NAPOPC_CE6 DA Server

User's Manual

[Version 1.10]

(Supports 7000, 8000, 87000 series modules and Modbus controllers)

Warranty

All products manufactured by ICP DAS Inc. are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS Inc. assumes no liability for damages consequent to the use of this product. ICP DAS Inc. reserves the right to change this manual at any time without notice. The information furnished by ICP DAS Inc. is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS Inc. for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

License

The user can use, modify and backup this software on a single machine. The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

Table of Contents

1 NAPOPC_CE6 DA Server

What is NAPOPC_CE6 DA Server? NAPOPC_CE6 DA Server is an integrated omnibus software package which combines OPC, Modbus TCP, Modbus RTU services, and Scankernel together. The particular design, "Rule Script", lets user can quickly establish a DCS control system with logic control, multi-communication services.

For UI design, NAPOPC_CE6 uses an explorer-style user interface to display a hierarchical tree of modules and groups with their associated tags. A group can be defined as a subdirectory containing one or more tags. A module may have many subgroups of tags. All tags belong to their module when they are scanned to perform I/O. (The "OPC" stands for "OLE for Process Control" and the "DA" stands for "Data Access".)

For software use, NAPOPC_CE6 creates a set-up procedure requiring at most three steps for different kinds of users. This kind of procedure simplifies the designing process for the programmer, and ensures the stability and efficiency of control system.

NAPOPC_CE6 not only can map the physical I/O to a specific Modbus address automatically, but also allows users to define their own variables into it. Therefore users can develop their own application program with eVC++, VB.NET, and VC#.NET programming language via Modbus RTU and Modbus TCP protocol to share their specific data with Modbus client. Moreover, users can operate the NAPOPC_CE6 and NAPOPC_ST/NAPOPC_XPE in coordination to create a fantastic solution integrating SCADA software with on-line data.

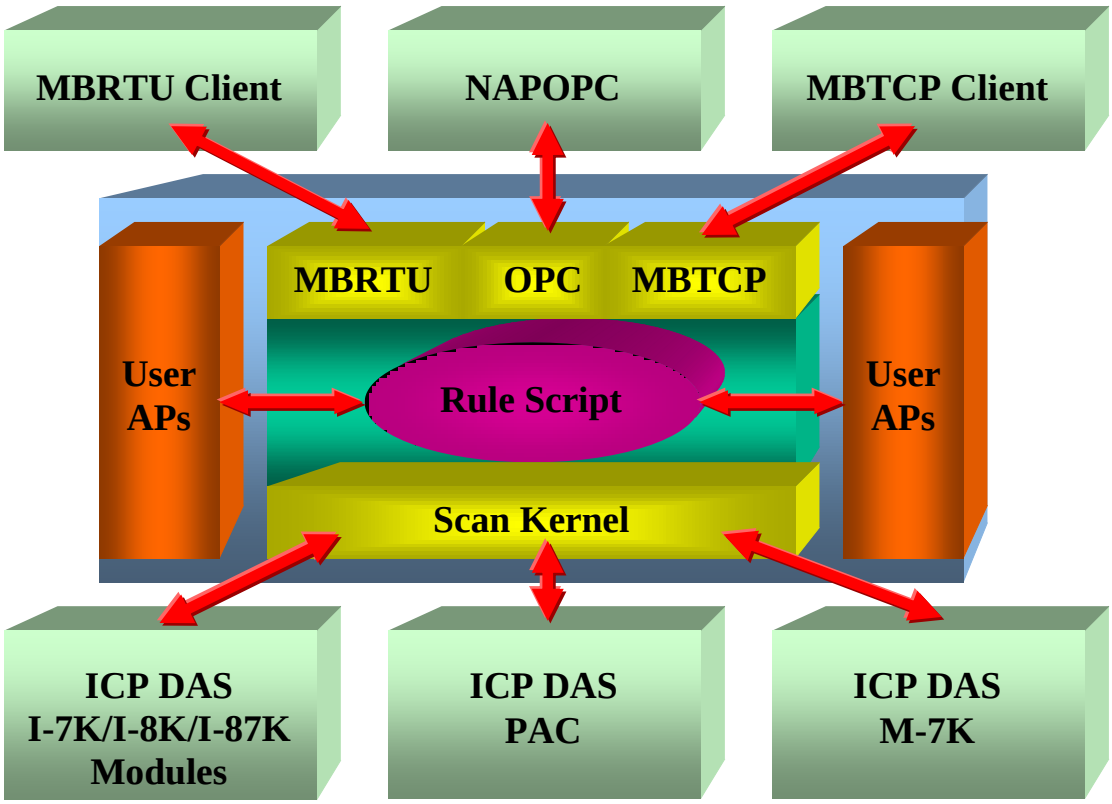


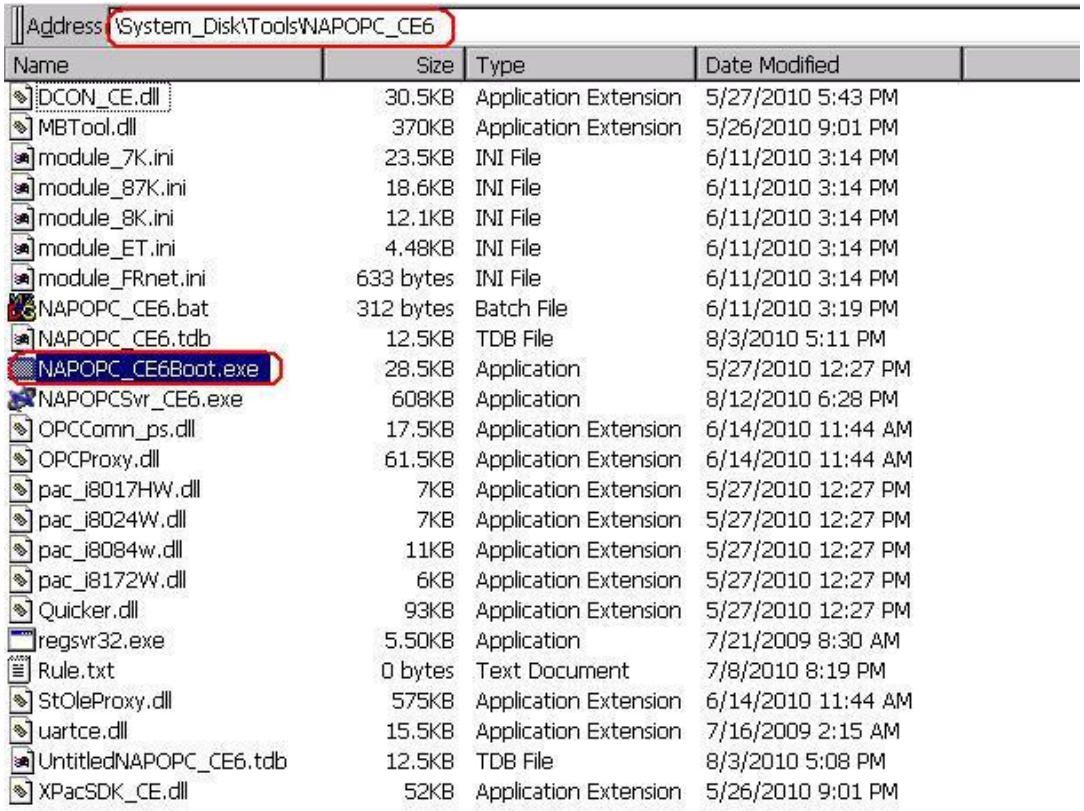
Fig 1-1

The main program of NAPOPC_CE6 is "NAPOPCSvr_CE6.exe". It automatically loads dynamic libraries under \System_Disk\Tools\NAPOPC_CE6 folder and calls functions in these libraries.

1.1 Install NAPOPC_CE6 DA Server

You have to execute "NAPOPC_CE6Boot.exe" in the \System_Disk\Tools\NAPOPC_CE6 of XP-8000-CE6 when you use NAPOPC_CE6 first time, after that, "NAPOPC_CE6Boot.exe" will register NAPOPC_CE6 automatically. Moreover, if you want to execute the " NAPOPCSvr_CE6.exe " automatically while XP-8000-CE6 boots up, please refer to the "Auto Execution" function at "2.4 XPAC Utility" of xp-8000-ce6_user_manual-vx.x.x.pdf and add path of " NAPOPC_CE6Boot.exe" into "Auto Execution".

NOTE: After above steps, please use "Save and Reboot" function at XPAC Utility to save registry exactly.



Name	Size	Type	Date Modified
DCON_CE.dll	30.5KB	Application Extension	5/27/2010 5:43 PM
MBTool.dll	370KB	Application Extension	5/26/2010 9:01 PM
module_7K.ini	23.5KB	INI File	6/11/2010 3:14 PM
module_87K.ini	18.6KB	INI File	6/11/2010 3:14 PM
module_8K.ini	12.1KB	INI File	6/11/2010 3:14 PM
module_ET.ini	4.48KB	INI File	6/11/2010 3:14 PM
module_FRnet.ini	633 bytes	INI File	6/11/2010 3:14 PM
NAPOPC_CE6.bat	312 bytes	Batch File	6/11/2010 3:19 PM
NAPOPC_CE6.tdb	12.5KB	TDB File	8/3/2010 5:11 PM
NAPOPC_CE6Boot.exe	28.5KB	Application	5/27/2010 12:27 PM
NAPOPCSvr_CE6.exe	608KB	Application	8/12/2010 6:28 PM
OPCComn_ps.dll	17.5KB	Application Extension	6/14/2010 11:44 AM
OPCProxy.dll	61.5KB	Application Extension	6/14/2010 11:44 AM
pac_i8017HW.dll	7KB	Application Extension	5/27/2010 12:27 PM
pac_i8024W.dll	7KB	Application Extension	5/27/2010 12:27 PM
pac_i8084w.dll	11KB	Application Extension	5/27/2010 12:27 PM
pac_i8172W.dll	6KB	Application Extension	5/27/2010 12:27 PM
Quicker.dll	93KB	Application Extension	5/27/2010 12:27 PM
regsvr32.exe	5.50KB	Application	7/21/2009 8:30 AM
Rule.txt	0 bytes	Text Document	7/8/2010 8:19 PM
StOleProxy.dll	575KB	Application Extension	6/14/2010 11:44 AM
uartce.dll	15.5KB	Application Extension	7/16/2009 2:15 AM
UntitledNAPOPC_CE6.tdb	12.5KB	TDB File	8/3/2010 5:08 PM
XPacSDK_CE.dll	52KB	Application Extension	5/26/2010 9:01 PM

Fig 1.1-1

After that, you execute the main program " NAPOPCSvr_CE6.exe " which would load dynamic libraries under \System_Disk\Tools\NAPOPC_CE6 folder by itself to start NAPOPC_CE6.

If the files under "\System_Disk\Tools\NAPOPC_CE6" loss or crash, please copy the files under "//xp-8000-ce6/tools/" in the CD to "\System_Disk\Tools\NAPOPC_CE6" by yourself.

1.2 Function Overview

1.2.1 Search Modules

The "Search Modules…" function lets you configure NAPOPC_CE6 automatically. It searches the RS-485 network and embedded modules to find modules and then generates tags automatically. This version of NAPOPC_CE6 not only generates AI/AO, DI/DO, Latched DI and Counter tags but also maps each tag to a unique modbus address.

Step 1: Click on the "Add/ Search Modules…" menu item or the  icon to search for modules.

Fig 1.2.1-1

Step 2: The "Search Modules" window pops up.

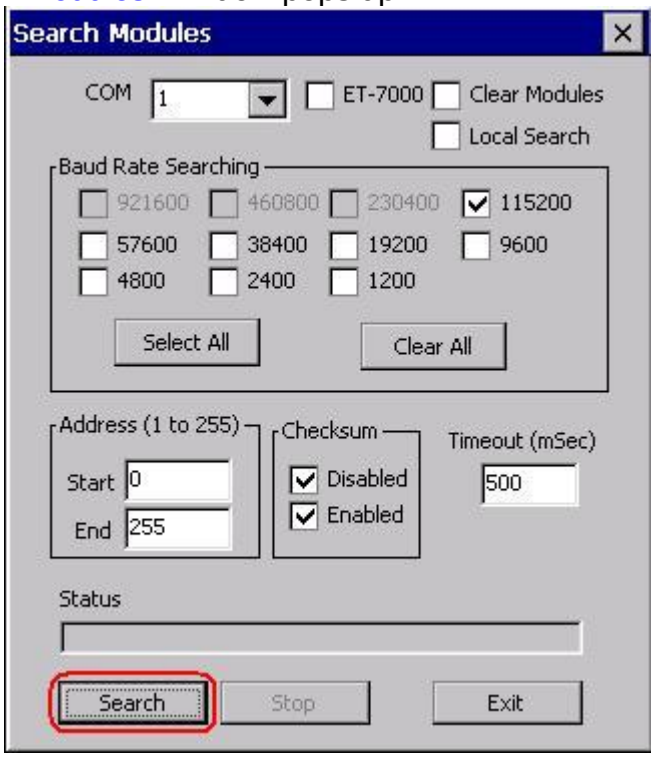


Fig 1.2.1-2

Step 3: If you want to search the I-8K I/O modules plugged in the XP-8000-CE6, you have to check the "Local Search" field. "COM 1" is for searching I-87K I/O modules plugged in the XP-8000-CE6.

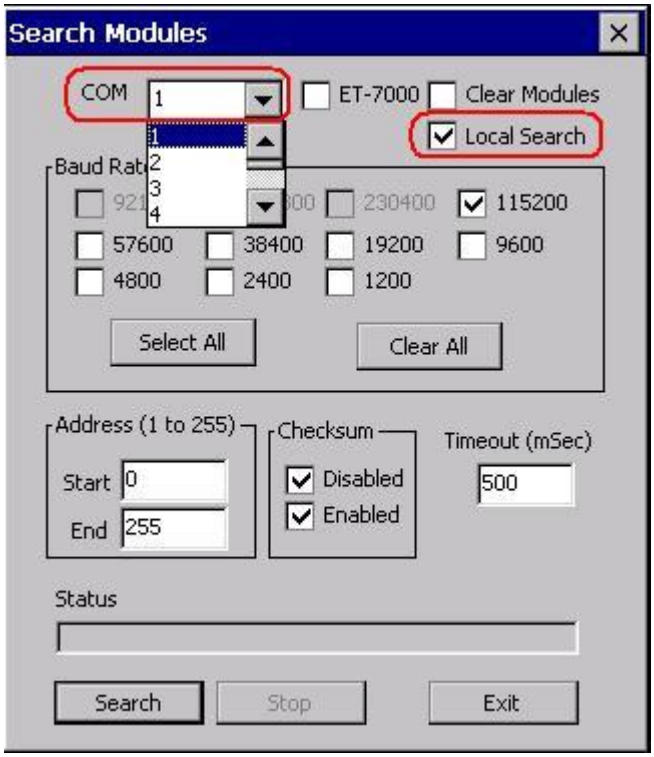


Fig 1.2.1-3

Step 4: If you want to search the I-7K/I-87K remote I/O modules via RS-232, you have to choice "COM 2" and uncheck the "Local Search".

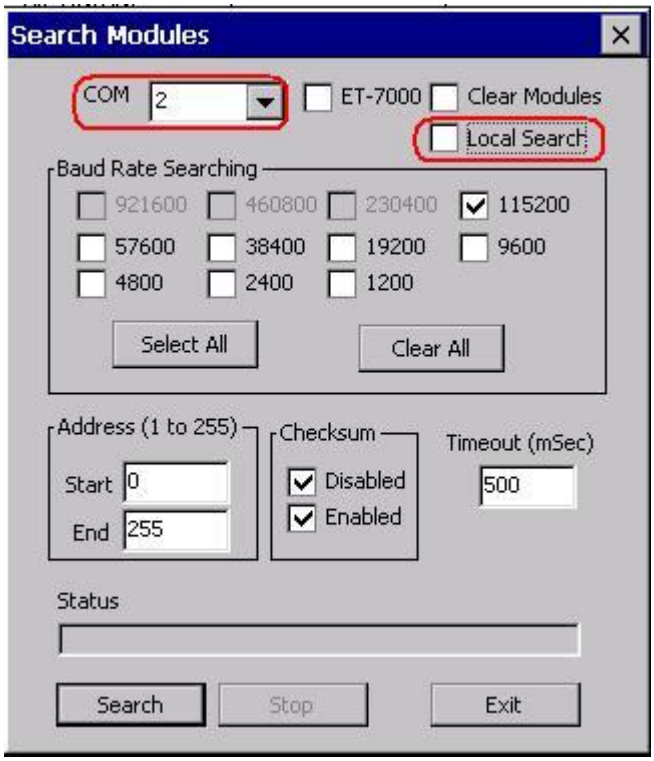


Fig 1.2.1-4

Step 5: If you want to search the I-7K/I-87K remote I/O modules via RS-485 and ET-7000 modules via Ethernet, you have to choice "COM 3" and "ET-7000", and uncheck the "Local Search".

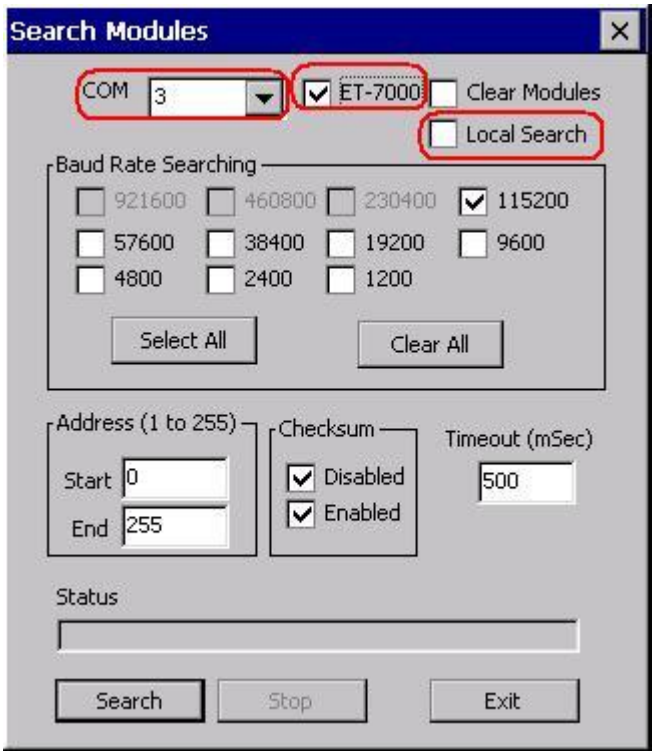


Fig 1.2.1-5

COM :

Specifies which "COM" port number to search. The default value is 0 and the valid range is from 0 to 255. Please verify the "COM" port number that the RS-485 network is connected to.

Modules	COM 1	COM 2	COM 3	COM 4	COM 5
Local I-87K	Yes	-	-	-	-
Remote I-7K/I-87K via RS-232	-	Yes	-	-	-
Remote I-7K/I-87K via RS-485	-	-	Yes	-	-
Remote I-7K/I-87K via RS-232/485	-	-	-	Yes	-
Remote I-7K/I-87K via RS-232	-	-	-	-	Yes

ET-7000:

If this field is checked, NAPOPC can search not only the modules communicating via COM port but also ET-7000 modules via Ethernet automatically.

Clear Modules:

Modules can be added many times. If this field is checked, it removes all modules from the list window before searching. Checking this box prevents adding a duplicate module. The default setting is "not checked".

Local Search:

If this field is checked, it searches the I-8K modules plugged in the XP-8000-CE6 first.

Baud Rate Searching:

Specifies which "Baud Rate" will be looking for. The default setting is "9600".

Naturally, if multiple baud rates are checked, the search will be longer. NAPOPC_CE6 has to close and then reopen the COM ports to communicate with modules when searching for multiple baud rates. This also reduces communication performance. Thus, using the same baud rate and COM port number for every module is highly recommended.

Select All:

Sets all the "Baud Rate" fields to be checked. Please refer to the above "Baud Rate Searching" section.

Clear All:

Sets all the "Baud Rate" fields to be unchecked (nothing to search). Please refer to the above "Baud Rate Searching" section.

Address/Start:

Specifies the starting address. The default value is 0 and the valid range is from 0 to 255. It won't search for an address below these settings.

