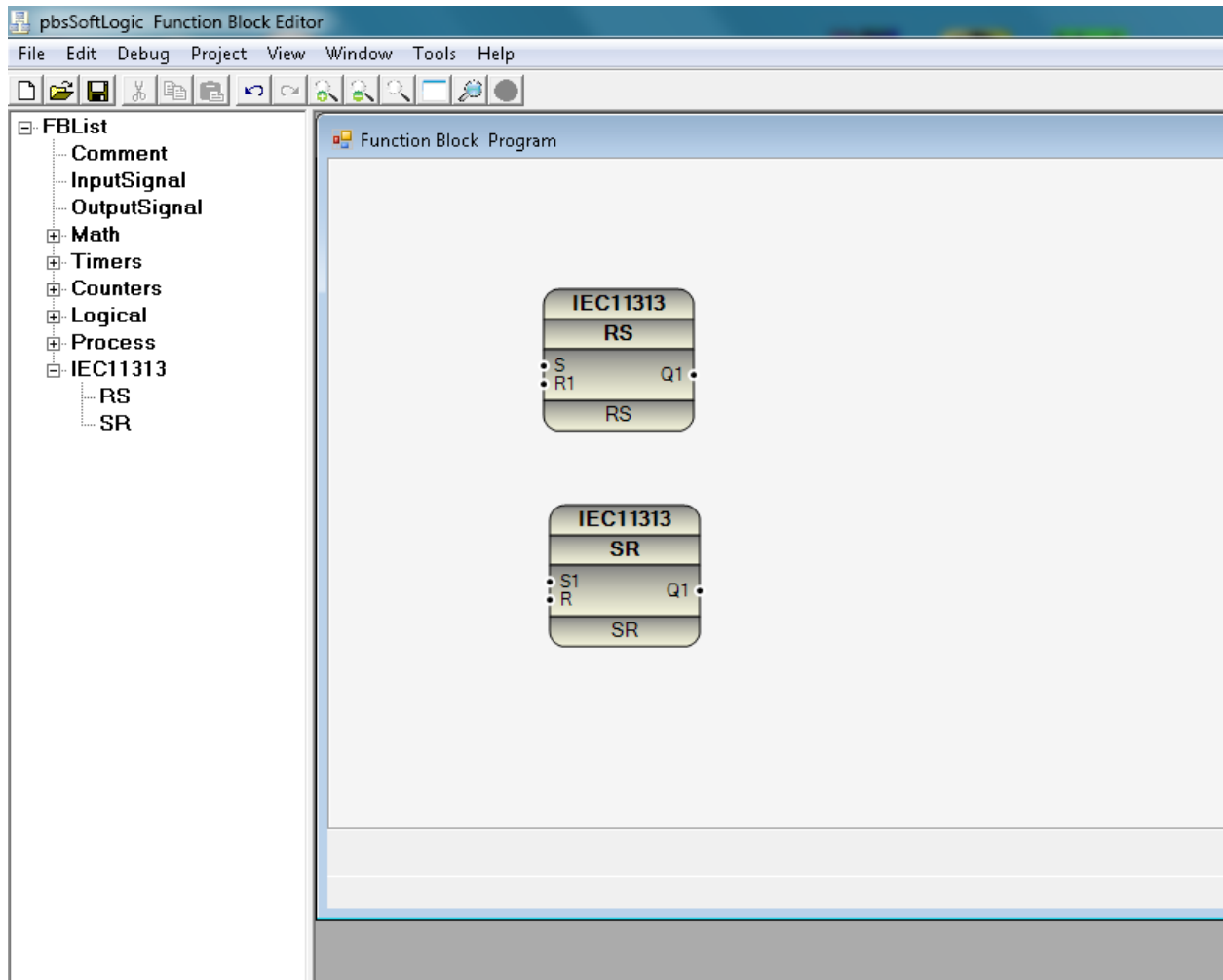
```
<?xml version="1.0"?>
<FBDef>
  <Version>1.0.0</Version>
  <FBDD>
    <Name>RS</Name>
    <Desc>RS Flip Flop</Desc>
    <Active>True</Active>
    <Interface></Interface>
    <InputList>
      <Input>
        <Name>S</Name>
        <Desc>Set Input</Desc>
        <Type>bool</Type>
        <Init>False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

19-1 Lua UDF Development

Lua scripting language is developed at 1993 by Roberto Ierusalimsky, 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.

19 – 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(Obj1 )

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

local Obj1o = {}

TmpPATH = Obj1["1"]
TmpPID = Obj1["2"]
TmpLogic = Obj1["3"]
TmpSRam = Obj1["4"]
TmpSD = Obj1["5"]

--read inputs
in1 = tonumber(Obj1["6"])
in2 = tonumber(Obj1["7"])

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

-- read Static data

-- Solev logic

--save static data

-- write outputs
Obj1o["1"] = tostring(out1)
Obj1o["2"] = tostring(out2)

return Obj1o
end

```

Obj1 = 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:

Obj1["1"] = path of RAMDisk Drive in Linux Controller for saving static data . for example it is like `"/mnt/ramdisk/"` it include `"/"` .

Obj1["2"] =unique Identifier of UDF .