Address/End:

Specifies the ending address. The default value is 255 and the valid range is from 0 to 255. It won't search for an address greater than these settings.

Checksum/Disabled:

If this field is checked, modules are searched with no checksum. If both the "Disabled" and "Enabled" fields were unchecked, the search would be undefined.

Checksum/Enabled:

If this field is checked, it searches modules with checksum. If both the "Disabled" and "Enabled" fields were unchecked, again, the search would be undefined.

Timeout:

Specifies the timeout value of communication to each module. The default value is 200 (equal to 0.2 Seconds), measured in millisecond(s) [0.001 Second(s)]. After a module has been found, this timeout value will also be recorded for further use.

Users can reduce this value to shorten the search time. Be careful. A shorter search time may cause communication failure.

Status:

It shows the searching status (includes: progress in %, Address in "A:??", Baud-Rate in "B:????", Checksum in "S:?" and Error-Code in "EC:??").

The timeout error code is 15. In most cases, it indicates no module has responded to the current command.

Search:

After setting the above options, click this button to search. The window will be closed automatically when completed.

Stop:

During the search, users can click the button to stop. The window will stay on the screen after the search is cancelled.

Exit:

Users can click the button to close the window.

Step 6: After the search, the discovered modules will be listed on the Device-Window (left side). Users can also see the tags on the Tag-Window (right side) generated by the "Search Modules…" function automatically.

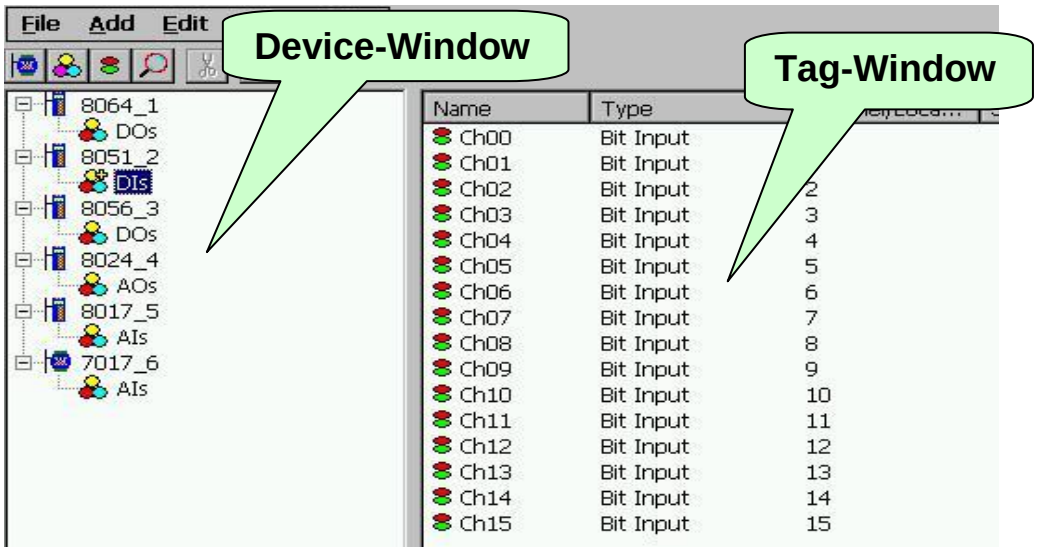


Fig 1.2.1-6

The "Search Modules…" function generates "Digital Input", "Digital Output" "Bit Input" or "Bit Output" tags.

The "Digital Input" and "Digital Output" tags use one communication to read the status of all channels, while the "Bit Input" and "Bit Output" tags use one communication to read only one-channel status. The "Digital Input" and "Digital Output" tags have better performance than the "Bit Input" and "Bit Output" tags. Using the "Digital Input" and "Digital Output" tags to access modules is highly recommended.

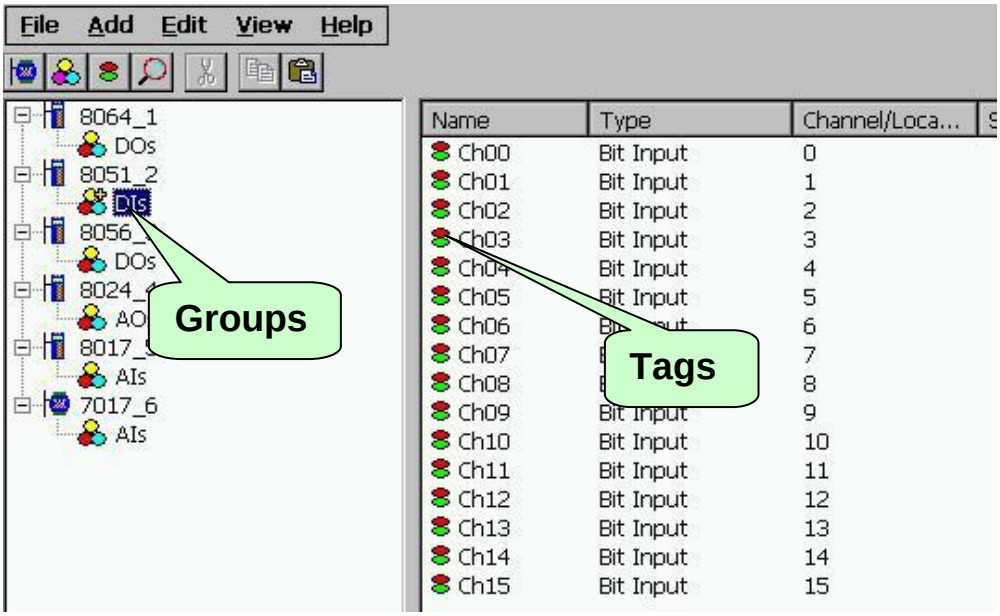


Fig 1.2.1-7

1.2.2 Monitoring Devices

Use the "Monitor" function to see values of tags by checking the "View/ Monitor" menu item. Uncheck the item to stop monitoring.

Step 1: Click the "View/ Monitor" menu item to enable monitor.

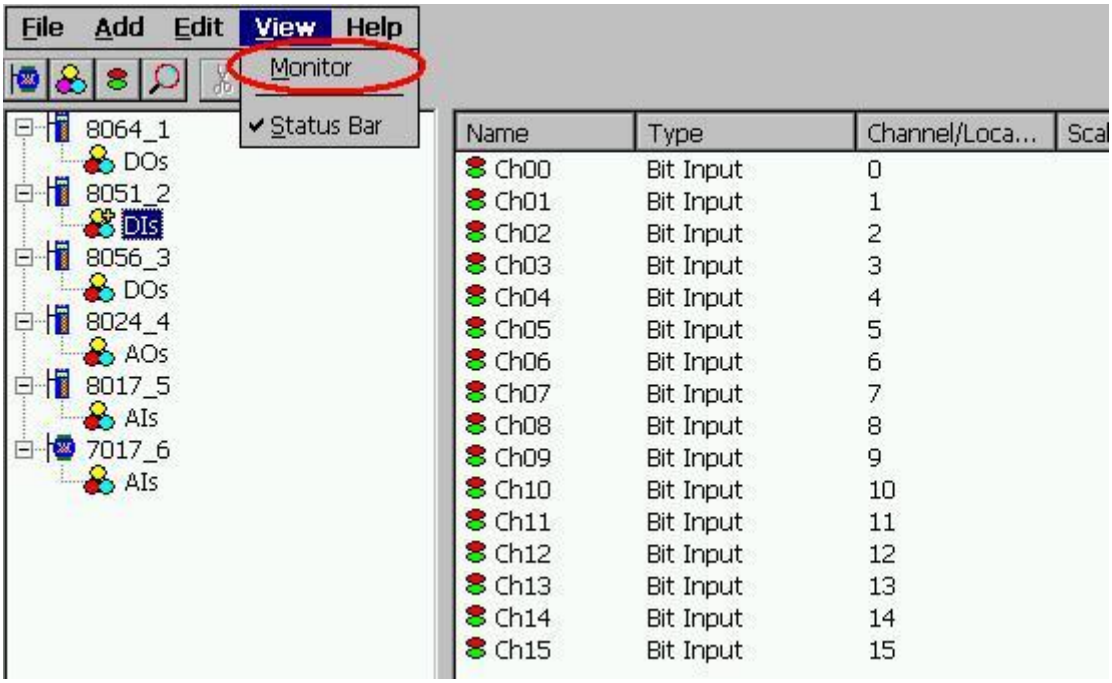


Fig 1.2.2-1

Step 2: Select the "AIs" group in the Device-Window (left side) to monitor its own Analog -Input tags.

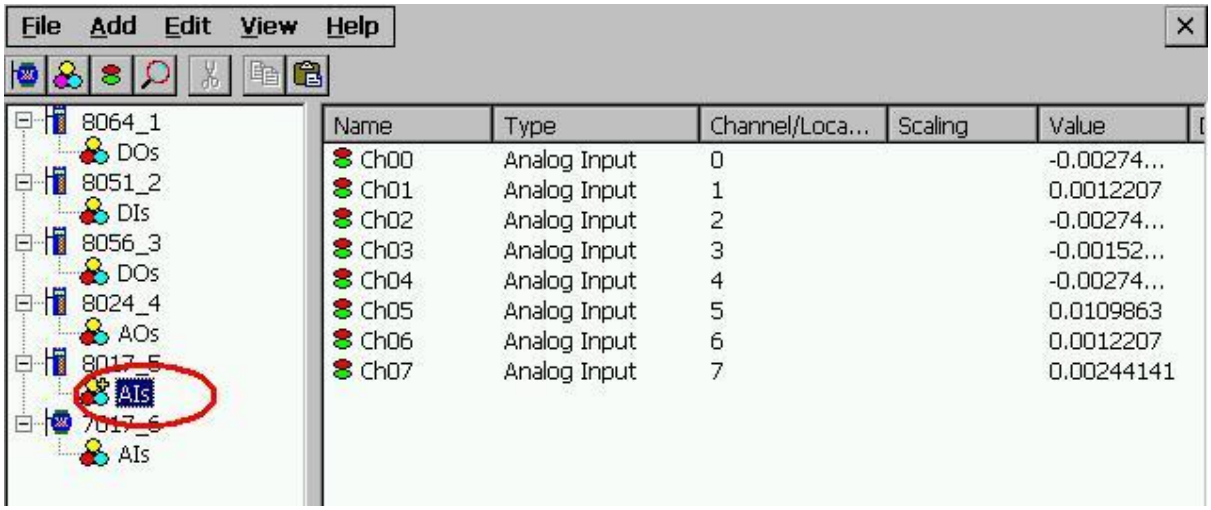


Fig 1.2.2-2

Step 3: Select the "8064" module on the Device-Window to monitor its own Digital-Output tags.

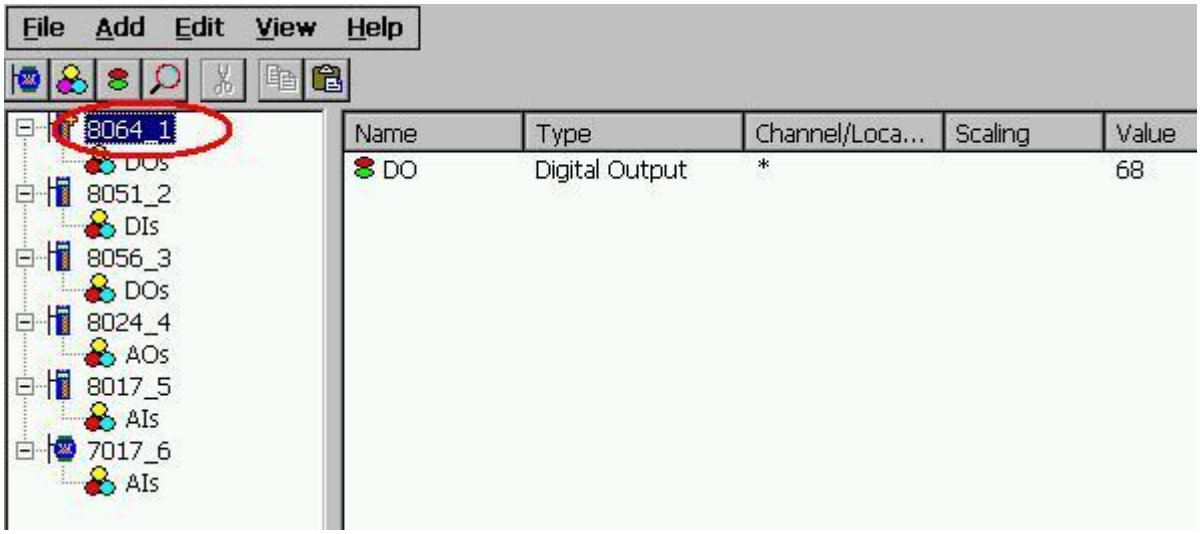


Fig 1.2.2-3

1.2.3 Adding a New Device

NAPOPC_CE6 provides three kinds of device, "DCON Device", "FRnet Device", and "Modbus Device" to be added. The "DCON Device" includes "I-8K/87K Embedded Modules", "Remote I/O Modules", and "Internal Device". The "Internal Device" could be the intermediary container between several user application programs or the intermediary device designing "Rule Script". The "FRnet Device" supports ICP DAS FRnet modules. The "Modbus Device" supports "Modbus RTU", "Modbus ASCII", and "Modbus TCP" protocol. NAPOPC_CE6 provides multi-thread communication via COM port and Ethernet. The maximum number of Modbus TCP master communication thread is limited to 32 by default.

1.2.3.1 Adding a New I-8K/I-87K Embedded Module

Step 1: Click on the "Add/ New Device…" menu item or the  icon to add a new module.

Fig 1.2.3.1-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "DCON" radio button.

Step 4: Click the "I-8K/I-87K Embedded Modules" radio button.

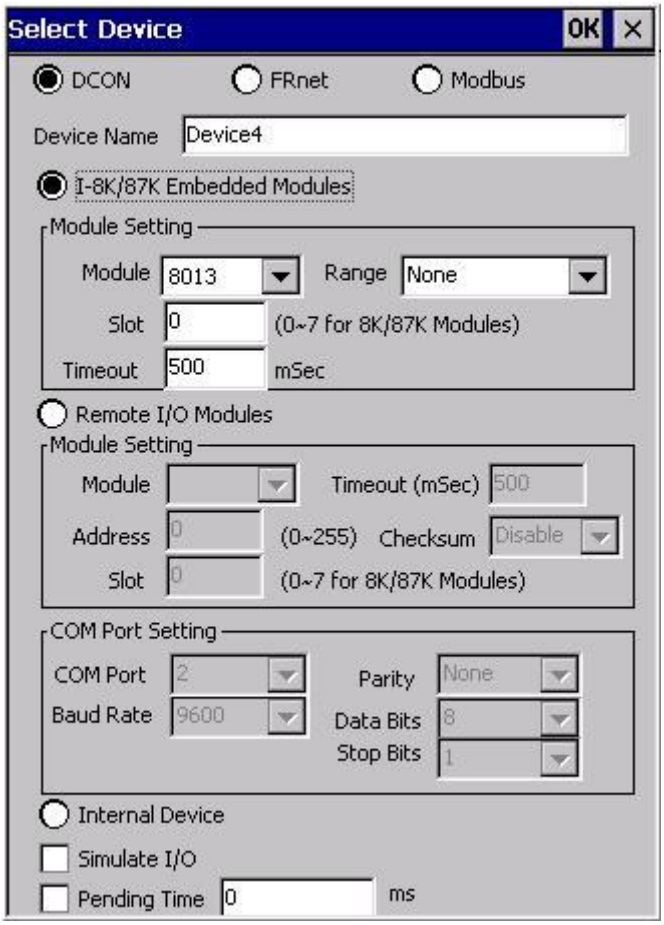


Fig 1.2.3.1-2

Device Name:

Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

Module:

User can click on the ComboBox to select a Module ID.

Timeout:

Specifies timeout (Response time) value for this module. A smaller timeout value may cause communication failure and a greater timeout value may reduce the performance of the client program.

Slot:

The XP-8000-CE6 has 3 or 7 slots to plug in. This "slot" field indicates the slot number that the I/O module used. The valid range is from 1 to 7.

Range:

It is for I-8017 and I-8024 module settings. Please refer to module manual to choose correct range.

Simulate I/O:

The "Simulate I/O" checkbox switches to a simulator of reading I/O. Since the simulator does not open the TCP/IP or COM port, it is an easy way to work with the server, to configure tags or to connect clients without requiring any hardware.

Pending Time:

Minimum interval time between two access. To activate this function, NAPOPC_CE6 can work under optimized communication performance. If this module only needs to be accessed 1 time per 5 seconds. You can set pending time as 5000 ms. NAPOPC_CE6 will automatically spread time resource to other modules which are connected with each other.

Step 5: Click on the "OK" button to add this new module.

1.2.3.2 Adding a New Remote I/O Module

Step 1: Click on the "Add/ New Device…" menu item or the  icon to add a new module.

Fig 1.2.3.2-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "DCON" radio button.

Step 4: Click the "Remote I/O Modules" radio button.

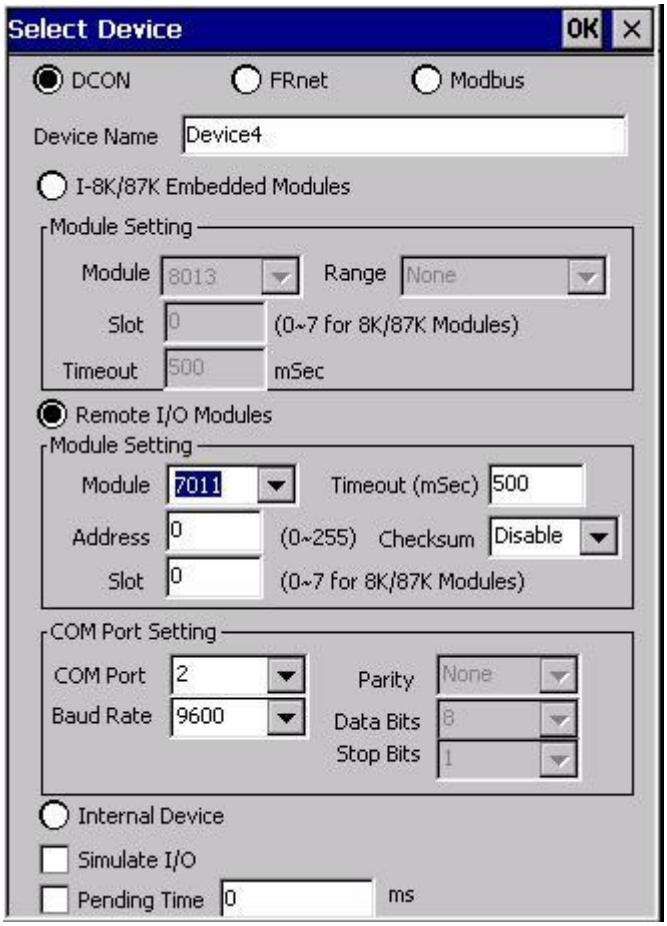


Fig 1.2.3.2-2

Device Name:

Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

Module:

User can click on the ComboBox to select a Module ID.

Address:

Specifies a Module Address for this module. The default value is 0 and the valid range is between 0 to 255. This field is disabled for the 8000 sub-devices. It will use the 8000 main-device's address.

Slot:

The 8000 main-device has 4 or 8 slots for the 8000 sub-device to plug in. This "slot" field indicates the slot number that the 8000 sub-device is using. The valid range is from 0 to 7.This field is disabled for 8000 main-device and 7000 series modules.

Timeout:

Specifies timeout (Response time) value for this module. A smaller timeout value may cause communication failure and a greater timeout value may reduce the performance of the client program. This field is disabled for the 8000 sub-devices and it will use the 8000 main-device's timeout value.

Checksum:

This checksum field must match the hardware setting. A mismatch will always cause a communication failure with this module.

This field is disabled for the 8000 sub-devices and it will use the 8000 main-device's checksum.

COM Port:

Specifies the COM port to be used. Please verfiy which COM port number that the RS-485 network is using. Wrong settings will always cause communication failure.

This field is disabled for the 8000 sub-devices. It will use the 8000 main-device's COM port setting.

Baud Rate:

Specifies the baud rate to be used. Verify the module's current baud rate. A wrong setting will always cause communication error for this module.

This field is disabled for the 8000 sub-devices. It will use the 8000 main-device's baud rate.

Simulate I/O:

The "Simulate I/O" checkbox switches to a simulator of reading I/O. Since the simulator does not open the TCP/IP or COM port, it is an easy way to work with the server, to configure tags or to connect clients without requiring any hardware. This field is disabled and not used for the 8000 main-device.

Pending Time:

Minimum interval time between two access. To activate this function, NAPOPC_CE6 can work under optimized communication performance. If this module only needs to be accessed 1 time per 5 seconds. You can set pending time as 5000 ms. NAPOPC_CE6 will automatically spread time resource to other modules which are connected with each other.

OK:

Click on the "OK" button to add the new module setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

Step 5: Click on the "OK" button to add this new module.

1.2.3.3 Adding a New Internal Device

Step 1: Click on the "Add/ New Device…" menu item or the  icon to add a new module.

Fig 1.2.3.3-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "DCON" radio button.

Step 4: Click on the "Internal Device" radio button.

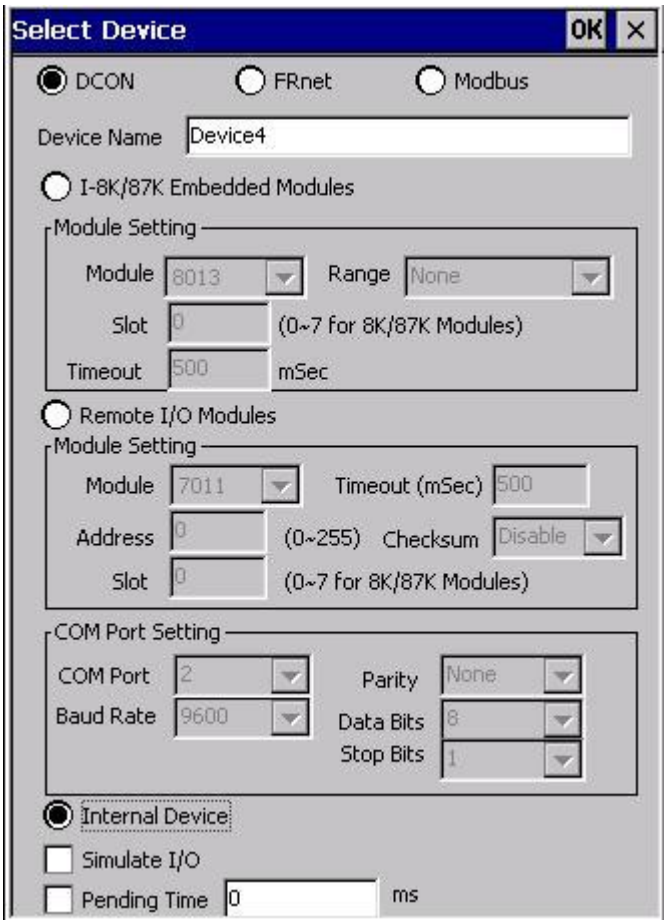


Fig 1.2.3.3-2

Device Name:
Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

Step 5: Click on the "OK" button to add this new module.

1.2.3.4 Adding a New FRnet Device

Step 1: Click on the "Add/ New Device…" menu item or the icon to add a new module.

Fig 1.2.3.4-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "FRnet" radio button.

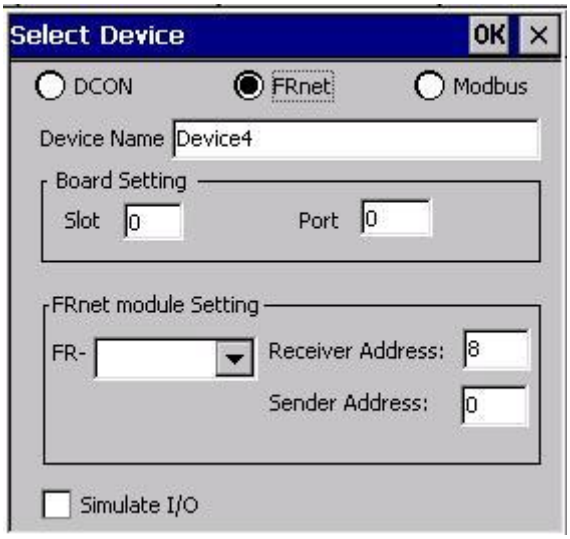


Fig 1.2.3.4-2

Device Name:

Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

Slot:

The XP-8000-CE6 has 3 or 7 slots to plug in. This "slot" field indicates the slot number that the I/O module used. The valid range is from 1 to 7.

Port:

The "Port" indicates the port number(0 or 1) of I-8172. Every FRnet I/O modules have to use I-8172 as FRnet communication module. Please refer to the I-8172 manual for more information.

FR-:

User can click on the Combo Box to select a FRnet module ID.

Receiver Address:

FRnet communication needs correct hardware configurations for the sender address (SA) and receiver address (RA) on the host controller and the remote module in the network. Please refer to the FRnet manual for more information.

Sender Address:

FRnet communication needs correct hardware configurations for the sender address (SA) and receiver address (RA) on the host controller and the remote module in the network. Please refer to the FRnet manual for more information..

Simulate I/O:

The "Simulate I/O" checkbox switches from reading I/O from the module to running a simulator. Since the simulator does not open the COM port, it is an easy way to work with the server, to configure tags or to connect clients without requiring any hardware.

1.2.3.5 Adding a New Modbus RTU Controller

Step 1: Click on the "Add/ New Device…" menu item or the  icon to add a new module.

Fig 1.2.3.5-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "Modbus" radio button.

Step 4: Click on the "Modbus RTU" radio button.

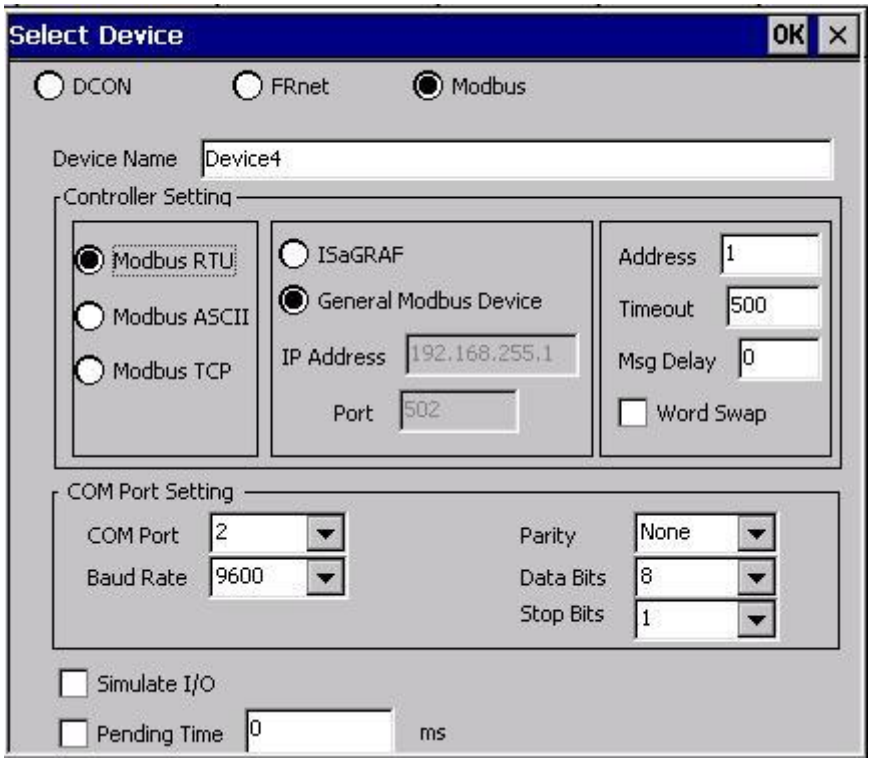


Fig 1.2.3.5-2

Device Name:

Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

ISaGRAF:

Connect ISaGRAF controller

General Modbus Device:

Connect general modbus device

Address:

Specifies a Address for this controller. The default value is 1 and the valid range is between 1 to 255.

Timeout:

Specifies timeout (Response time) value for this controller. A smaller timeout value may cause communication failure and a larger timeout value may reduce the performance of the client program.

Msg Delay:

Specifies message delay value for this controller. The default value is 0 ms. A smaller msg delay value may have a higher system loading, but it will have a faster data exchange speed.

Word Swap:

The "Word Swap" checkbox switches the interpretation of 4 Byte values. Sometimes we need to make the checkbox "TRUE" in order to achieve the purpose of Lo-Hi/Hi-Lo communication.

COM Port:

Specifies the COM port to be used. Please verfiy which COM port number that the RS-485 network is using. Wrong settings will always cause communication failure.

Baud Rate:

Specifies the baud rate to be used. Verify the module's current baud rate. A wrong setting will always cause communication error for this controller.

Parity:

Specifies the parity scheme to be used. It is one of the following values.

Value	Description
None	No parity
Even	Even
Odd	Odd

Data Bits:

Specifies the number of bits in the bytes transmitted and received.

Stop Bits:

Specifies the number of stop bits to be used. It is one of the following values.

Value	Description
1	1 stop bit
2	2 stop bits
1.5	1.5 stop bits

Simulate I/O:

The "Simulate I/O" checkbox switches from reading I/O from the module to running a simulator. Since the simulator does not open the COM port, it is an easy way to work with the server, to configure tags or to connect clients without requiring any hardware.

Pending Time:

Minimum interval time between two access. To activate this function, NAPOPC_CE6 can work under optimized communication performance. If this module only needs to be accessed 1 time per 5 seconds. You can set pending time as 5000 ms. NAPOPC_CE6 will automatically spread time resource to other modules which are connected with each other.

OK:

Click on the "OK" button to add the new controller setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

Step 5: Click on the "OK" button to add this new device.

1.2.3.6 Adding a New Modbus ASCII Controller

Step 1: Click on the "Add/ New Device…" menu item or the icon to add a new module.

Fig 1.2.3.6-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "Modbus" radio button.

Step 4: Click on the "Modbus ASCII" radio button.

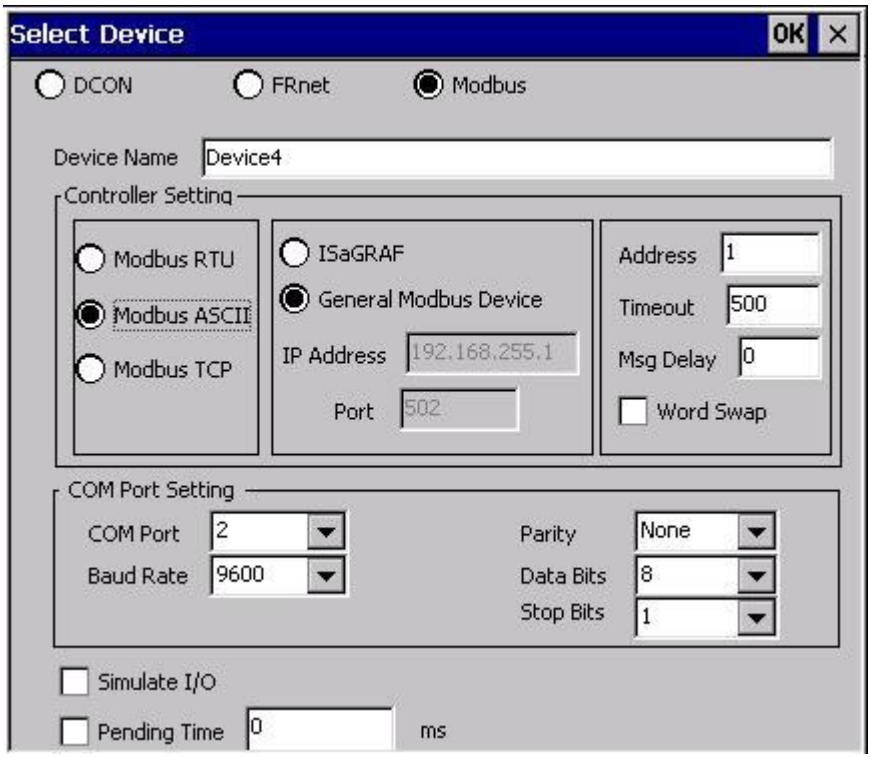


Fig 1.2.3.6-2

Device Name:

Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

ISaGRAF:

Connect ISaGRAF controller

General Modbus Device:

Connect general modbus device

Address:

Specifies a Address for this controller. The default value is 1 and the valid range is between 1 to 255.

Timeout:

Specifies timeout (Response time) value for this controller. A smaller timeout value may cause communication failure and a larger timeout value may reduce the performance of the client program.

Msg Delay:

Specifies message delay value for this controller. The default value is 0 ms. A smaller msg delay value may have a higher system loading, but it will have a faster data exchange speed.

Word Swap:

The "Word Swap" checkbox switches the interpretation of 4 Byte values. Sometimes we need to make the checkbox "TRUE" in order to achieve the purpose of Lo-Hi/Hi-Lo communication.

COM Port:

Specifies the COM port to be used. Please verfiy which COM port number that the RS-485 network is using. Wrong settings will always cause communication failure.

Baud Rate:

Specifies the baud rate to be used. Verify the module's current baud rate. A wrong setting will always cause communication error for this controller.

Parity:

Specifies the parity scheme to be used. It is one of the following values.

Value	Description
None	No parity
Even	Even
Odd	Odd

Data Bits:

Specifies the number of bits in the bytes transmitted and received.

Stop Bits:

Specifies the number of stop bits to be used. It is one of the following values.

Value	Description
1	1 stop bit
2	2 stop bits
1.5	1.5 stop bits

Simulate I/O:

The "Simulate I/O" checkbox switches from reading I/O from the module to running a simulator. Since the simulator does not open the COM port, it is an easy way to work with the server, to configure tags or to connect clients without requiring any hardware.

Pending Time:

Minimum interval time between two access. To activate this function, NAPOPC_CE6 can work under optimized communication performance. If this module only needs to be accessed 1 time per 5 seconds. You can set pending time as 5000 ms. NAPOPC_CE6 will automatically spread time resource to other modules which are connected with each other.

OK:

Click on the "OK" button to add the new controller setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

Step 5: Click on the "OK" button to add this new device.

1.2.3.7 Adding a New Modbus TCP Controller

Step 1: Click on the "Add/ New Device…" menu item or the icon to add a new module.

Fig 1.2.3.7-1

Step 2: The "Select Device" window pops up.

Step 3: Click on the "Modbus" radio button.

Step 4: Click on the "Modbus TCP" radio button.

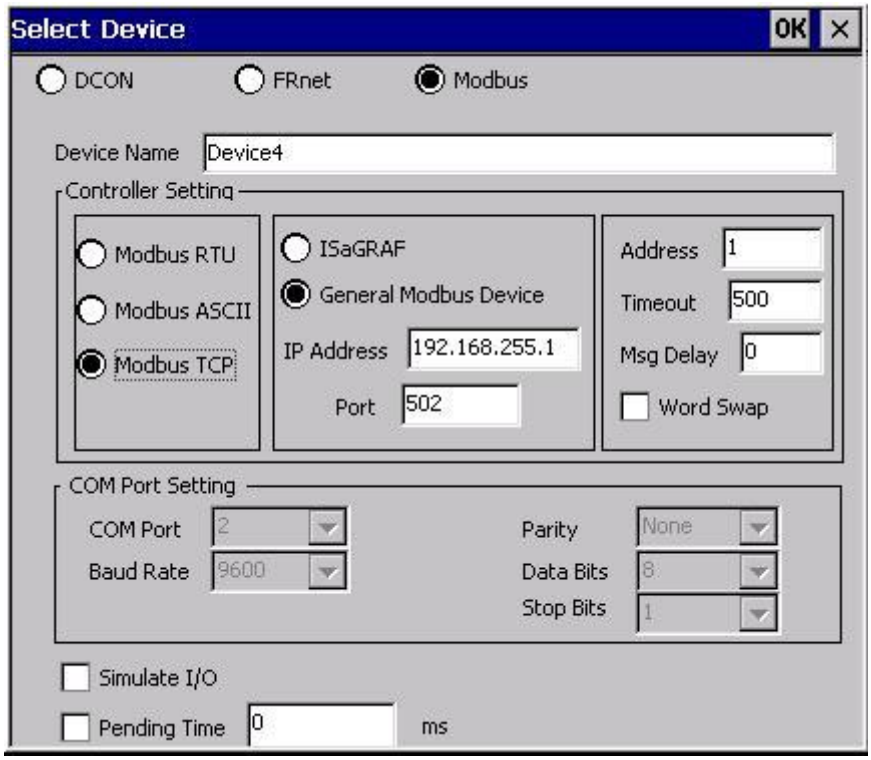


Fig 1.2.3.7-2

Device Name:

Names with spaces or punctuation such as "|!.," cannot be used within a module name. The clients use the "Device Name" and "Tags" to access its value. The "Device Name" can not be the same as any other module.

ISaGRAF:

Connect ISaGRAF controller

General Modbus Device:

Connect general modbus device

IP Address:

The uniqe IP address of your Modbus TCP controller.

Port:

You have to set up the value with "502" for communicating with ICP DAS Modbus TCP controller

Address:

Specifies a Address for this controller. The default value is 1 and the valid range is between 1 to 255.

Timeout:

Specifies timeout (Response time) value for this controller. A smaller timeout value may cause communication failure and a larger timeout value may reduce the performance of the client program.

Msg Delay:

Specifies message delay value for this controller. The default value is 0 ms. A smaller msg delay value may have a higher system loading, but it will have a faster data exchange speed.

Word Swap:

The "Word Swap" checkbox switches the interpretation of 4 Byte values. Sometimes we need to make the checkbox "TRUE" in order to achieve the purpose of Lo-Hi/Hi-Lo communication.

Simulate I/O:

The "Simulate I/O" checkbox switches from reading I/O from the module to running a simulator. Since the simulator does not open the COM port, it is an easy way to work with the server, to configure tags or to connect clients without requiring any hardware.

Pending Time:

Minimum interval time between two access. To activate this function, NAPOPC_CE6 can work under optimized communication performance. If this module only needs to be accessed 1 time per 5 seconds. You can set pending time as 5000 ms. NAPOPC_CE6 will automatically spread time resource to other modules which are connected with each other.

OK:

Click on the "OK" button to add the new controller setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

1.2.4 Adding a New Group

Step 1: Click on the "Add/ New Group" menu item or the  icon to add a new group.

Step 2: The "Group" window pops up.

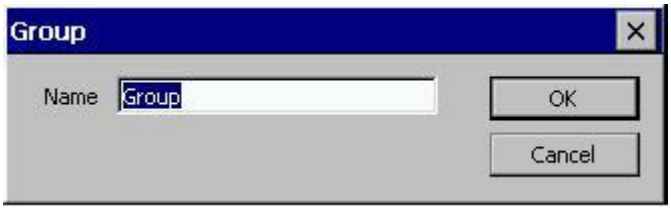


Fig 1.2.4-1

Name:

A "Group Name" may have any name, but avoid names with spaces or punctuation such as "|!.,". The "Group Name" must not be used twice. A group can be defined as a subdirectory containing one or more tags. A device may have many subgroups of tags. All tags belong to their module when they are scanned to perform I/O.

1.2.5 Adding a New Tag

1.2.5.1 Adding New Tags For I-7K/8K/87K/ZigBee/FRnet Module

Step 1: Click on the "Add/Generate Tags" menu item to add new tags.

Fig 1.2.5.1-1

Step 2: "Generate Tags" function will generate tags for the device you choose.

Step 3: Double click the tag to check the property.

Step 4: Choice the "Settings" page. Because the tag belongs to the module-type device, the "I/O Module" radio button is active.