Obj1["3"] = name of program. In Linux Kernel it is always "logic"

Obj1["4"] = SRAM address in controller . It is RAM with battery backup. It include `"/"`

Obj1["5"] =SD address . It is External flash SD card address for data logging. It include `"/"`

Points:

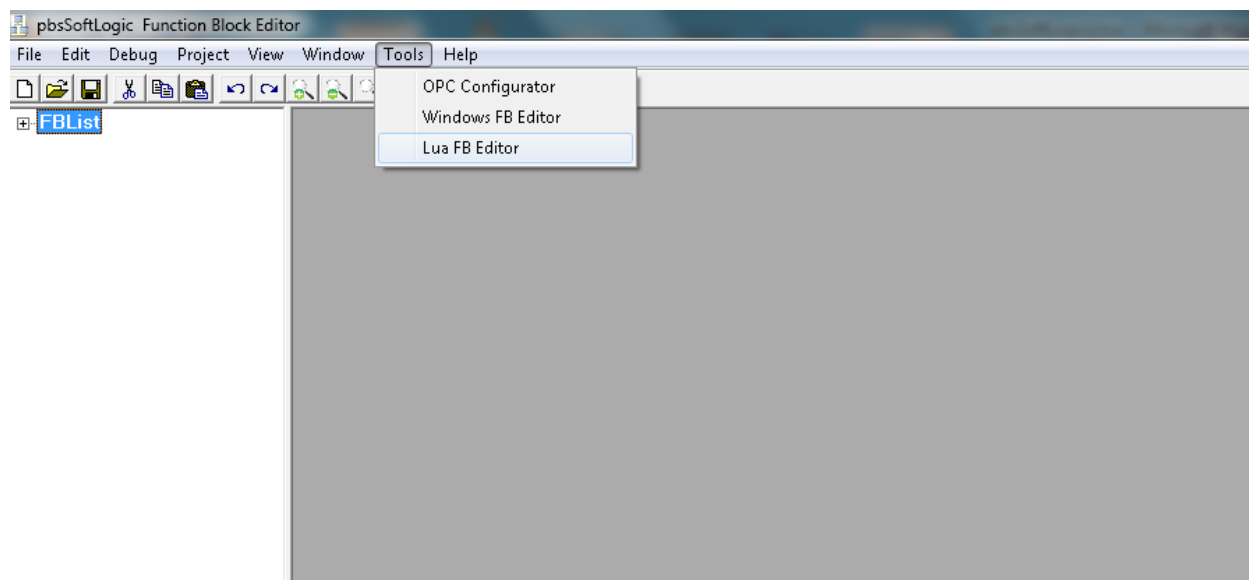
- UDF inputs start from key "6".
- All key value should be as string number: "1" ,"2" ,"3",...
- All inputs are pass as string to Lua . So you should change its type to number by tonumber function. Example in1 = tonumber(obj1["6"]) . This is value of first UDF input.

obj1o is return table from Lua.

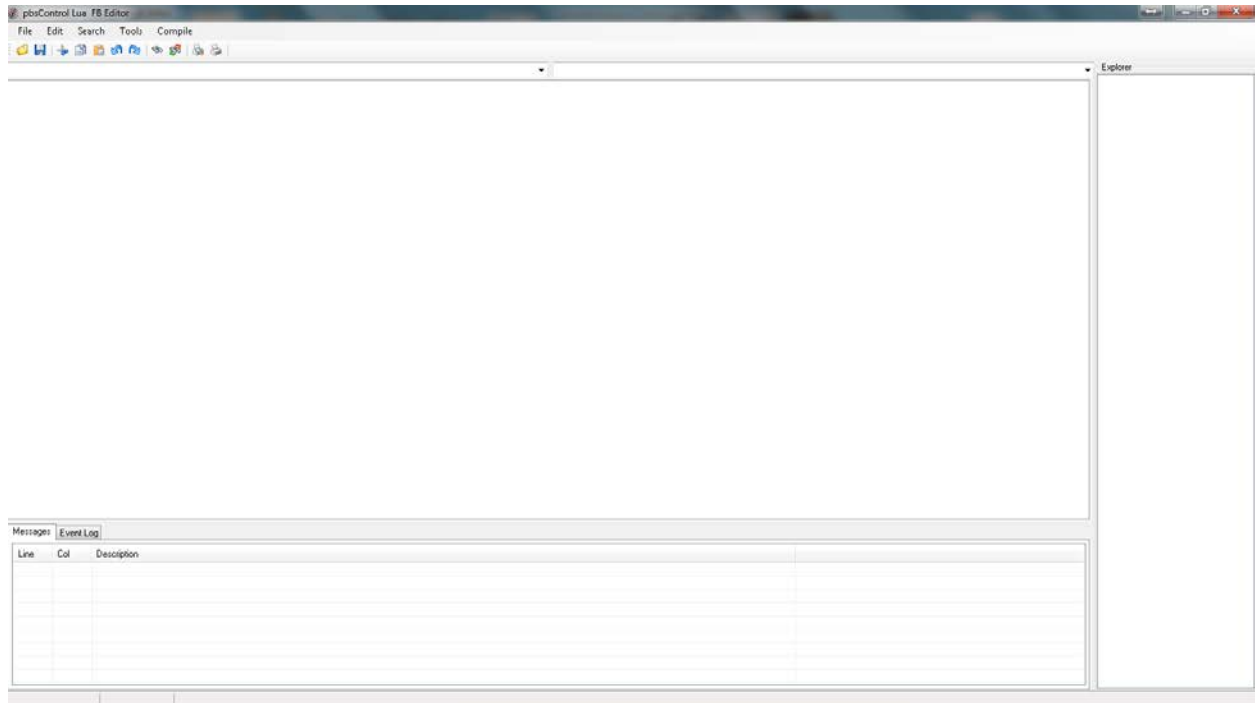
Points:

- 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:

```

1  function NewLua_Fun(Obj1)
2      -- Lua Function = Function without static data . There is no memory in function code.
3      local Obj1 = {}
4      --Define Output Variables
5      local out1=0
6      local out2=0
7      --add more output variables here
8      --Define Internal Variables
9      --Read Input variables
10     Input1 = tonumber(Obj1["6"])
11     Input2 = tonumber(Obj1["7"])
12     --add more input variables here
13     --Solve Logic
14     --Write your logic here
15     out1 = Input1 + Input2
16     out2 = Input1 - Input2
17     --Write outputs
18     Obj1["1"] = tostring(out1)
19     Obj1["2"] = tostring(out2)
20     return Obj1
21 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(Obj)
    -- Lua Function = Function without static data . There is no memory in function code.
    local Obj = {}
    --Define Output Variables
    local L=0
    --add more output variables here
    --Define Intrnal Variables
    --Read Input variables
    X = tonumber(Obj["6"])
    Y = tonumber(Obj["7"])
    --add more input variables here
    --Solve Logic
    --Write your logic here
    L = (X ^2 + Y^2)^0.5
    --Write outputs
    Obj["1"] = tostring(L)
    return Obj
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:

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

In this Lua FB sample, we consider following variables:

- Two output signal – ou1 , out2
- Three static signal – state , dt , input1old
 - o State shows current state of FB.
 - o 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 .
 - o input1old is used for detecting rising edge of input1 signal .

```

22  --Read Static data
23  local fr = io.open(TmpPATH .. "NewLua_FB_" .. TmpPID .. "_" .. TmpLogic .. ".dat" , "r")
24  if fr ~= nil then
25      while true do
26          line = fr:read()
27          if line ~= nil then
28              i , j = string.find(line, "=")
29              n = string.sub(line, 1, j-1)
30              if n == "state" then
31                  state = tonumber(string.sub(line, j+1))
32              end
33              if n == "dt" then
34                  dt = tonumber(string.sub(line, j+1))
35              end
36              if n == "input1old" then
37                  input1old = tonumber(string.sub(line, j+1))
38              end
39              -- add more static variables here
40              -- if n == "varstatic1" then
41                  -- varstatic1 = tonumber(string.sub(line, j+1))
42              -- end
43              if n == "out1" then
44                  out1 = tonumber(string.sub(line, j+1))
45              end
46              if n == "out2" then
47                  out2 = tonumber(string.sub(line, j+1))
48              end
49              -- All Output Variables should be Static
50              -- if n == "out3" then
51                  -- out3 = tonumber(string.sub(line, j+1))
52              -- end
53          else
54              break
55          end
56      end
57  fr:close()
58  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:  Obj1["1"] = tostring(out1)
91:  Obj1["2"] = tostring(out2)
92:  return Obj1
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 Input1 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
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:

$a \% b == a - \text{math.floor}(a/b)*b$ 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
```

19-3 Writing C code for Linux and cross compiling of UDF

You can develop your UDF by C Language too . Developing UDF by C has advantage and disadvantage.

Advantage : Performance of UDF with C Language is better than Lua Scripting .

Disadvantage : you should compile your UDF for each RTU and Operating systems .

For RTUs which is running Debian Based OS , you can compile UDF directly on the RTU .

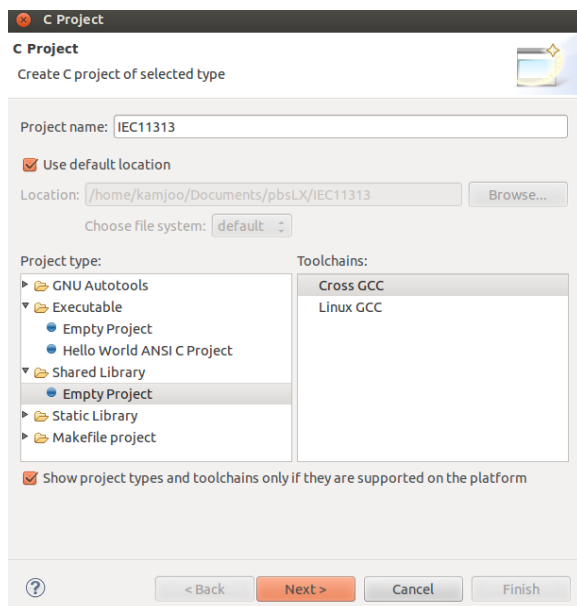
But for RTUs without gcc , you need to use cross compiler to compile UDF .

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 – Cross Compiler for your RTU .

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

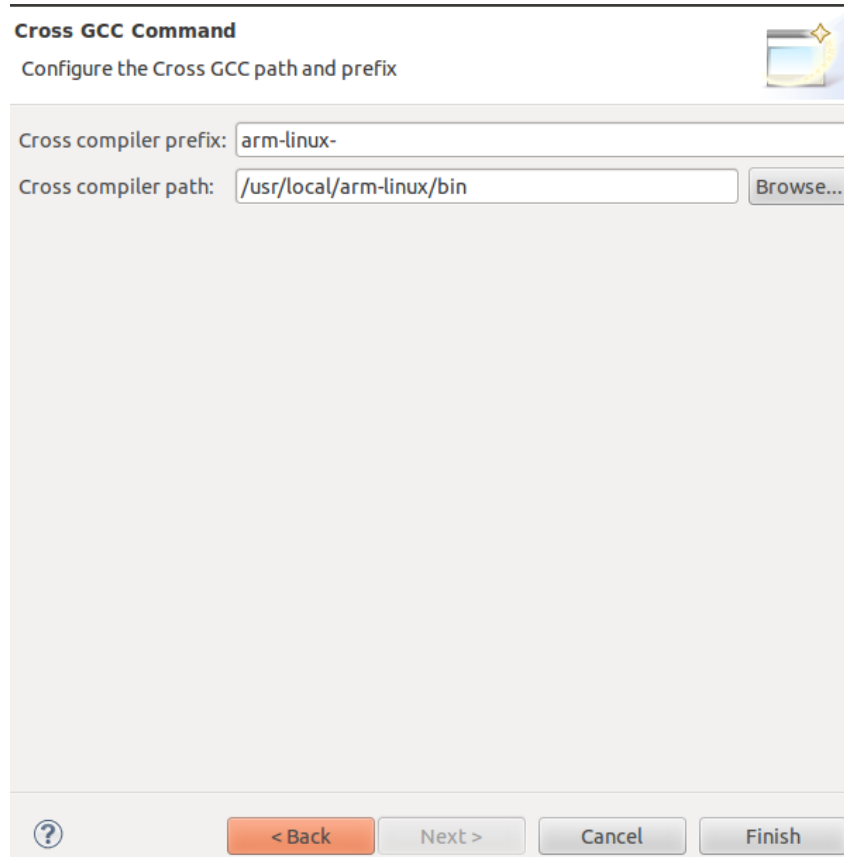
Open eclipse IDE and make a new C Project. Project name should be exactly same name of UDF group, for our example "IEC11313"



Project Type: Share Library

Toolchain : Cross GCC