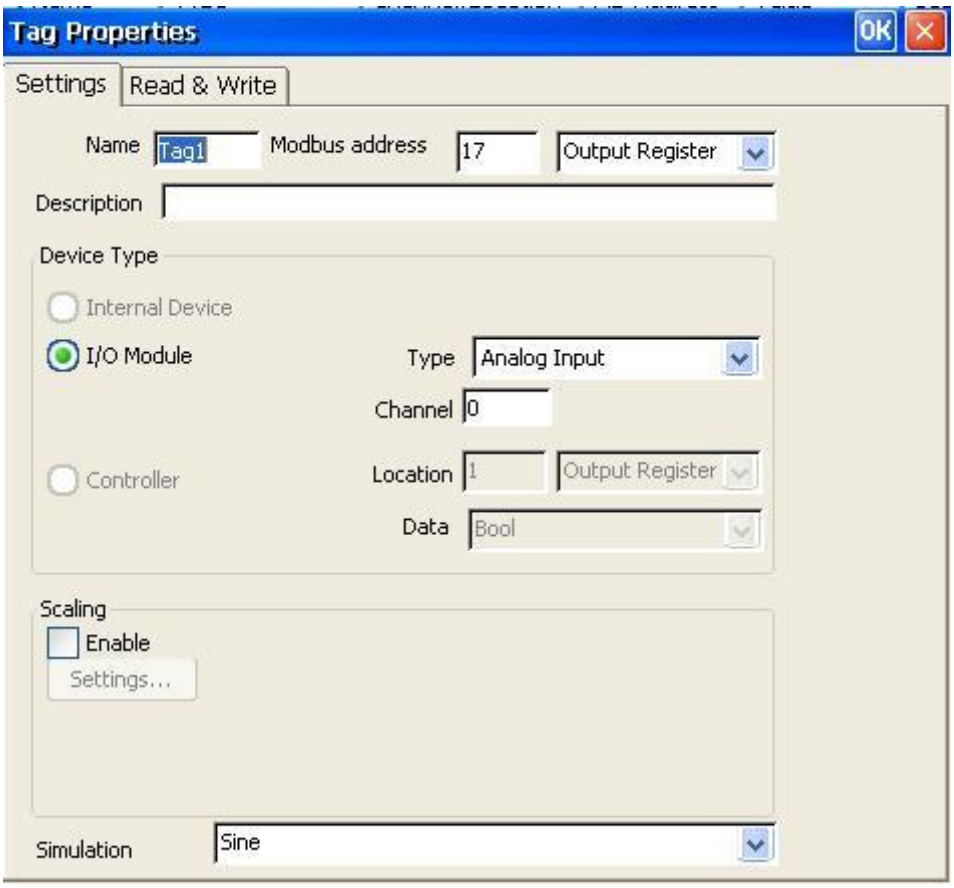


Fig 1.2.5.1-2

Name:

Any "Tag Name" may be used, but avoid names with spaces or punctuation such as "|!.,". The clients will use the "Device Name" and "Tags" to access its value. Hence the "Tag Name" cannot be a duplicate of another tag in the same group.

Modbus address:

Specifies an unique modbus address for this tag in order to communicate with modbus client. The default address is already an unique one.

After that, you also need to choose the address type. There are four address types you can choose.They are "Input Coil", "Output Coil", "Input Register", and "Output Register" which depends on your tag property. It is important to give an appropriate modbus address type and address value.

Address Type	Range
Output Coil	000001 - 001000
Input Coil	100001 - 101000
Input Register	300001 - 301000
Output Register	400001 - 401000

Description:

Specifies the description text for this tag. This can be blank.

Type:

Shows the command to be used for this tag. Different modules support different commands.

Channel:

Specifies the channel number to be used for this tag. The "Digital Input" and "Digital Output" tags do not use this channel setting, because all channels are read with one communication.

Simulation:

The valid signal is SINE, RAMP and RANDOM. This field is validated when the module uses simulation I/O. Please refer to the "Adding A New Device" section.

OK:

Click on the "OK" button to add the new tag setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

Scaling:

Enable:

Check this check-box to enable the "Settings…" button.

Settings:

Click on this button to set the scaling feature.

For more information, please refer to the section "1.2.5.4 Scaling Settings".

1.2.5.2 Adding a New Tag For Internal Device

Step 1: Click on the "Add/ New Tag" menu item or the icon to add a new tag.

Step 2: The "Tag Properties" window pops up.

Step 3: Choice the "Settings" page. Because the tag belongs to the internal-type device, the "Internal Device" radio button is active.

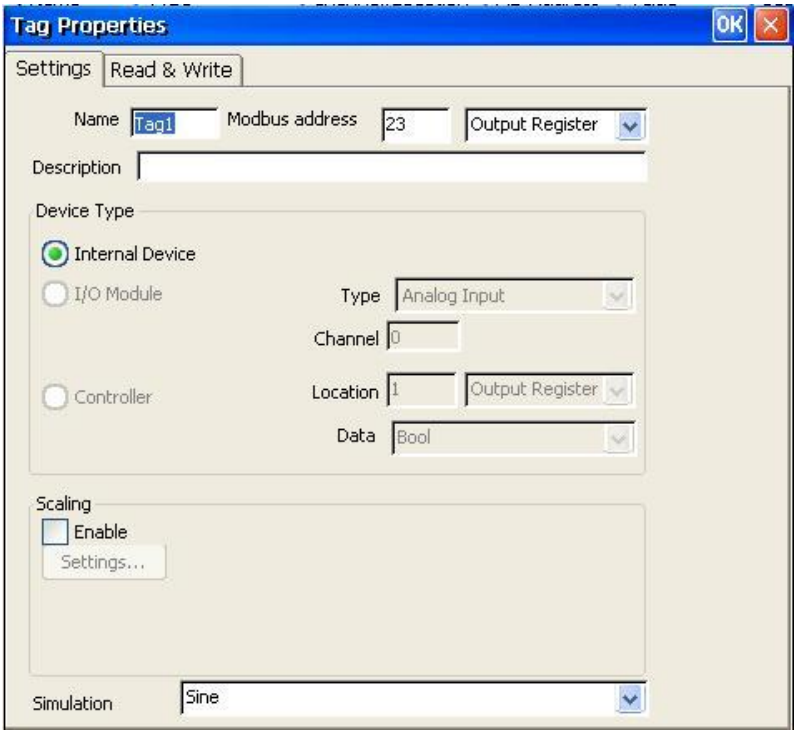


Fig 1.2.5.2-1

Name:

Any "Tag Name" may be used, but avoid names with spaces or punctuation such as "|!.,". The clients will use the "Device Name" and "Tags" to access its value. Hence the "Tag Name" cannot be a duplicate of another tag in the same group.

Modbus address:

Specifies an unique modbus address for this tag in order to communicate with modbus client. The default address is already an unique one.

After that, you also need to choose the address type. There are four address types you can choose.They are "Input Coil", "Output Coil", "Input Register", and "Output Register" which depends on your tag property. It is important to give an appropriate modbus address type and address value.

Address Type	Range
Output Coil	001001 - 020999
Input Coil	101001 - 120999
Input Register	301001 - 320999
Output Register	401001 - 420999

Description:

Specifies the description text for this tag. This can be blank.

1.2.5.3 Adding a New Tag For Modbus Device

Step 1: Click on the "Add/ New Tag" menu item or the icon to add a new tag.

Step 2: The "Tag Properties" window pops up.

Step 3: Choice the "Settings" page. Because the tag belongs to the controller-type device, the "Controller" radio button is active.

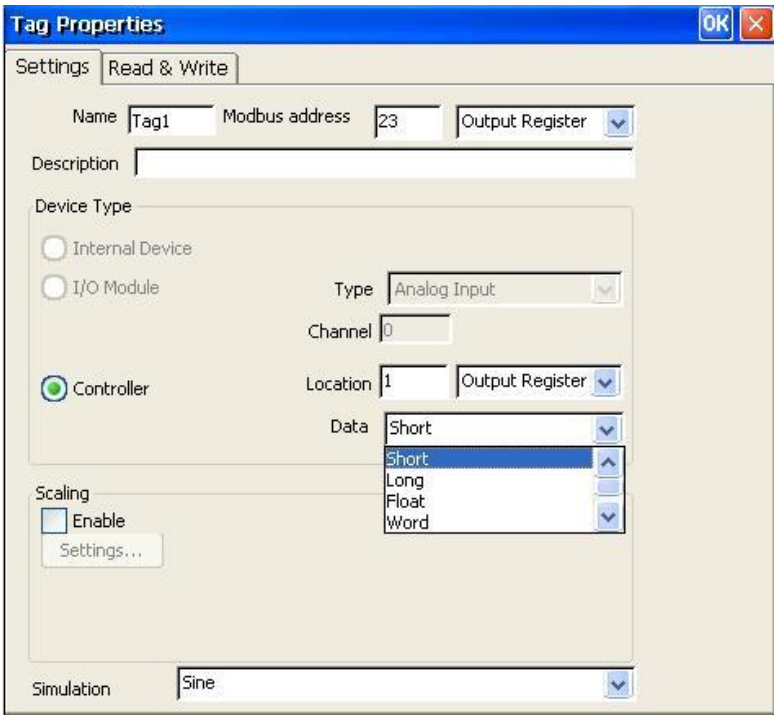


Fig 1.2.5.3-1

Name:

Any "Tag Name" may be used, but avoid names with spaces or punctuation such as "|!.,". The clients will use the "Device Name" and "Tags" to access its value. Hence the "Tag Name" cannot be a duplicate of another tag in the same group.

Modbus address:

Specifies an unique modbus address for this tag in order to communicate with modbus client. The default address is already an unique one.

After that, you also need to choose the address type. There are four address types you can choose.They are "Input Coil", "Output Coil", "Input Register", and "Output Register" which depends on your tag property. It is important to give an appropriate modbus address type and address value.

Address Type	Range
Output Coil	000001 - 001000
Input Coil	100001 - 101000
Input Register	300001 - 301000
Output Register	400001 - 401000

Description:

Specifies the description text for this tag. This can be blank.

Location:

Specifies the tag address. It must be the same with the the variable address in the controller. Besides, you have to choice the location type. After you choice the location number, there are four location types you can choice. They are "Input Coil", "Output Coil", "Input Register", and "Output Register". When you monitor controller device(see 1.2.2 Monitoring Device), the "Channel/Location" field will show a value according to the location and location type as belows.

Location Type	Range
Output Coil	000001 - 065536
Input Coil	100001 - 165536
Input Register	300001 - 365536
Output Register	400001 - 465536

Data:

Specifies the data type of this tag which's location type is "Input Register" or "Output Register". NAPOPC_CE6 supports five kinds of data type which are "Short", "Long", "Float", "Word", and "DWord".

Data Type	Definition	Range
Short	16-bit signed integer	-32768~32767
Long	32-bit signed integer	-2147483648~2147483647
Float	Floating-point variable	-1.7E-308~1.7E+308
Word	16-bit unsigned integer	0~65535
DWord	32-bit unsigned integer	0~4294967295

The data type of "Input Coil" or "Output Coil" is "Bool".

Simulation:

The valid signal is SINE, RAMP and RANDOM. This field is validated when the module uses simulation I/O. Please refer to the "Adding A New Device" section.

OK:

Click on the "OK" button to add the new tag setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

Scaling:

Enable:

Check this check-box to enable the "Settings…" button.

Settings:

Click on this button to set the scaling feature.

For more information, please refer to the section "1.2.5.4 Scaling Settings".

1.2.5.4 Scaling Settings

In general, the "Scaling" feature is only useful for the floating-point data type.

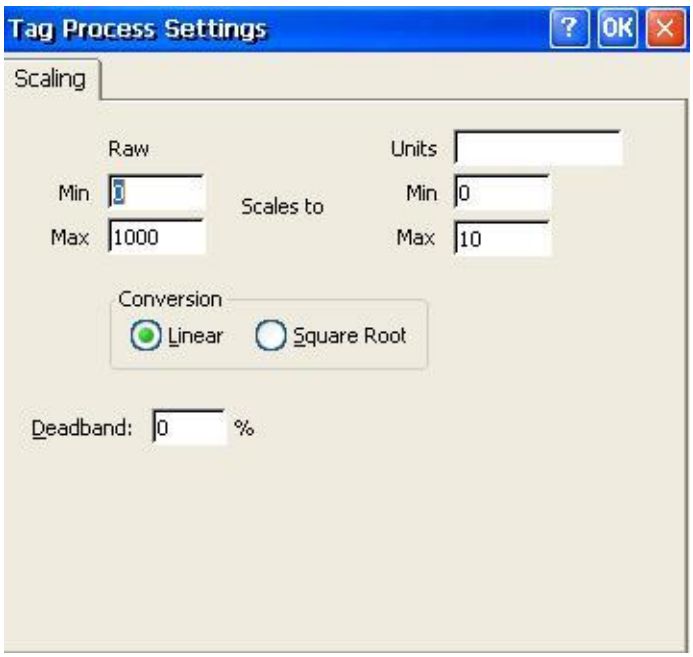


Fig 1.2.5.4-1

Raw Data:

Min: The original Minimum value. ([MinRaw])

Max: The original Maximum value. ([MaxRaw])

Scales to:

Units: The unit of the scaled value. (Just for reference only.)

Min: The scaled Minimum value. ([MinScale])

Max: The scaled Maximum value. ([MaxScale])

Conversion:

Linear:

Scaled Value = ((Original Value – [MinRaw]) / ([MaxRaw] – [MinRaw]))
* ([MaxScale] – [MinScale]) + [MinScale]

Square Root:

Scaled Value = ((sqrt(Original Value) – [MinRaw]) * ([MaxScale] – [MinScale]))
/ sqrt([MaxRaw] – [MinRaw]) + [MinScale]

Deadband(%):

In general, keep "0" in this field. Deadband will only apply to items in the group that have a dwEUType of Analog available. If the dwEUType is Analog, then the EU Low and EU High values for the item can be used to calculate the range for the item. This range will be multiplied with the Deadband to generate an exception limit. An exception is determined as follows:

Exception if (absolute value of (last cached value - current value) > PercentDeadband * (EU High –EU Low))

OK:

Click the "OK" button to save these settings.

Cancel:

Click the "Cancel" button to avoid any changes.

1.2.6 Adding Multi Tags for Modbus Device

This function only work when the device's protocol is Modbus.

Step 1: Click on the "Add/ Multi Tags" menu item

Fig 1.2.6 -1

Step 2: The "Add Multi Tags Dialog" dialog box pops up.

Step 3: Choose correct "Prototype" ,"Data Type" and key in Modbus address.

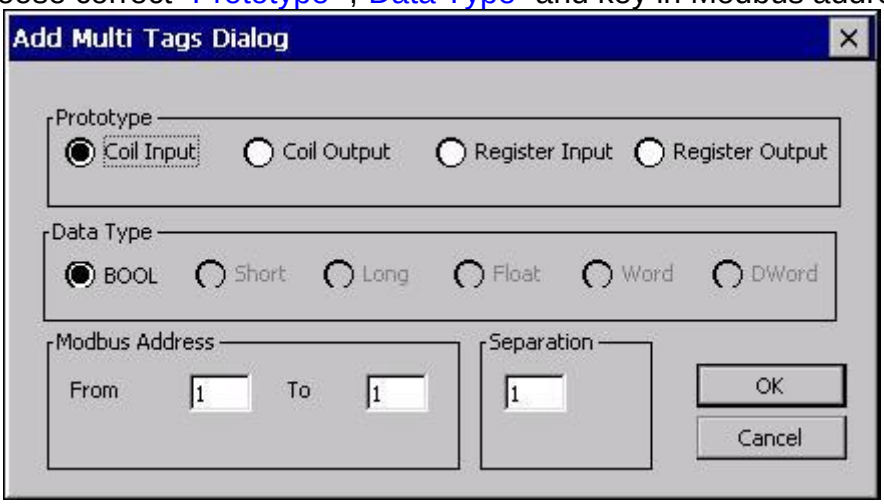


Fig 1.2.6 -2

Prototype:

There are four kinds of prototype for modbus tag. "Coil Input", "Coil Output", "Register Input" and "Register Output".

Data Type:

"Bool" : 8 bits, True or False
"Short" : 16 bits, -32768 ~ 32767
"Long" : 32 bits, -2147483648. ~ 2147483647
"Float" : 32 bits, float numbers
"Word" : 16 bits, 0 ~ 65535
"DWORD" : 32 bits, 0 ~ 4294967295

Modbus Address:

"From" : modbus address number of start tag, 1 ~ 65535
"To" : modbus address number of end tag. 1 ~ 65535

Separation:

Separation numbers between each tag. 1 ~ 100

OK:

Click on the "OK" button to add the new tag setting.

Cancel:

Click on the "Cancel" button to avoid any changes.

1.2.7 Read/Write the Tags

First, you have to use the "Monitor" function to see values of tags by checking the "View/ Monitor" menu item. Select a tag and right click the mouse button. Then select the "Properties.." option. Choose the "Read & Write" page to read/write the tag.

Step 1: Click the "View/ Monitor" menu item to enable monitor.

Step 2: Select a tag and right click the mouse button. Then select the "Properties.." option.

Step 3: Choose the "Read & Write" page. You can see the "Tag name" and "Access right" at the first. If the access right is "Read only!", the write function is disable.

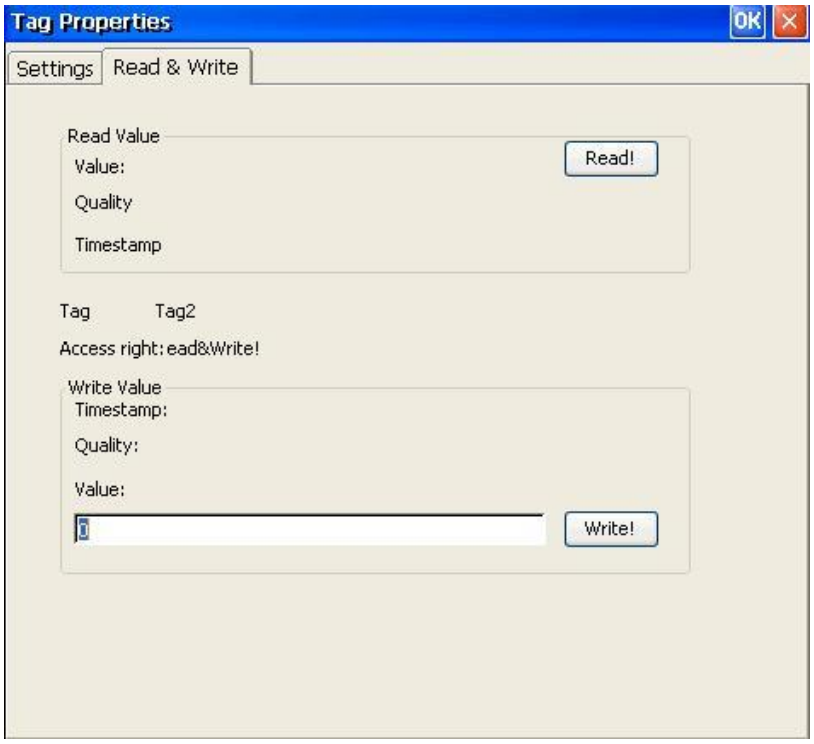


Fig 1.2.7-1

Read Value/Value:

You can press the "Read!" button to read the tag value as you saw on the "Tag-Window".

Read Value/Quality:

Three kinds of qualities, "Good", "Bad", and "Uncertain", would be shown. If the communication status is good, the quality shows "Good". If the communication status has something wrong, the shows "Bad". And the other situation is "Uncertain".

Read Value/Timestamp:

It shows the time, when you read the tag.

Tag name:

It is the same with the "Name" at the "Settings" page. You can modify it at the "Settings" page.

Access right:

There are two kinds of aceess rights, "Read Only!" and "Read&Write!". The access right depends on what kind of tag property it is. Please refer to the "1.6 Adding A New Tag"

Write Value/Timestamp:

It shows the time that the tag is written.

Write Value/Quality:

Three kinds of qualities, "Good", "Bad", and "Uncertain", would be shown. If the communication status is good, the quality shows "Good". If the communication status has something wrong, the shows "Bad". And the other situation is "Uncertain".

Write Value/Value:

You can press the "Write!" button to write the value you key-in to the tag. If the tag data type is "Boolean" the write value "0" means "OFF" and the write value "not 0" means "ON".

1.2.8 Editing A Device/Group/Tag properties

To edit an existing Device or Group, just select the Device or Group and then select the "Properties…" option.

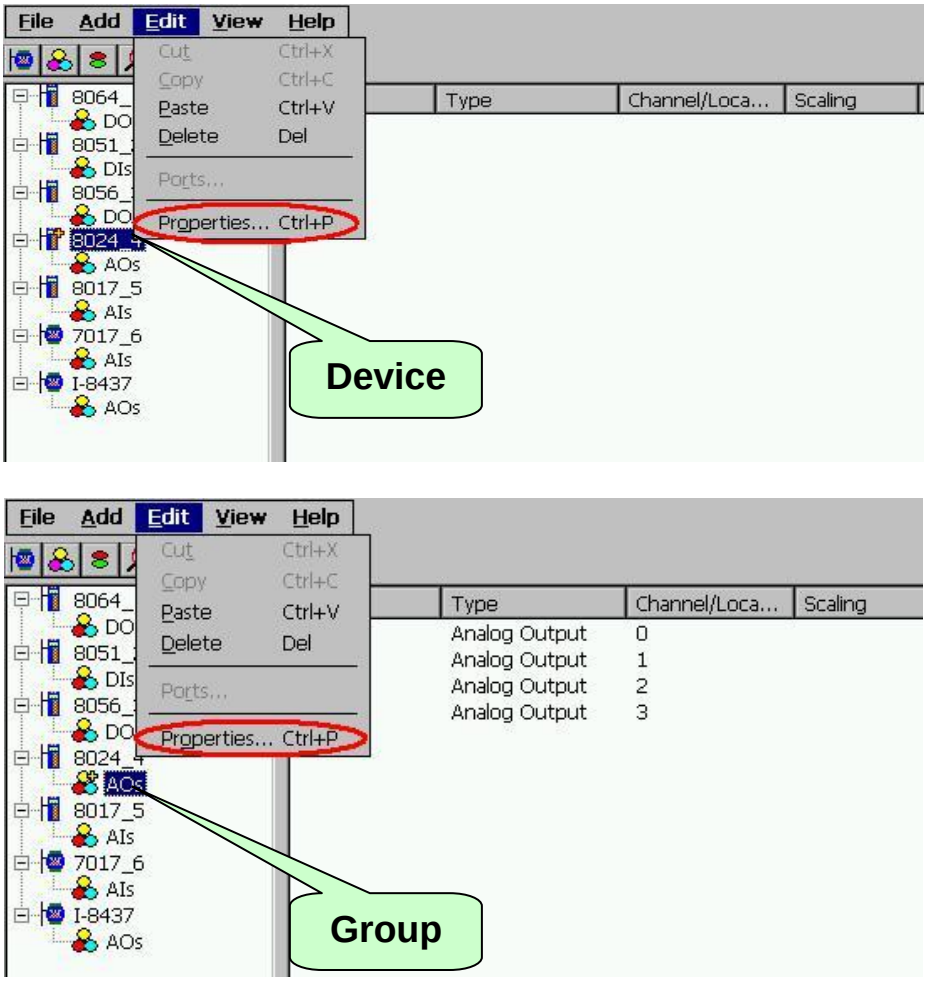


Fig 1.2.8-1

To edit an existing Tag, just select the Tag and right click mouse button to select "Properties…" option.

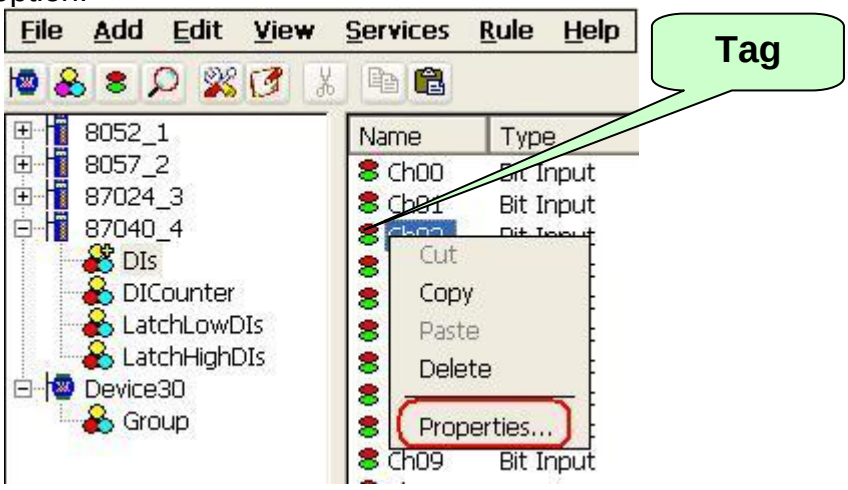


Fig 1.2.8-2

1.2.9 Deleting A Device/Group/Tag

To delete an existing Device or Group, just select the Device or Group and then select the "Delete…" option.

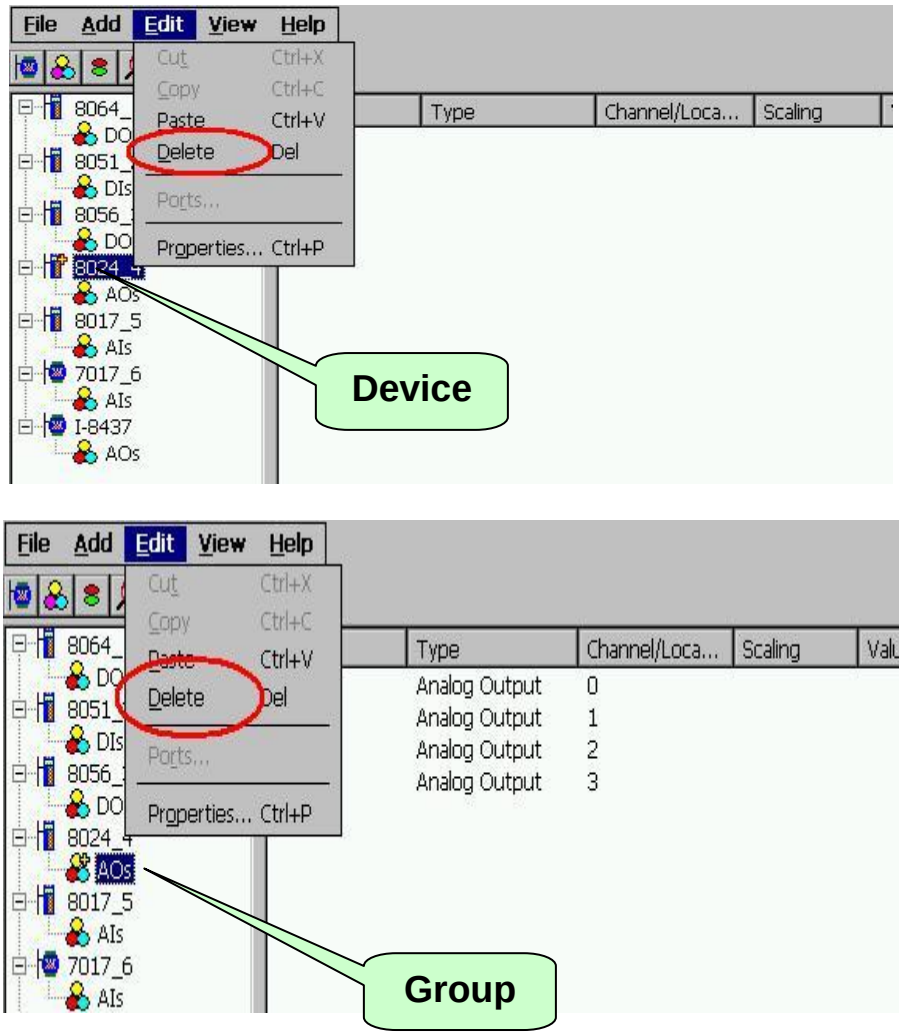


Fig 1.2.9-1

To delete an existing Tag, just select the Tag and right click mouse button to select "Delete…" option.

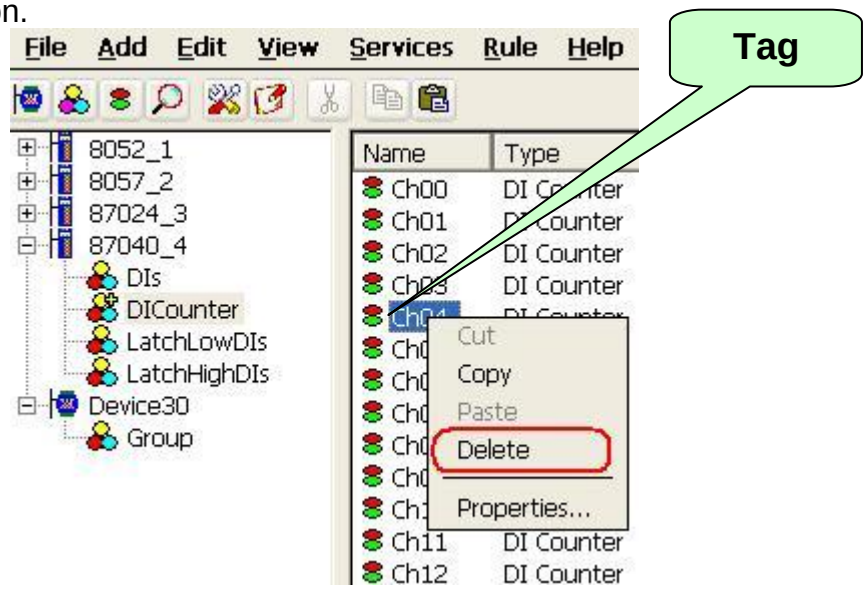


Fig 1.2.9-2

1.2.10 Generating Tags

This function lets you easily test the NAPOPC_CE6 in the simulation mode. It is only valid if the selected device of module type has no sub "Module", "Group" and "Tag".

Step 1: Select a device of module type you want to generate tags.
Step 2: Click on the "Add/ Generate Tags" menu item.

Fig 1.2.10-1

Tags are generated depending on the Module-ID. Possible tags are "Analog Input", "Analog Output", "Digital Input", "Digital Output", "Latched DI" and "Counter".

1.2.11 Services Setup

This function lets you define which services you want to active for exchanging data with the other programs. NAPOPC_CE6 provides "RPC Server", "Modbus RTU", "Modbus ASCII", "Modbus TCP", and "Active ScanKernel" four services to be choosed. In them, the "RPC Server" is a mechanism which allows NAPOPC_ST/NAPOPC_XPE DA Server use NAPOPC_CE6 via "Remote Procedure Call". If you wanna create a "RPC" device at NAPOPC_ST/NAPOPC_XPE site, please check this at NAPOPC_CE6 site. "Modbus RTU", "Modbus ASCII", and "Modbus TCP" services would active immediately by checking. The "Active ScanKernel" service should check at all situation except to be the intermediary program between user application programs.

Step 1: Click on the "Services/Setup" menu item.

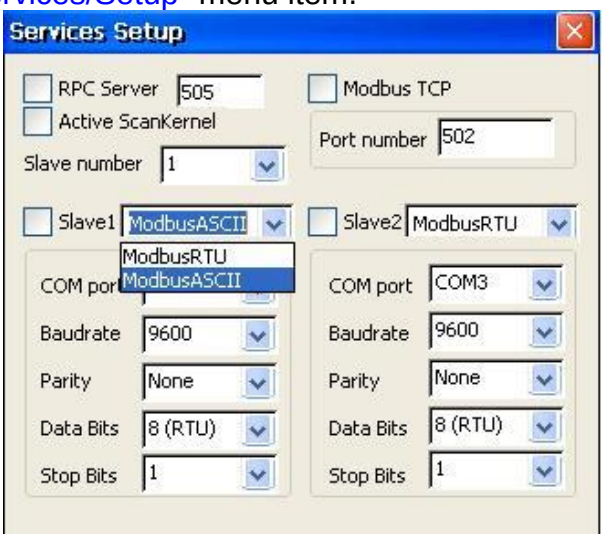


Fig 1.2.11-1

Step 2: Choose the services you want.

RPC Server Port:

You have to set up the value with "505" for communicating with NAPOPC_ST or NAPOPC_XPE.

COM Port:

Specifies the COM port to be used. Please verfiy which COM port number that the RS-485 network is using. Wrong settings will always cause communication failure.

Baudrate:

Specifies the baud rate to be used. Verify the module's current baud rate. A wrong setting will always cause communication error for this controller.

Parity:

Specifies the parity scheme to be used. It is one of the following values.

Value	Description
None	No parity
Even	Even
Odd	Odd

Data Bits:

Specifies the number of bits in the bytes transmitted and received.

Stop Bits:

Specifies the number of stop bits to be used. It is one of the following values.

Value	Description
1	1 stop bit
2	2 stop bits
1.5	1.5 stop bits

1.2.12 Rule Script Editor

This function lets you design your rule base for making your XP-8000-CE6 to be a DCS via NAPOPC_CE6. The description of rule base of NAPOPC_CE6 is like "IF…THEN…". The left upper corner in the "Rule Script Editor" has four conditions behind "IF" in which the variables are showed as modbus address and combined with "AND/OR" each other. The right upper corner in the "Rule Script Editor" has four outputs behind "THEN" in which the variables are showed as modbus address and combined with "AND" each other. The relation between timer value and other variables is "AND".

If the variable behind "IF" is "0xxxxx" or "1xxxxx", the "Status" would be "0" or "1". The value "0" means "OFF" and the value "1" means "ON". If the variable is "3xxxxx" or "4xxxxx", the "Status" would depend on the data type of variable.

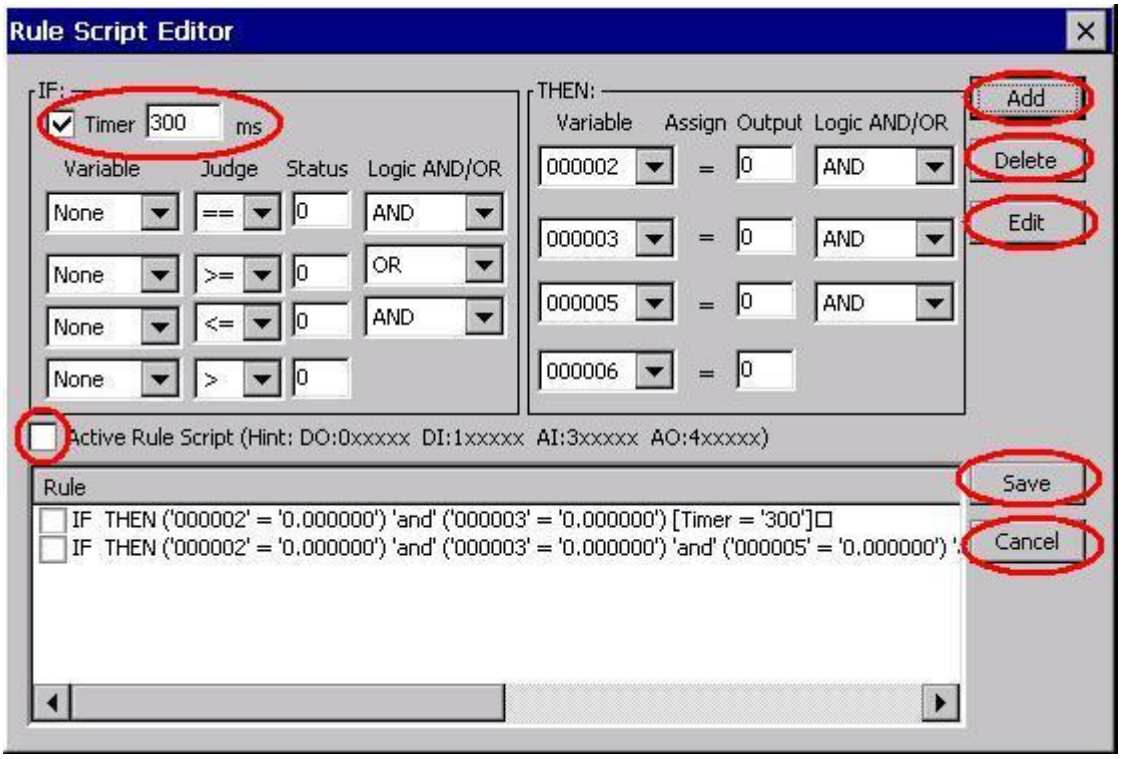


Fig 1.2.12-1

Add:

Press this button to the "Rule list" after editing each rule.

Delete:

Check the rules in the "Rule list", and then press this button to delete.

Edit:

Click the rule in the "Rule list" to edit, and after that press this button to update.

Save:

Save the "Rule list" to be "Rule.txt" after finishing editing.

Cancel:

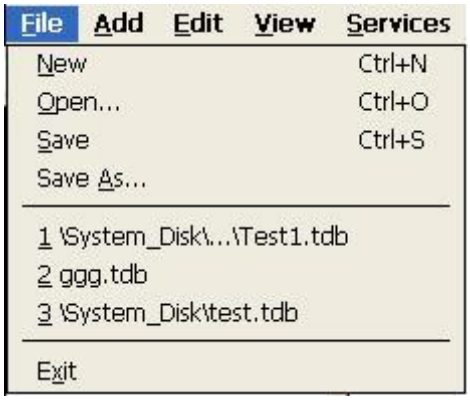
Leave this editor.

Active Rule Script:

It would be active immediately after checking this option. If you wish to act the "Rule script" after rebooting NAPOPC_CE6, you should save file with "File/Save".

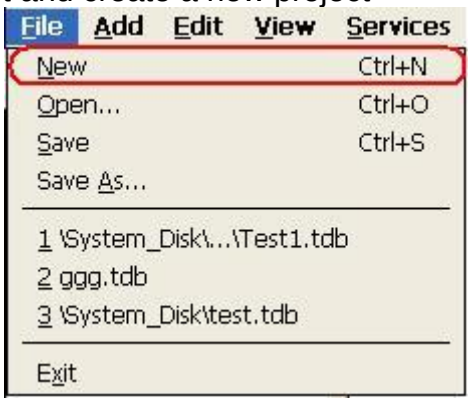
1.2.13 File

This function lets you save and load the configurations of NAPOPC_CE6. For taking the correct configuration file of NAPOPC_CE6 "*.tdb" after rebooting the XP-8000-CE6, you not only use "File/Save" to save in the NAPOPC_CE6 but also need to "Reboot" in the "XPAC Utility". Moreover, NAPOPC_CE6 will automatically load the last configuration file with every launch.



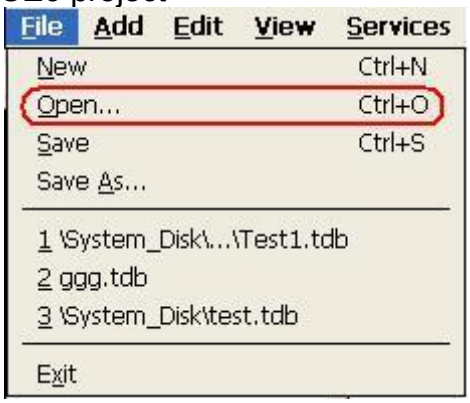
New:

Clean current project and create a new project



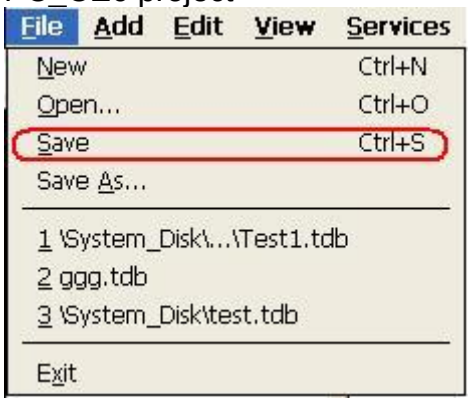
Open:

Load old NAPOPC_CE6 project



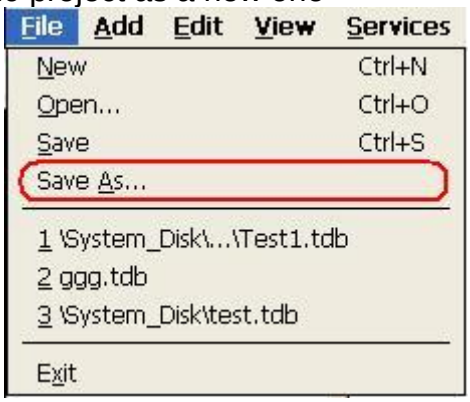
Save:

Save current NAPOPC_CE6 project



Save as…:

Save NAPOPC_CE6 project as a new one



1.2.14 About

Click on the "Help/ About NAPOPC_CE6" menu item to see the "About NAPOPC_CE6" window. It shows the version number.