Click on Next.



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

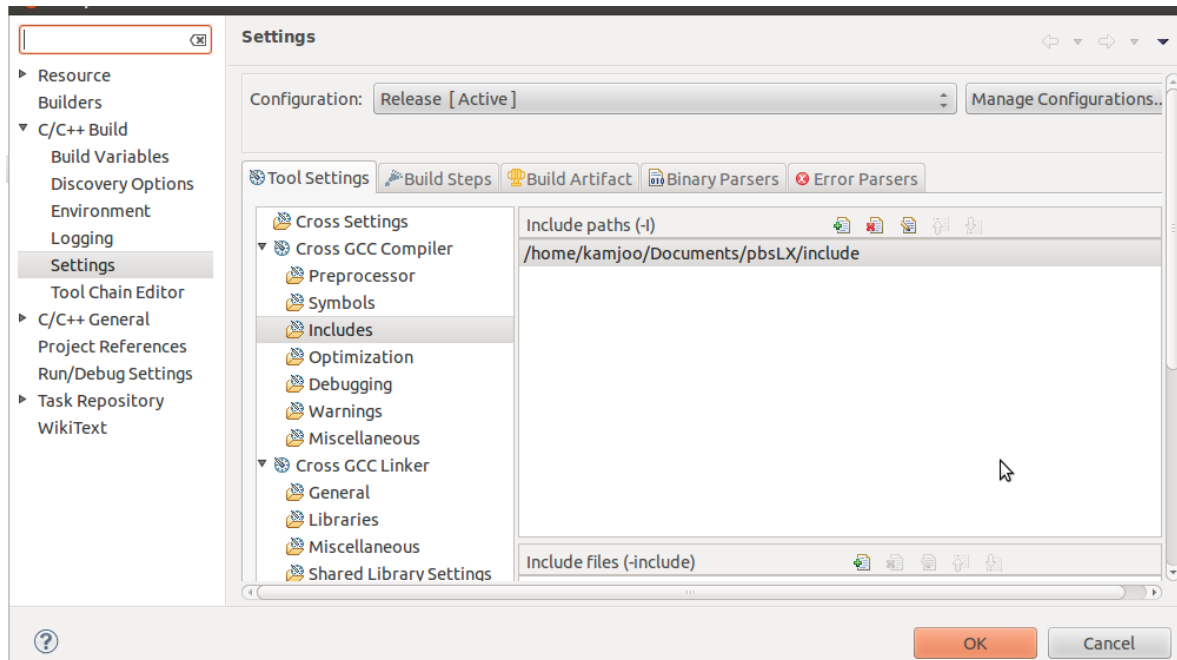
Click on finish button.

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 .

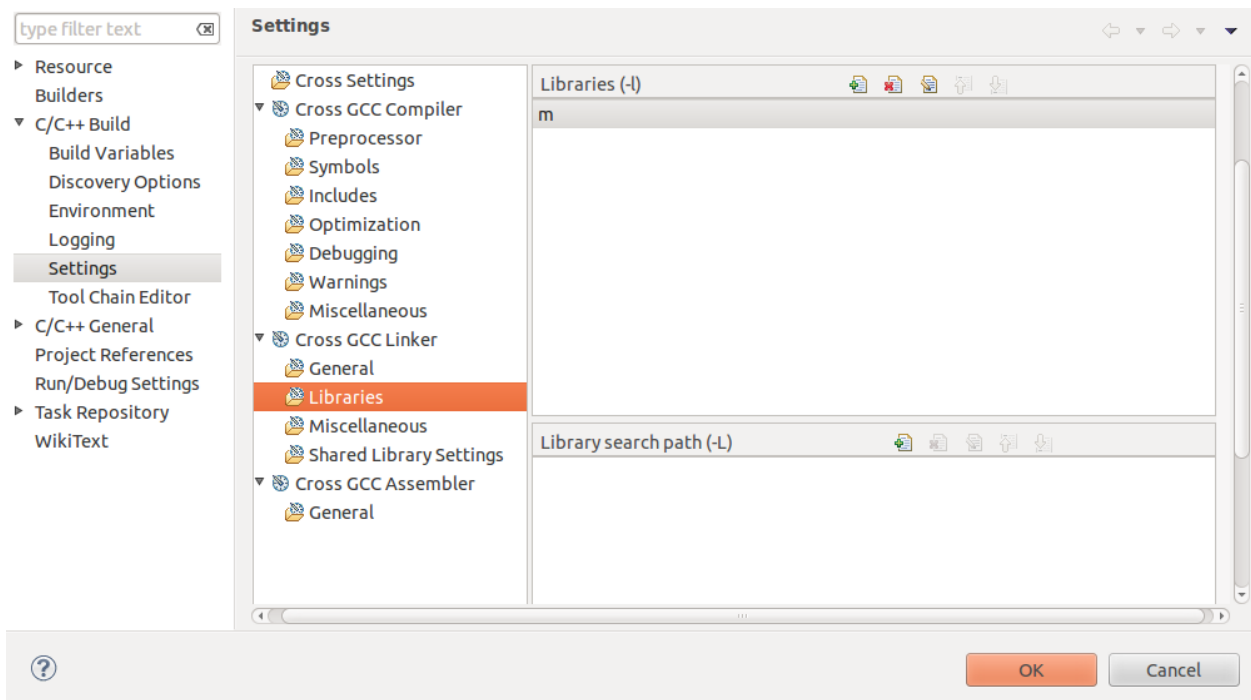
Select IEC11313 project in eclipse and refresh project to include MainIEC11313.c file to project .

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 .

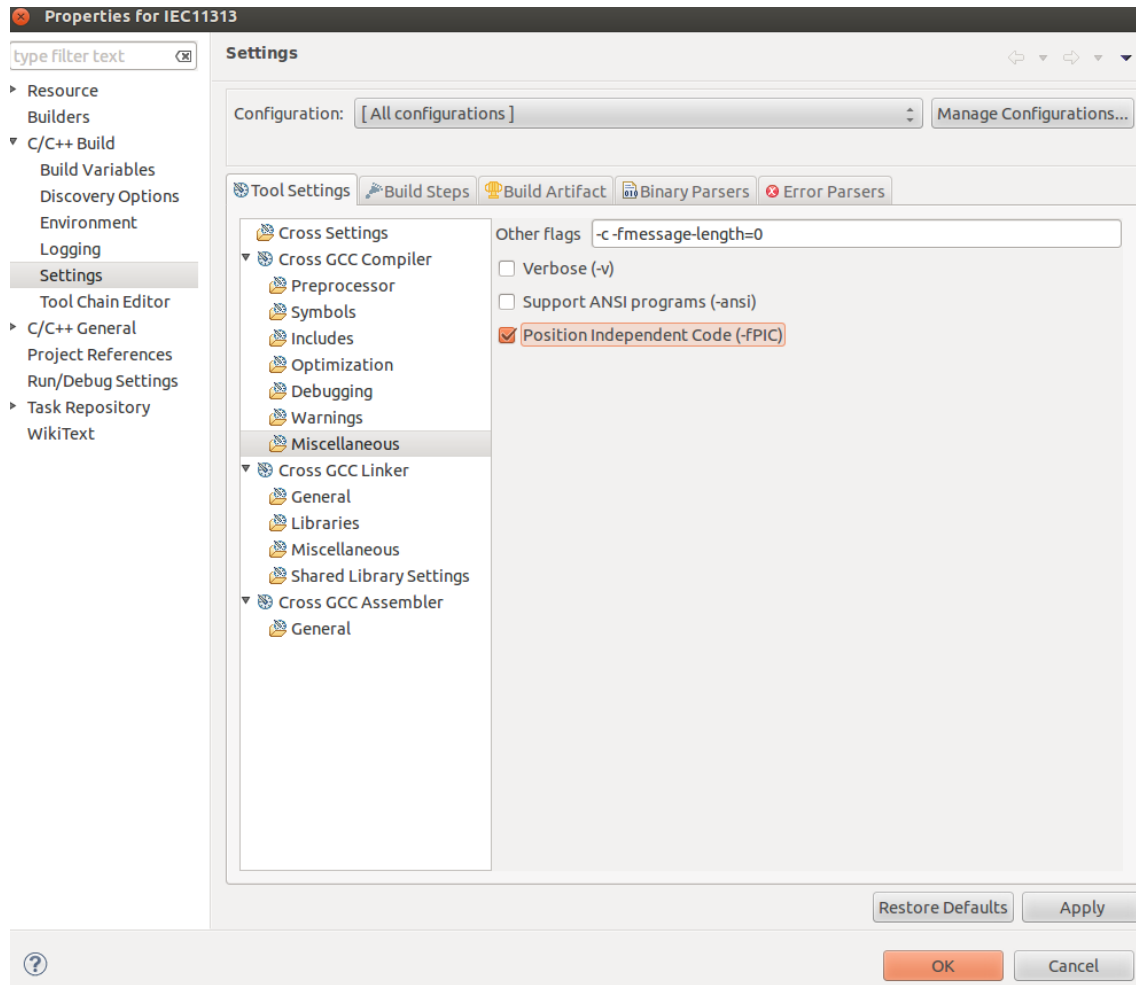


Select release mode as active mode in manage configuration. Add include directory for Debug and release configuration.

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 = Obj[i].dvalue;

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

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

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

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

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

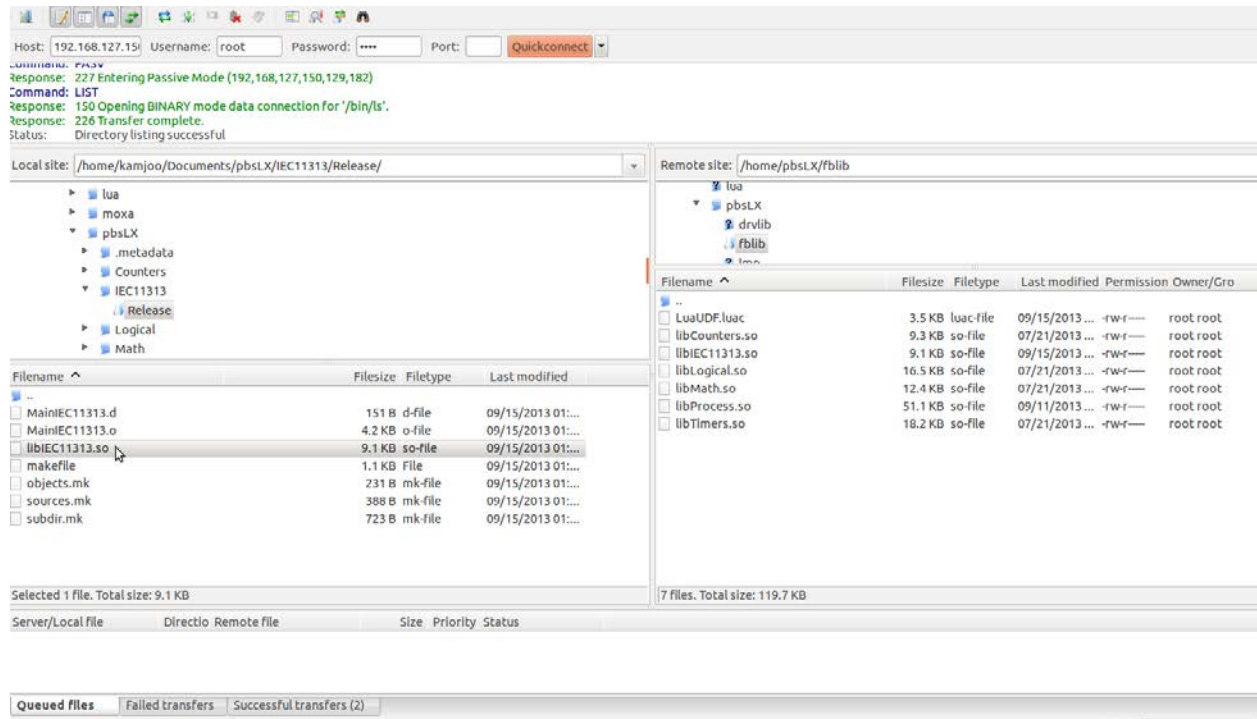
fclose(m_db);
```