Step 1: Click on the "Help/ About NAPOPC_CE6" menu item.

Step 2: The "About NAPOPC_CE6" window pops up.

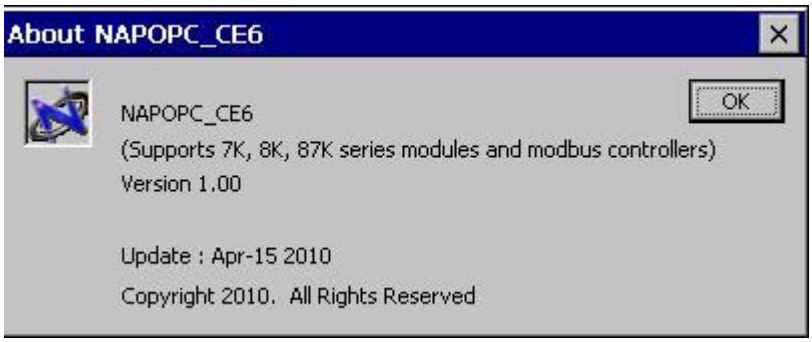


Fig 1.2.14-1

1.2.15 Minimize NAPOPC_CE6

If you want to minimize NAPOPC_CE6, please click on the top-right corner.

Fig 1.2.15-1

After clicking the question mark, NAPOPC_CE6 will minimize itself at the status bar. It will be restored by double clicking it.

Fig 1.2.15-2

2 Quick Start

Please follow these steps:

1. Wiring Modules or Controllers.
   Wiring modules in the RS-485 network.
   Wiring controllers to XP-8000-CE6
   (Refer to xp-8000-ce6_user_manual-verx.x.x.pdf)

2. Configuring Modules or Controllers.
   Using the DCON Utility to set modules.
   (Refer to xp-8000-ce6_user_manual-verx.x.x.pdf)

3. Running NAPOPC_CE6
   Launch NAPOPC_CE6 by means of executing the " NAPOPCSvr_CE6.exe" or "NAPCOP_CE6Boot.exe"

4. Searching Modules.
   Refer to the "1.2.1 Searching Modules.." section to search modules.

5. Adding a new controller
   Refer to the "1.2.3 Adding A New Device" section to add a new modbus RTU or modbus TCP controller.

6. Saving Configuration.
   Refer to the "1.2.13 File Save" section to save the configuration.

7. Closing NAPOPC_CE6.
   Close NAPOPC_CE6 by clicking the "File/Exit" menu item.

3 Remote Accessing

OPC Client has two ways to access the OPC Server. One is called "Local Accessing", and the other is called "Remote Accessing". If the OPC Client and the OPC Server are at the same computer, we said this kind of architecture is "Local Accessing". In other words, if the OPC Client should access OPC Server through a network, we said this kind of architecture is "Remote Accessing".

The following figure shows the integrated architecture including "Local Accessing" and "Remote Accessing". At the real Process Industry, the two ways are often used at the same time. At the Process Management Layer, we often use "Local Accessing" architecture to monitor and control manufacturing processes. At the Business Management Layer, we just set up the OPC Client to collect the process information from the Process Management Layer. If you just want to construct the "Local Accessing" architecture, you do not need to read this chapter. If you want to construct the "Remote Accessing" architecture, you have to know how to set up the DCOM between OPC Client and OPC Server.

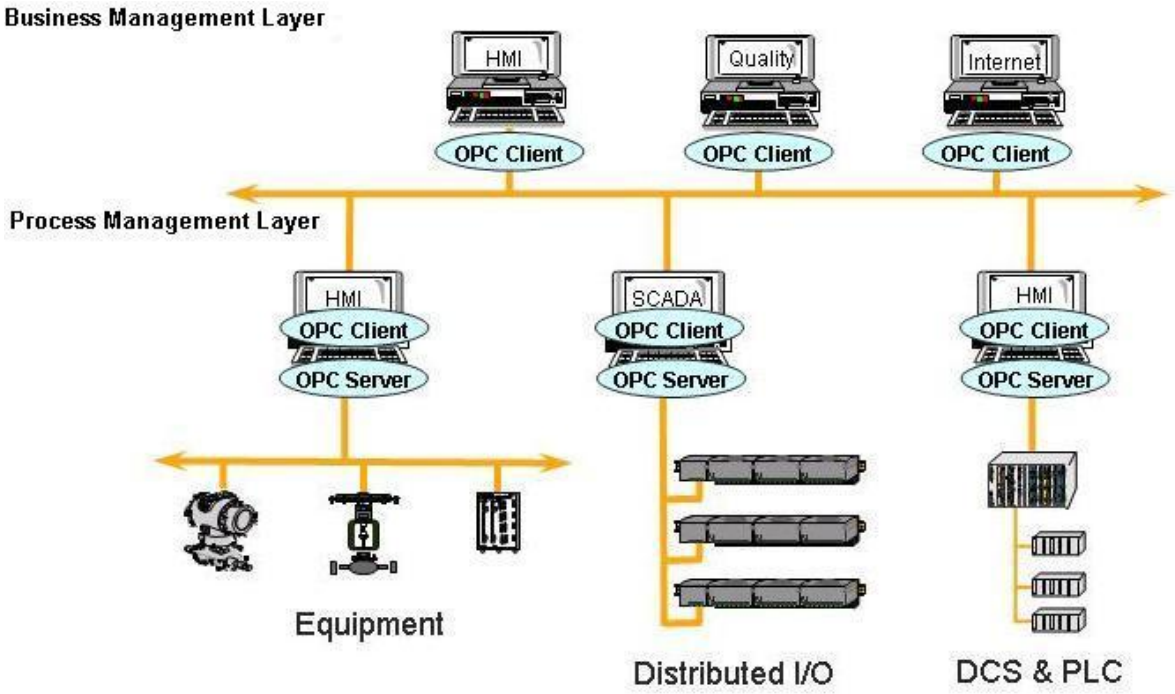


Figure 3-0-1 Local access and Remote access architecture.

3.1 System Requirement

To access a remote OPC server over a network, it is required to enable the DCOM mechanism on both stations, where the client and server are resided.

It is not possible to launch a secure process on a Windows 95 computer from a client computer. All processes in Windows 95 run in the security context of the currently logged-on user; therefore, DCOM on Windows 95 does not support remote activation. A server application on a Windows 95 computer will have to be launched manually or by some other mechanism to be accessed by a client application on another computer. Consequently, the "DefaultLaunchPermissions" and "LaunchPermissions" registry values have no affect on Windows 95.

Platform	Does the platform support the DCOM?
Windows 95	No. Users need to download and install the DCOM95.EXE and DCM95CFG.EXE from Microsoft's web site to enable the remote access.
Windows 98	Yes. Windows 98 supports the DCOM mechanism. It is recommended to upgrade to the newest version of DCOM98. The newest DCOM98 is also available at Microsoft's web site.
Windows NT 4.0	Yes. Windows NT 4.0 supports the DCOM mechanism. It is recommended to upgrade to the newest Service Pack for Windows NT 4.0 (Service Pack 3 or newer one).
Windows 2000	Yes. Windows 2000 supports the DCOM mechanism.
Windows XP	Yes. Windows XP supports the DCOM mechanism.

3.2 Configuring DCOM

Before making changes, register the server application in the registry of both the client and server computers. This may involve either running the server applications setup program or running the server application, then shutting it down on both computers. The server application does not need to reside on the client computer.

If the server uses custom interfaces, the marshaling code must be installed on the client and server computers. Automation servers that support "vtbl-binding" must install their type libraries on the client and server computers. Automation servers that do not support "vtbl-binding" do not need to install their type libraries on the client computer.

After changing the registry, run the client application on the client computer. The DCOM looks at the server application registry entries on the client computer and determines the name of the server computer. It will then connects to the server computer, use the server computer registry to determine the location of the server application, and start the server application on that computer.

You can change the registry with the DCOMCnfg.exe tool, the OLE Viewer tool, or manually. For more information on using OLE Viewer or manual changes, please refer to the "Q158582, HOWTO: Configure a Non-DCOM Server and Client to Use DCOM" article on Microsoft's web site. For more information on using DCOMCnfg.exe to configure the DCOM, please refer to "Inside Distributed COM", written by Guy Eddon and Henry Eddon in 1998 for Microsoft Press.

This section shows you how to configure the DCOM status with DCOMCnfg.exe graphic-driven utility (can be found in the Windows NT system32 folder or in the Windows95/98 system folder) on the client and server computer.

The following table shows three combinations of DCOM settings related to XP-8000-CE6. You can see XP-8000-CE6 can be client site and server site with itself, but XP-8000-CE6 only can be server site against XPAC and PC. The limitation is due to DCOM security. We only choose Windows XP for example to set up DCOM because there are too many kinds of OS on PC. You can use other Microsoft desktop operation system on our PC.

Client Site	Server Site
PC(NAPOPC_ST Server)	XP-8000-CE6 (NAPOPC_CE6 Server)
XPAC(NAPOPC_XPE Server)	XP-8000-CE6 (NAPOPC_CE6 Server)
XP-8000-CE6(NAPOPC_CE6 Server)	XP-8000-CE6 (NAPOPC_CE6 Server)

3.2.1 Configuring On the Server Site (XP-8000-CE6)

System Requirement

OS version:

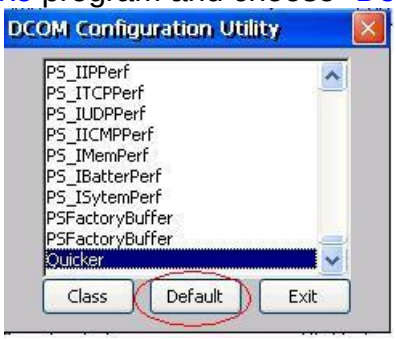
XP-8000-CE6 OS 1.3.1.0 or later

Program:

NAPOPC_CE6
DCOMCnfg.exe
NTLMUser.exe

Configuring DCOM

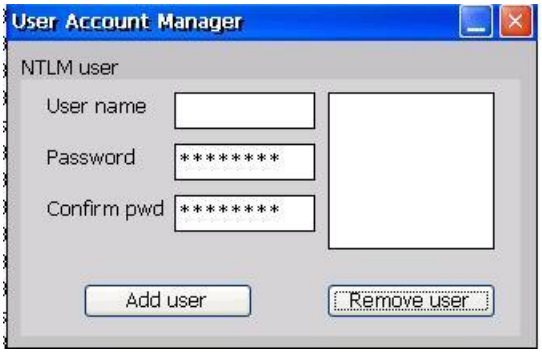
Step 1: Run the dcomcnfg.exe program and choose "Default".



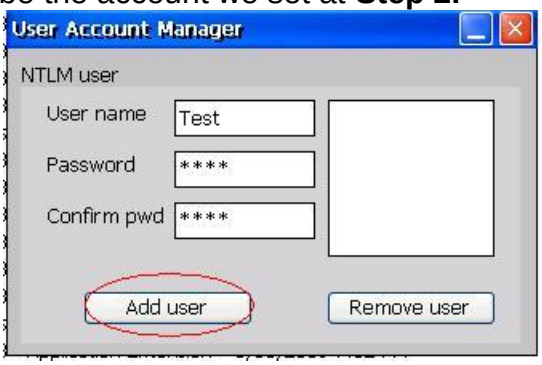
Step 2: Select the "Access" button to add an account.

Step 3: Select the "Launch" button to add an account as above.

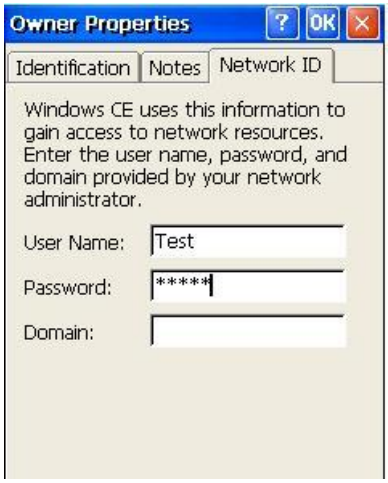
Step 4: Execute NTLMUser.exe



Step 5: Fill out "User name", "Password", and "Confirm pwd" and press "Add user". The "User name" must be the account we set at **Step 2.**



Step 6: Select "Control Panel" → "Owner Properties" → "Network ID" and fill out the User name/Password we set up before.



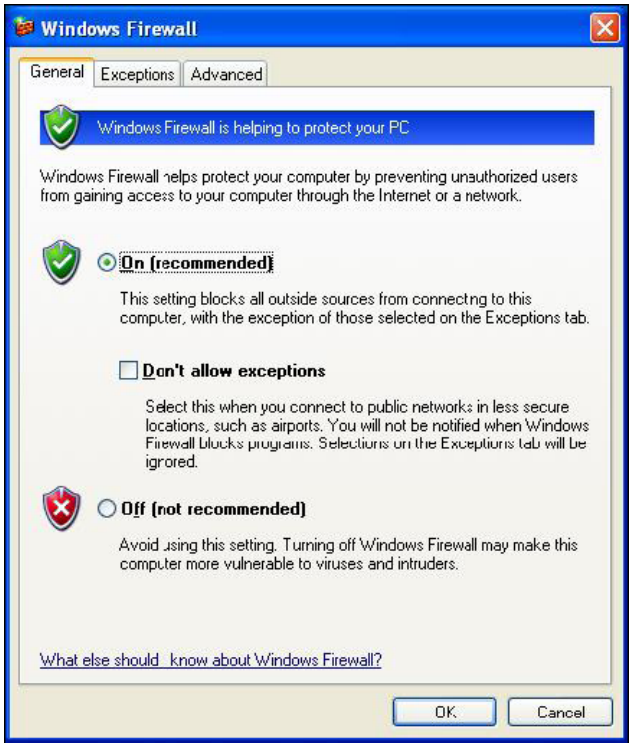
Step 7: Run XPAC Utility to save and reboot.

3.2.2 Configuring On the Client Site (PC)

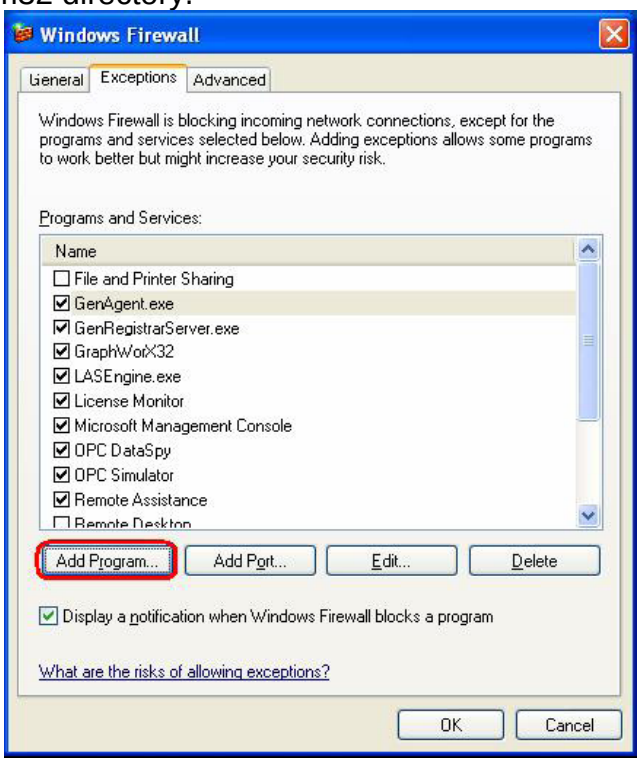
Configuring the Firewall

Step1: By default the windows firewall is set to "On". This setting is recommended by Microsoft and by OPC to give your machine the highest possible protection. For trouble shooting, you may wish to temporarily turn off the firewall to prove or disprove that the firewall configuration is the source of any communication failure.

Note: It may be appropriate to permanently turn off the firewall if the machine is sufficiently protected behind a corporate firewall. When turned off, the individual firewall settings outlined here need not be performed to allow OPC communication.



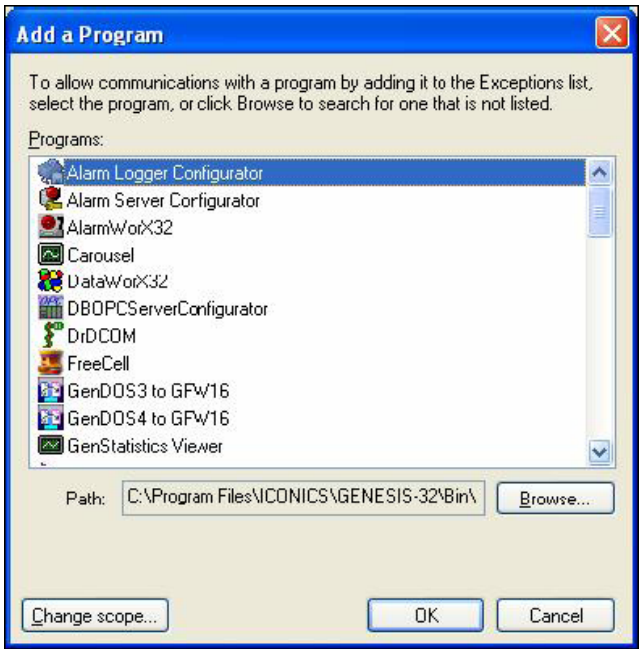
Step 2: Select the .Exceptions tab and add all OPC Clients and Servers to the exception list. Also add Microsoft Management Console (used by the DCOM configuration utility in the next section) and the OPC utility OPCEnum.exe found in the Windows\System32 directory.



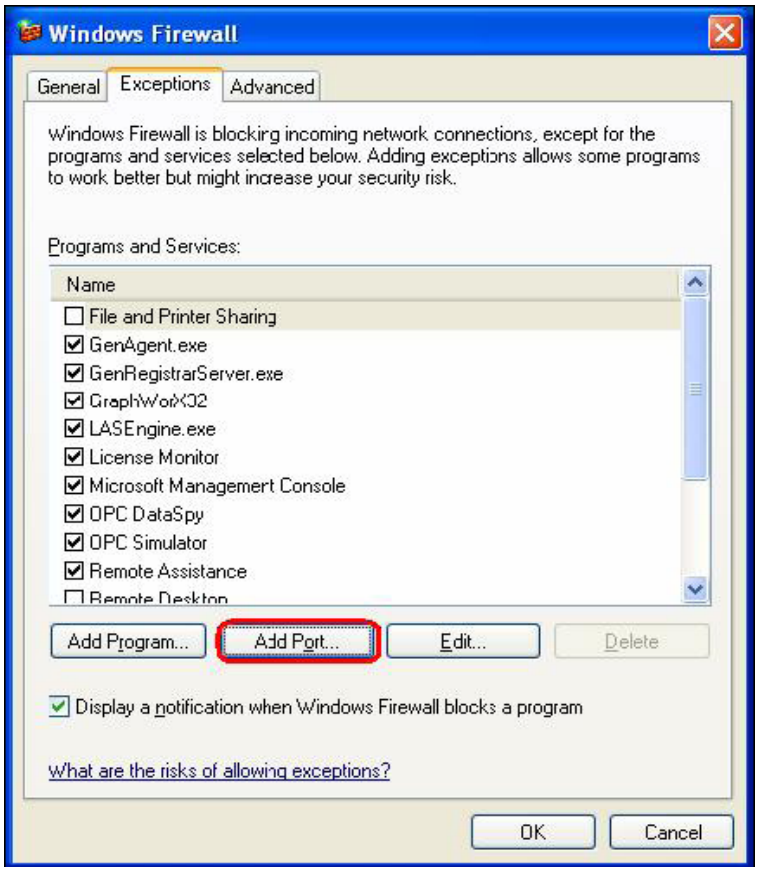
In the Add a Program dialog, there is a listing of most applications on the machine,

but note that not all of them show up on this list. Use the "Browse" button to find other executables installed on the computer.

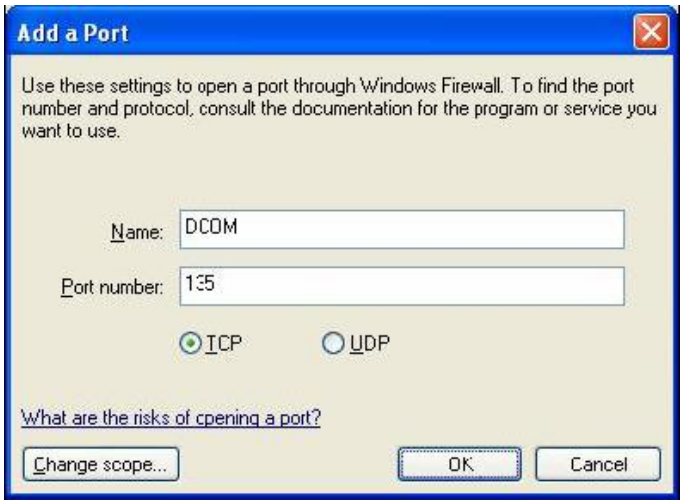
Note: Only EXE files are added to the exceptions list. For in-process OPC Servers and Clients (DLLs and OCXs) you will need to add the EXE applications that call them to the list instead.



Step 3: Add TCP port 135 as it is needed to initiate DCOM communications, and allow for incoming echo requests. In the Exceptions tab of the Windows Firewall, click on Add Port.

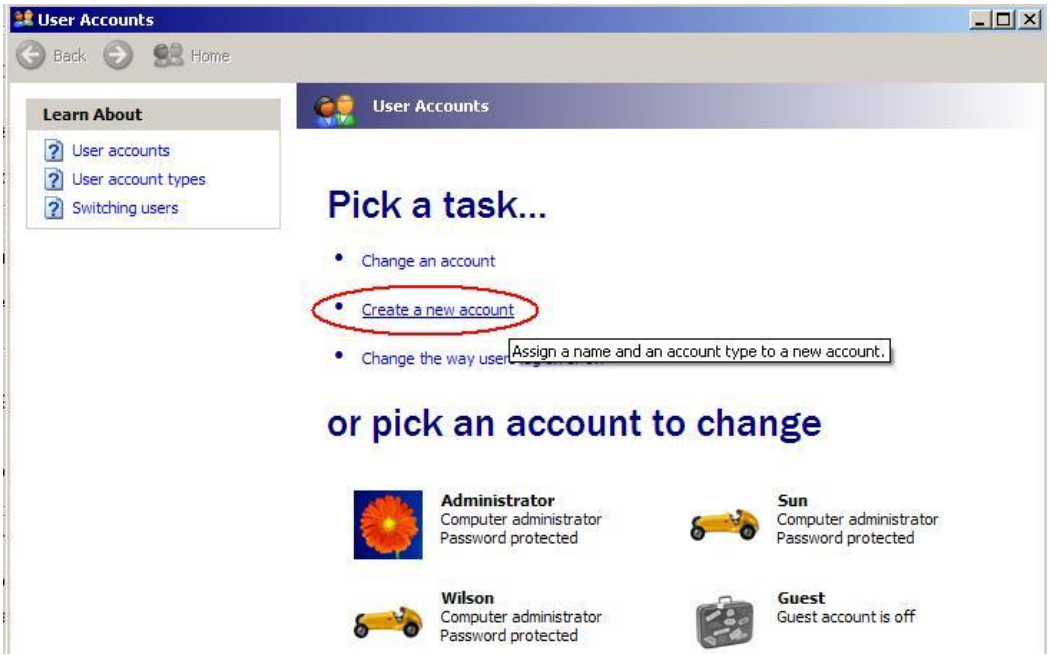


In the Add a Port dialog, fill out the fields as follows:
Name: DCOM
Port number: 135
Choose the TCP radio button



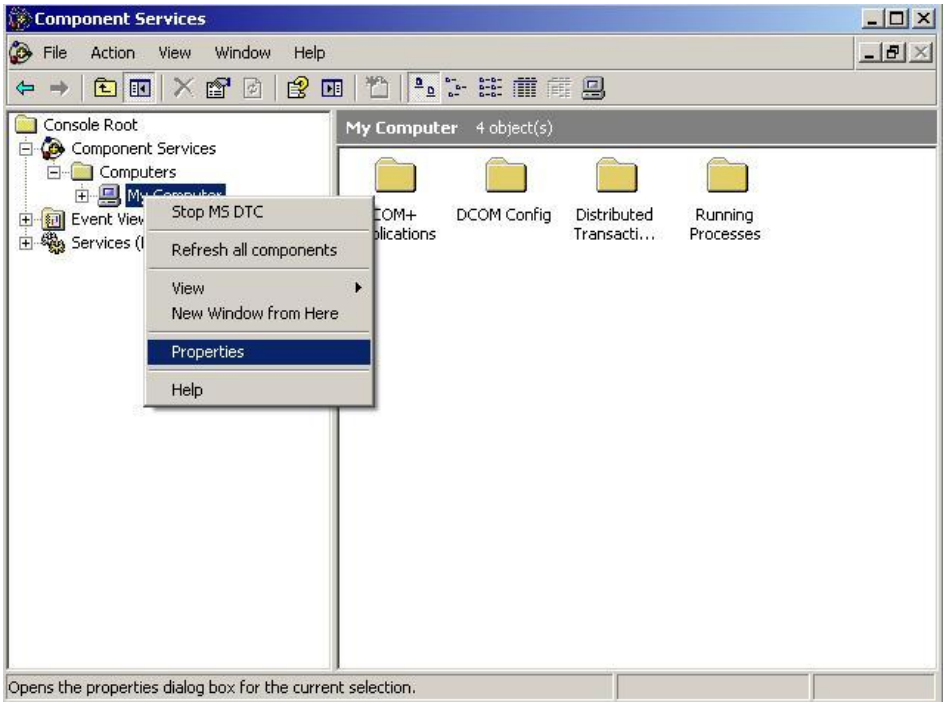
Creating the Account

Step 1: Create a account which must be the same with the account of server site.



Configuring DCOM

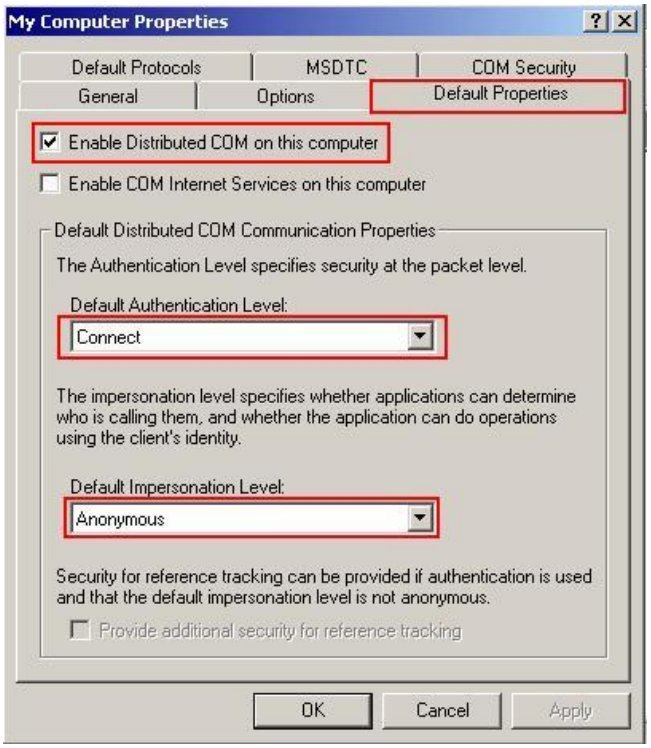
Step 1: Run the dcomcnfg.exe program to launch component services. Right clieck "My Computer" and choose "Properties".



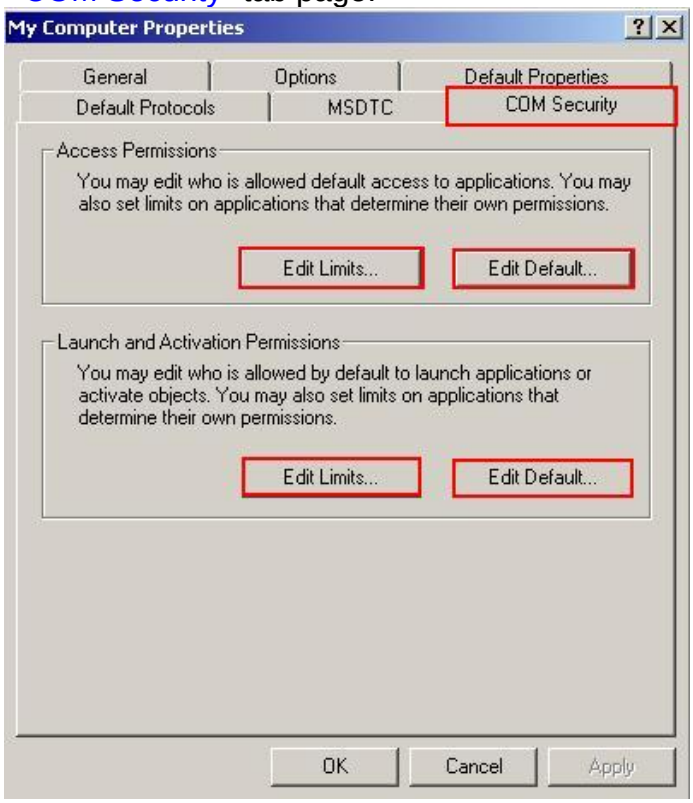
Step 2: Select the "Default Properties" tab page.

Step 3: Use the following settings:

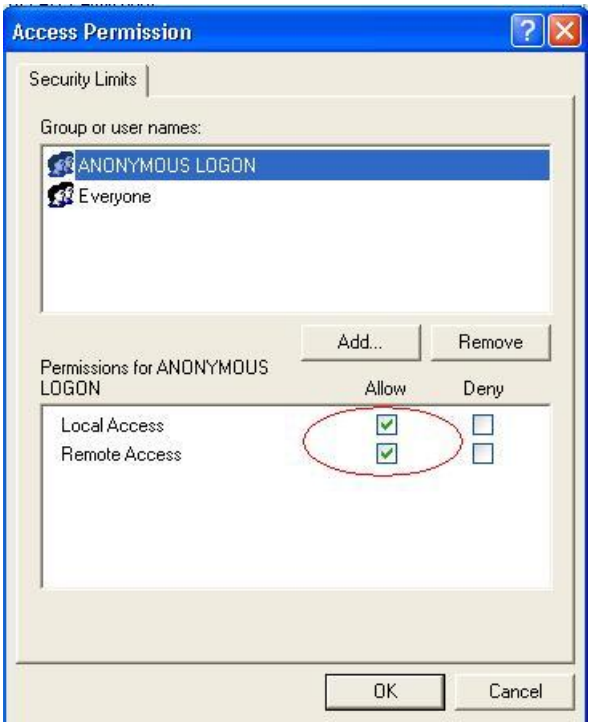
Field Name	Set to
Enable Distributed COM on this computer	Checked
Default Authentication Level:	Connect
Default Impersonation Level:	Anonymous

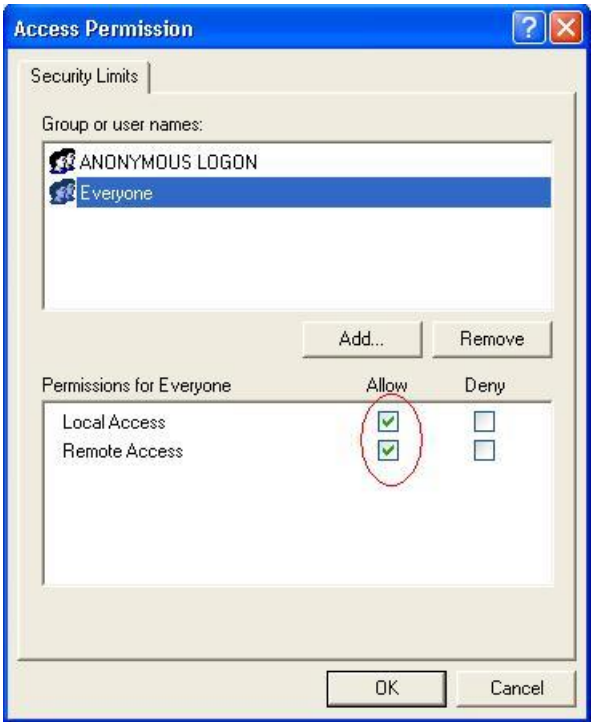


Step 4: Select the "COM Security" tab page.

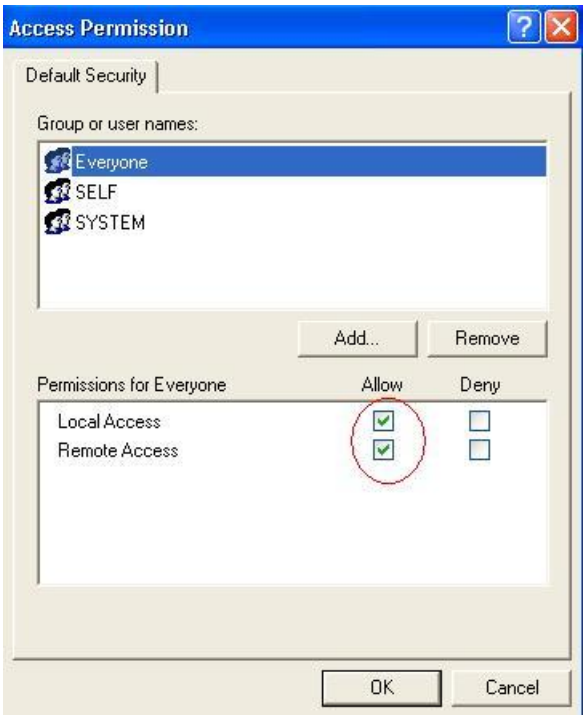


Step 5: Click on the "Edit Limits…" of "Access Permissions" button to set.

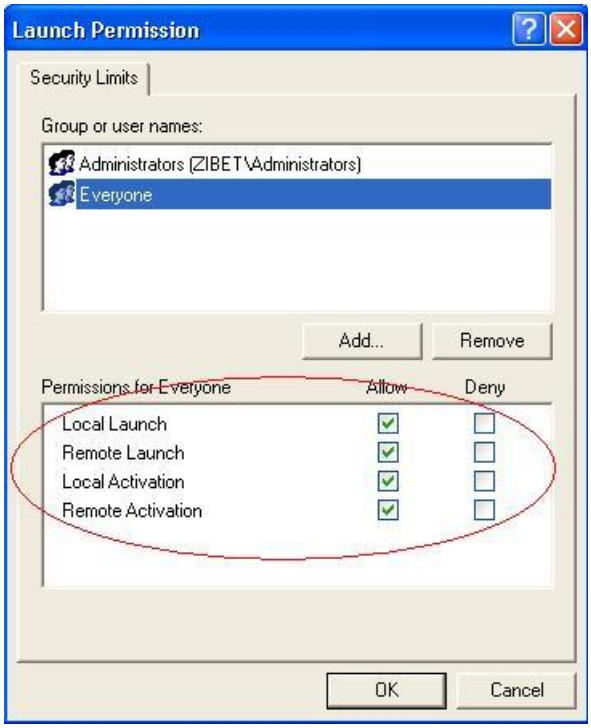




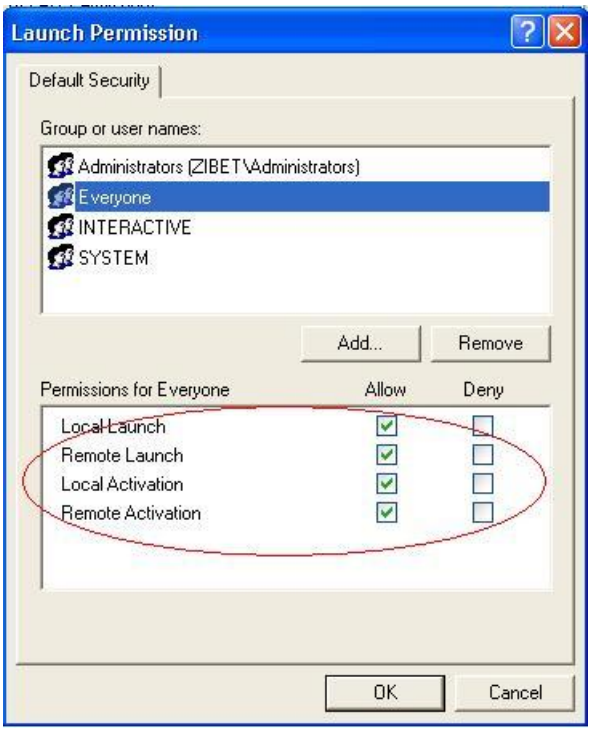
Step 6: Click on the "Edit Default…" of "Access Permissions" button to set.



Step 7: Click on the "Edit Limits…" of "Launch and Activation Permissions" button to set.

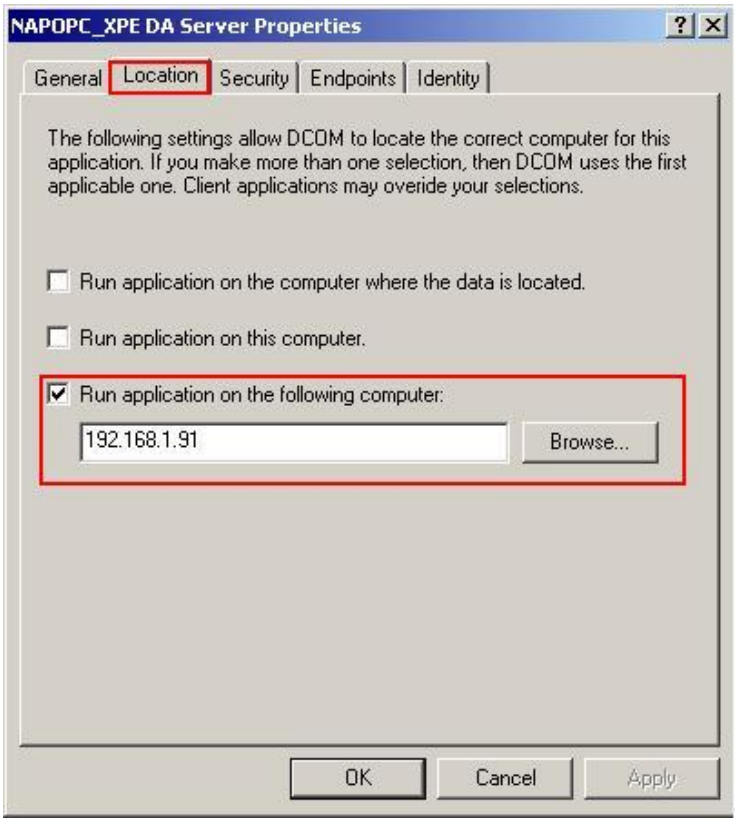


Step 8: Click on the "Edit Default…" of "Launch and Activation Permissions" button to set.



Step 9: Right click on the "NAPOPC DA Server" of "DCOM Config" button and select "Properties".

Step 10: Select the "Location" tab page and check "Run application on the following computer". And enter the Server IP here.