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 .



Compiling UDF on Debian based RTUs :

When you make your UDF in previous step by Eclipse , it will make automatically all necessary make files . You can use same files or make a new project based on Linux GCC (Not Crossed GCC) in Ubuntu .

If you use same Cross compiling project for debian you need to remove all prefixes about your Cross compiler from make files transfer project to debian and make UDF by make command .

20 – pbsSDK :User defined Communication Driver Development

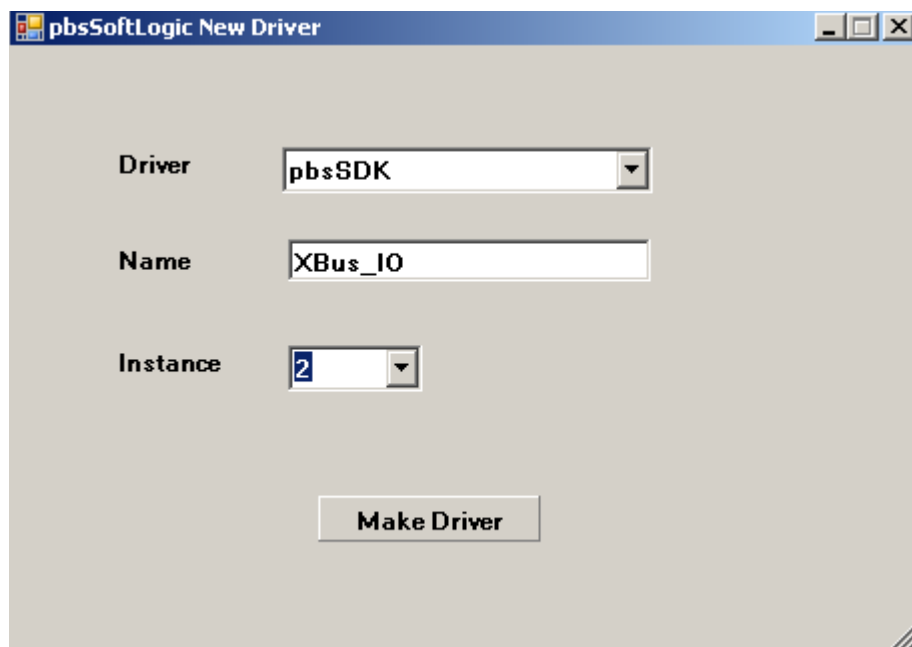
pbsSDK is Software development kit for developing user defined communication protocols in pbsSoftLogic .

If you want to communicate with devices with none standard communication protocols then you need to develop pbsSoftlogic driver by pbsSDK.

For different targets, you need different tools and compilers:

- WinCE: Visual studio 2005 with installed SDK for your hardware. DLL Module.
- Embedded Linux: Eclipse IDE and tool chain for your hardware. Loadable Library SO Module.
- Win32 : Visual studio 2008/2010 . DLL Module

Open project options in project menu. Define new I/O driver and select pbsSDK from driver list.



Suppose name of driver is XBus_IO and it has instance number 2.

You can define unlimited communication protocols by pbsSDK, but only 8 different instances can work on a controller. Suppose in a project you need following driver configuration:

- XBus_IO Instance = 1
- XBus_IO Instance = 2
- YBus_IO Instance = 3
- YBus_IO Instance = 4
- ZBus_IO Instance = 5

XBus , YBus and ZBus are developed based on pbsSDK interface and all of them should have unique Instance number . **Following configuration is not correct:**

- XBus_IO Instance = 1
- XBus_IO Instance = 2
- YBus_IO Instance = 1
- YBus_IO Instance = 2
- ZBus_IO Instance = 1

Click on “Make Driver” button. pbsSoftlogic will make new directory with XBus_IO in your project and will make two configuration files .

- Options.xml: you should use this file for passing different parameters to your C DLL module.
- pbsSDKTags.xml: will use for defining communications tags for using in your logic.

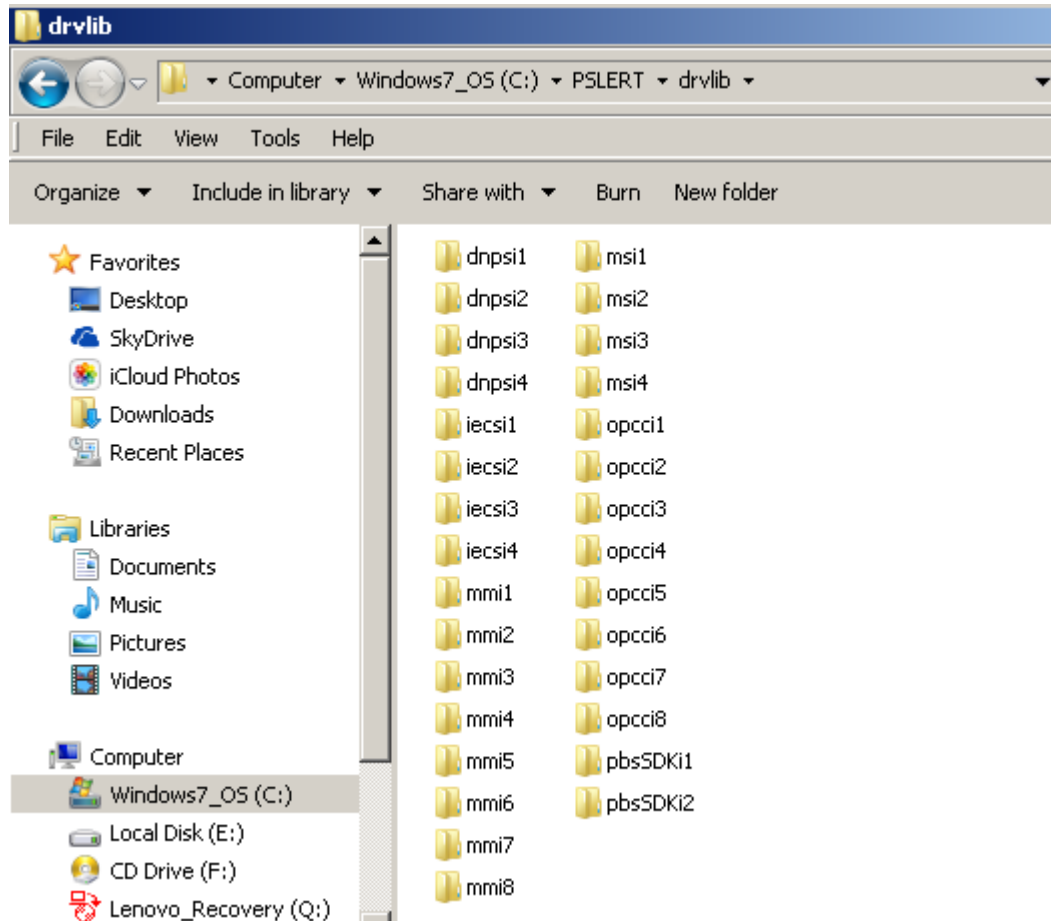
All Tags has same double type in pbsSoftLogic kernel.

Options.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Options>
  <Version>1.0.0</Version>
  <Node>
    <Name>DriverDirName</Name>
    <Desc>Driver Directory name Inside Controller</Desc>
    <Value>pbsSDK1</Value>
  </Node>
  <Node>
    <Name>DriverLibName</Name>
    <Desc>Name of Driver Library inside controller</Desc>
    <Value>pbsSDKLib1</Value>
  </Node>
  <Node>
    <Name>Instance</Name>
    <Desc>Instance</Desc>
    <Value>2</Value>
  </Node>
  <Param Name="Param1" Desc="Param1" Value="1" />
  <Param Name="Param2" Desc="Param2" Value="2" />
  <Param Name="Param3" Desc="Param3" Value="3" />
  <Param Name="Param4" Desc="Param4" Value="4" />
  <Param Name="Param5" Desc="Param5" Value="5" />
</Options>
```

DriverDirName , DriverLibName and Instance are not optional and all of them must be in options.xml file . Do not remove them and just change their value.

DriverDirName : Name of Directory inside controller . All pbssoftLogic drivers are located at \drvlib\ directory.



Dnpsi{n} = dnp3 slave driver for instance number n

iecsi{n} = iec870-5-101/104 slave driver for instance number n

mmi{n} = modbus master driver for instance number n

msi{n} = modbus slave driver for instance number n

opcci{n} = OPC client driver for instance number n

For pbsSDK driver you can use any name and set name in DriverDirName Parameter. Without I and {n}

Suppose you will name XBus_IO Driver directory as XBus : DriverDirName = XBus

Then you should make a directory inside \drvlib\ with XBusi1 for instance 1 .

DriverLibName: name of library inside \drvlib\XBusi1\ directory . Suppose your DLL name is XBUS.dll then you should change its name to XBusi1.dll and copy it inside \drvlib\XBusi1

For instance 3 you need following directory and library name:

\drvlib\xbusi3\xbusi3.dll

Instance: Driver instance number.

Param Tags are used for passing parameters to your driver. You can define any number of parameters and its format is under you control. pbsSoftLogic will pass all these parameters as string to runtime kernel and to your driver . So you can consider any format for parameters and their value.

Suppose XBus driver wants to communicate with two Devices (xbus is master and devices are slave) through RS485 and each device has unique ID .

You can set Parameters as following:

```
<Param Name="ComPort" Desc="Serial Communication Port Number " Value="1"/>
<Param Name="BaudRate" Desc="Communication BaudARte" Value="19200"/>
<Param Name="Mode" Desc="RS232-RS485-RS422" Value="RS485"/>
<Param Name="Parity" Desc="Even-Odd-None" Value="None"/>
<Param Name="ScanTime" Desc="Scan Time msec" Value="100"/>

<Param Name="Device1.System.ID" Desc="Device1.ID" Value="1"/>
<Param Name="Device1.Block1.Type" Desc="Device1.Block1.Type" Value="DI"/>
<Param Name="Device1.Block1.StartAdd" Desc="Device1.Block1.Startadd" Value="0"/>
<Param Name="Device1.Block1.Channels" Desc="Device1.Block1.Channels" Value="64"/>

<Param Name="Device1.Block2.Type" Desc="Device1.Block2.Type" Value="AI"/>
<Param Name="Device1.Block2.StartAdd" Desc="Device1.Block2.Startadd" Value="0"/>
<Param Name="Device1.Block2.Channels" Desc="Device1.Block2.Channels" Value="16"/>

<Param Name="Device2.System.ID" Desc="Device2.ID" Value="2"/>
<Param Name="Device2.Block1.Type" Desc="Device2.Block1.Type" Value="DO"/>
<Param Name="Device2.Block1.StartAdd" Desc="Device2.Block1.Startadd" Value="0"/>
<Param Name="Device2.Block1.Channels" Desc="Device2.Block1.Channels" Value="64"/>

<Param Name="Device2.Block2.Type" Desc="Device2.Block2.Type" Value="AO"/>
<Param Name="Device2.Block2.StartAdd" Desc="Device2.Block2.Startadd" Value="0"/>
<Param Name="Device2.Block2.Channels" Desc="Device2.Block2.Channels" Value="16"/>
```

pbsSoftLogic runtime kernel inside controller will pass all parameters with their value as string to user defined communication driver before initializing driver .

so you can easily detect communication parameters and device specifications insider your driver.

You can find sample driver code for above configuration in \psbssoftlogic\sdk directory.

pbsSDKtags.xml

You should define driver Tags in this file. As an example look at following tag definition:

```
<Version>1.0.0</Version>
<Tag Name="Device1.Block1.Tag1" Init="0" Address="1" />
<Tag Name="Device1.Block1.Tag2" Init="0" Address="2" />
<Tag Name="Device1.Block1.Tag3" Init="0" Address="3" />
<Tag Name="Device1.Block1.Tag4" Init="0" Address="4" />
<Tag Name="Device2.Block1.Tag5" Init="0" Address="5" />
<Tag Name="Device2.Block1.Tag6" Init="0" Address="6" />
<Tag Name="Device2.Block2.Tag7" Init="0" Address="7" />
<Tag Name="Device2.Block2.Tag8" Init="0" Address="8" />
```

Name format is under your control and you can select any format for tag name.

Address is Tag address and its type is int .

Init is Tag Init Value and its type is double.

You can define maximum 1024 Tags for each instance.

Driver C Interface for win32 and WinCE: for win32 and WinCE you need to make DLL module with following interface:

```
extern "C"
{
    __declspec(dllexport) int pbsInit();
    __declspec(dllexport) void pbsDown();
    __declspec(dllexport) int pbsAddTag( char * pTagName ,double pInitValue ,int pAddress );
    __declspec(dllexport) void pbsWriteTag(int pTagh , double pTagvalue );
    __declspec(dllexport) void pbsReadTag(int pTagh , double * pTagvalue);
    __declspec(dllexport) void pbsSetParamValueByName(char * pName , char * pParamValueStr);
};
```

1 - pbsSoftlogic runtime kernel will pass all parameters by pbsSetParamValueByName to your DLL Module .

2 – Driver tags will add to DLL module by pbsAddTag Function. This function will return Tag handle that runtime kernel will use it for Read/Write operation.

3 – Kernel will call pbsInit() function .

4 – Kernel will read /Write Tags in each Logic circle by pbsReadTag and pbsWriteTag functions.

You can find sample code for Win32 /WinCE and Linuc kernel at \pbssoftlogic\SDK directory.

21 – Standard Function Blocks Definition

pbsSoftLogic has many ready and tested Function Block for easy and fast programming.

There are following groups in pbsSoftLogic:

- **24.1 Math:** *Mathematics Function Groups*
- **24.2 Timers:** *Timer Function Blocks and Signal Generators*
- **24.3 Counters:** *Counters Function Blocks*
- **24.4 Logical:** *Logical Function Block Groups*
- **24.5 Process:** *High Level Process Function Blocks*
- **24.6 IEC11313:** *Standard Function Blocks based on IEC1131-3 standard*
- **24.7 Scheduling:** *Daily, Weekly, Monthly and Yearly Scheduling Function Blocks*