Step 11: Select the "Identity" tab page and check "The interactive user"

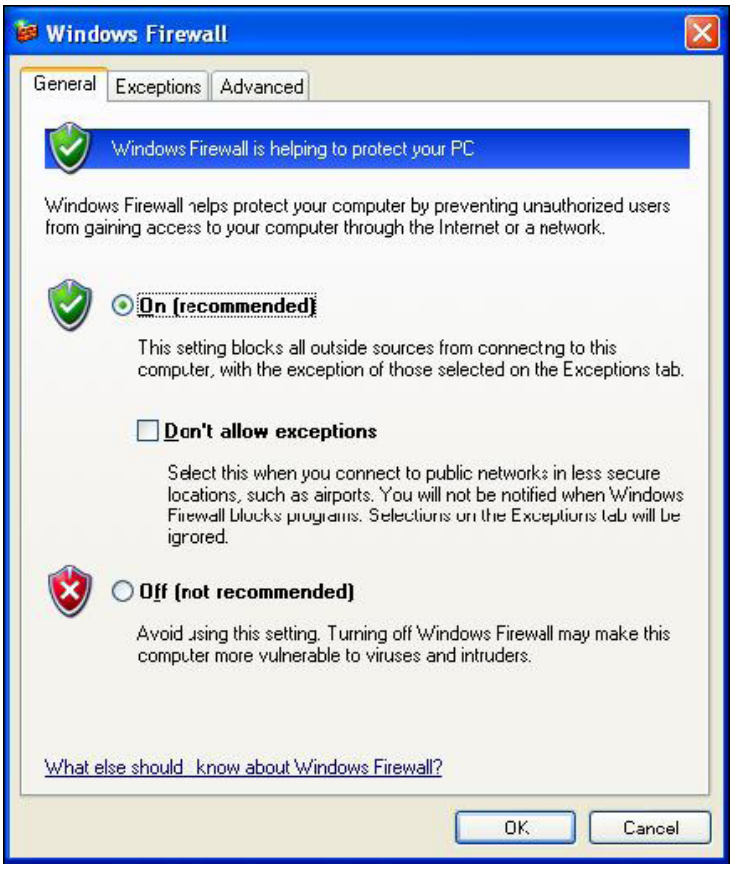
Step 12: Restart PC

3.2.3 Configuring On the Client Site (XPAC)

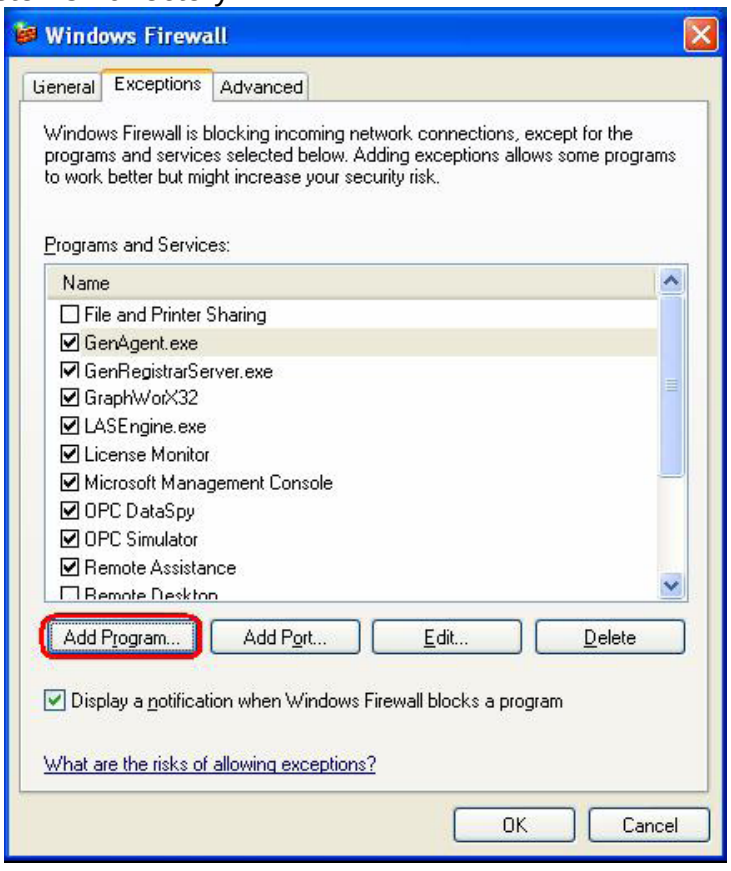
Configuring the Firewall

Step1: By default the windows firewall is set to "On". This setting is recommended by Microsoft and by OPC to give your machine the highest possible protection. For trouble shooting, you may wish to temporarily turn off the firewall to prove or disprove that the firewall configuration is the source of any communication failure.

Note: It may be appropriate to permanently turn off the firewall if the machine is sufficiently protected behind a corporate firewall. When turned off, the individual firewall settings outlined here need not be performed to allow OPC communication.

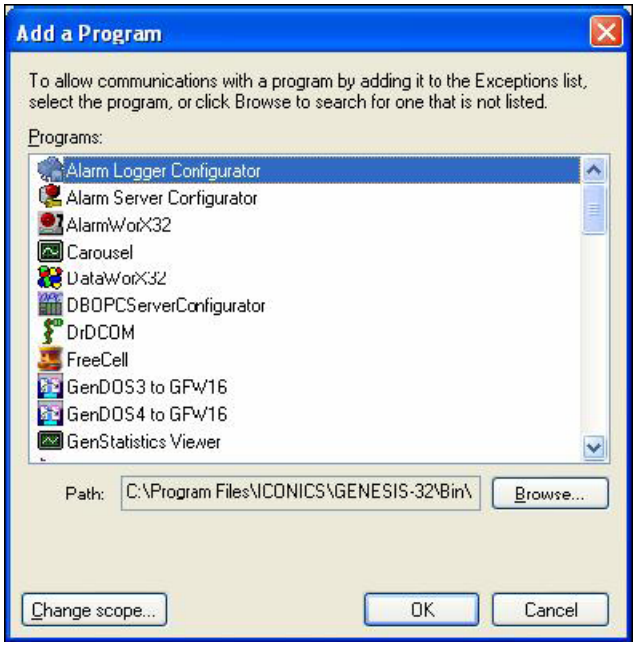


Step 2: Select the .Exceptions tab and add all OPC Clients and Servers to the exception list. Also add Microsoft Management Console (used by the DCOM configuration utility in the next section) and the OPC utility OPCEnum.exe found in the Windows\System32 directory.

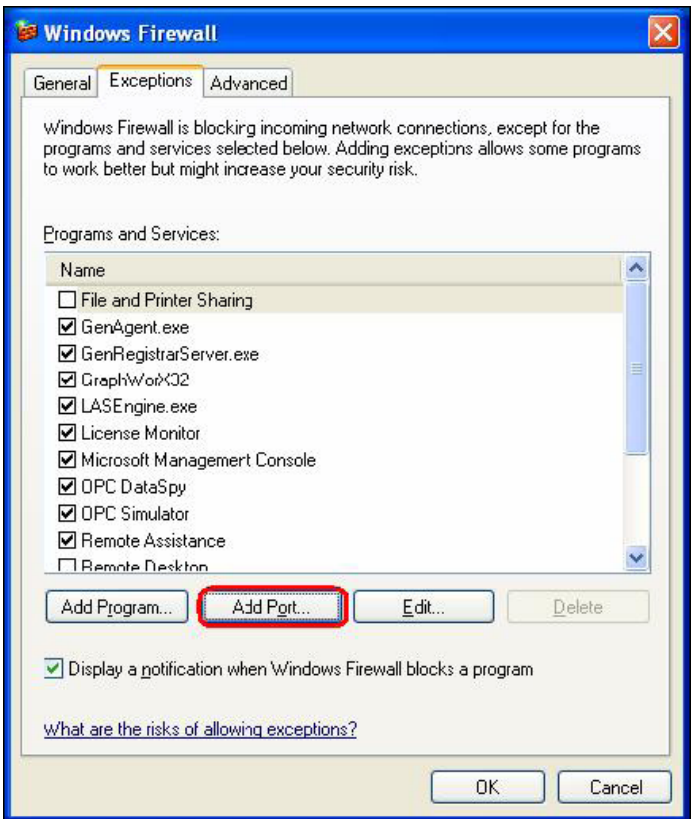


In the Add a Program dialog, there is a listing of most applications on the machine, but note that not all of them show up on this list. Use the "Browse" button to find other executables installed on the computer.

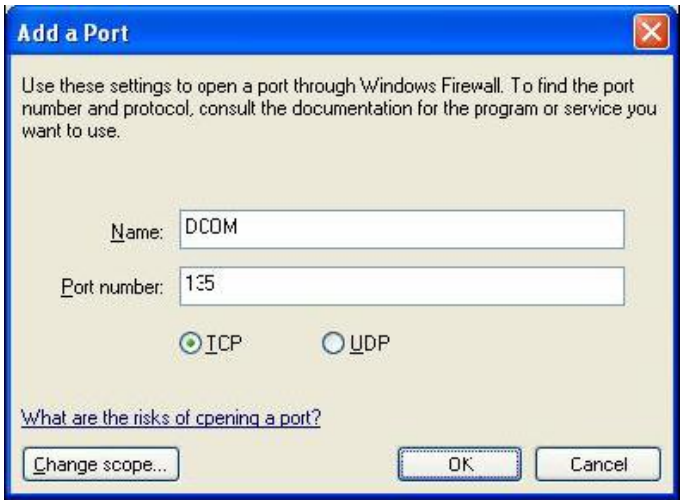
Note: Only EXE files are added to the exceptions list. For in-process OPC Servers and Clients (DLLs and OCXs) you will need to add the EXE applications that call them to the list instead.



Step 3: Add TCP port 135 as it is needed to initiate DCOM communications, and allow for incoming echo requests. In the Exceptions tab of the Windows Firewall, click on Add Port.

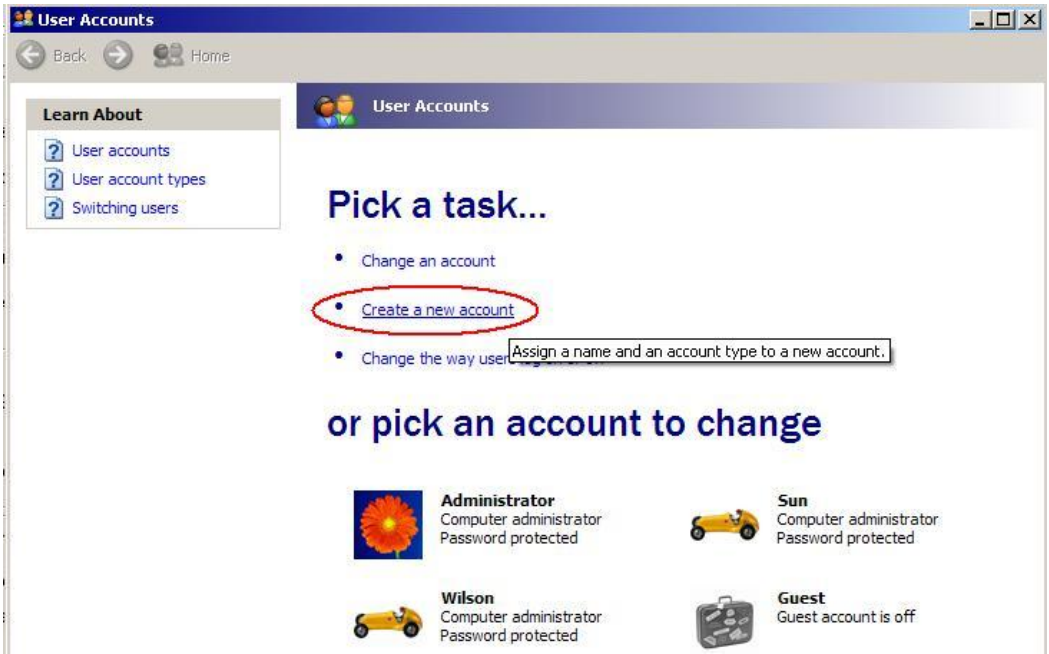


In the Add a Port dialog, fill out the fields as follows:
Name: DCOM
Port number: 135
Choose the TCP radio button



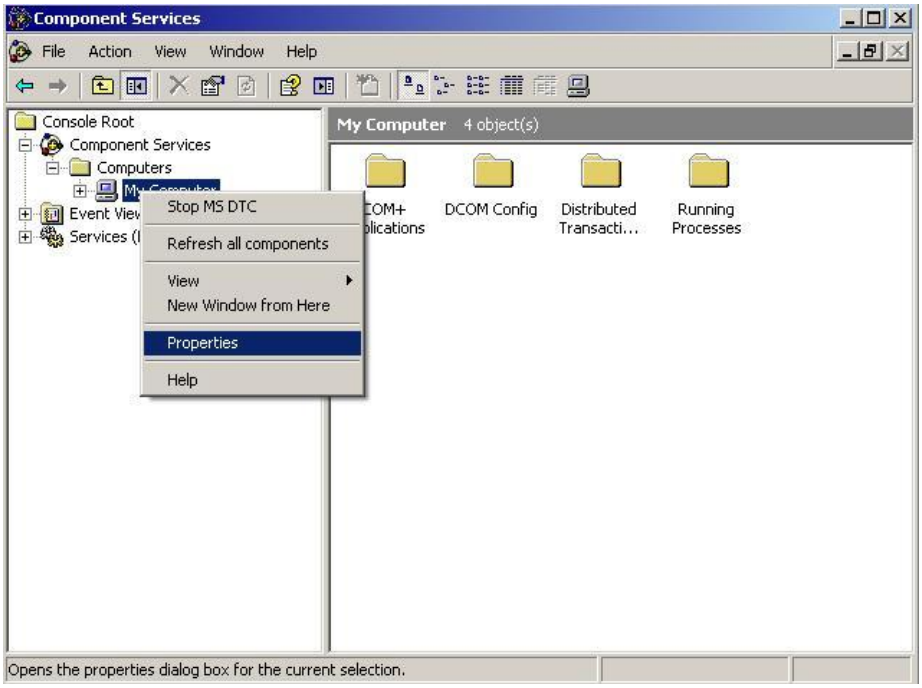
Creating the Account

Step 1: Create a account which must be the same with the account of server site.



Configuring DCOM

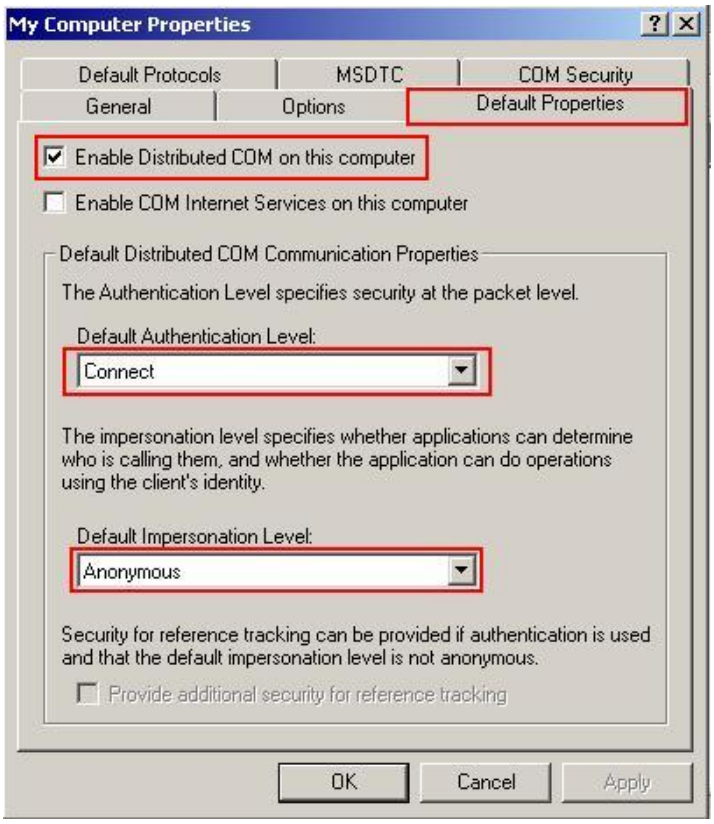
Step 1: Run the dcomcnfg.exe program to launch component services. Right clieck "My Computer" and choose "Properties".



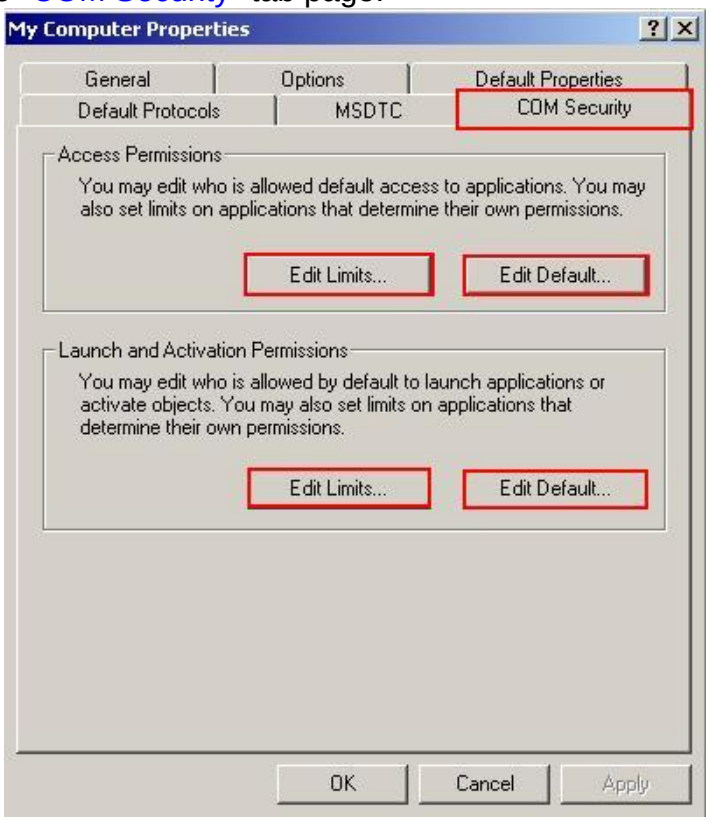
Step 2: Select the "Default Properties" tab page.

Step 3: Use the following settings:

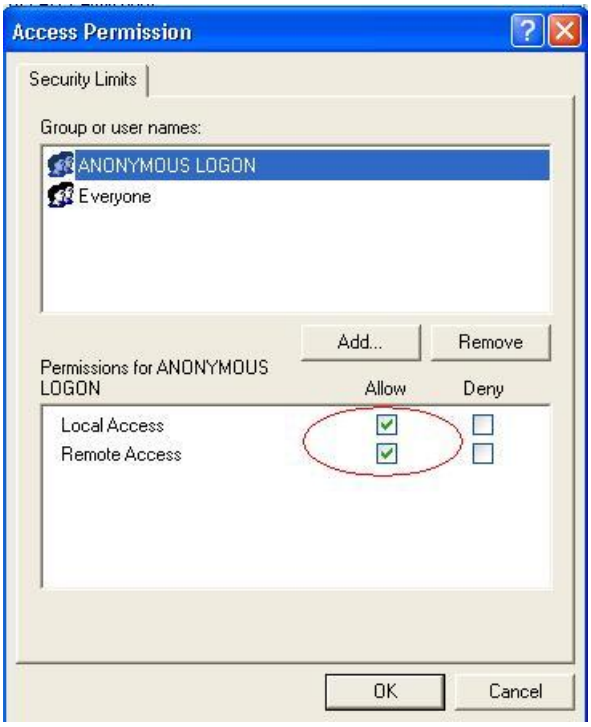
Field Name	Set to
Enable Distributed COM on this computer	Checked
Default Authentication Level:	Connect
Default Impersonation Level:	Anonymous

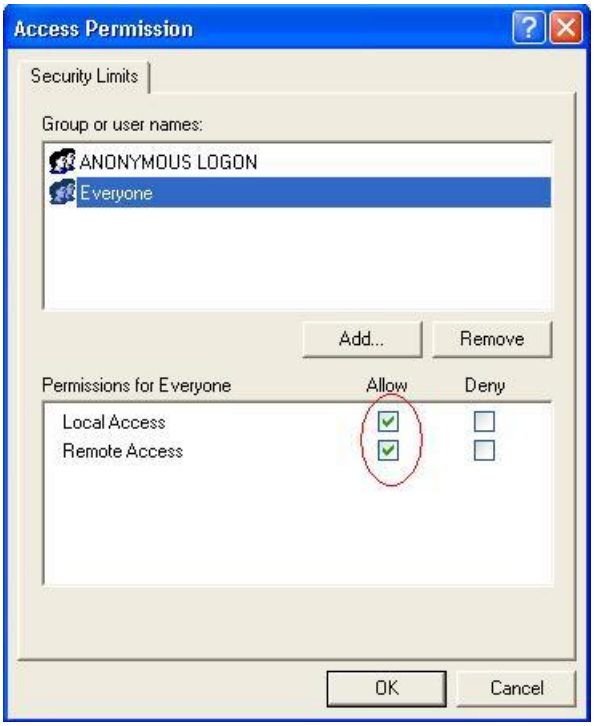


Step 4: Select the "COM Security" tab page.

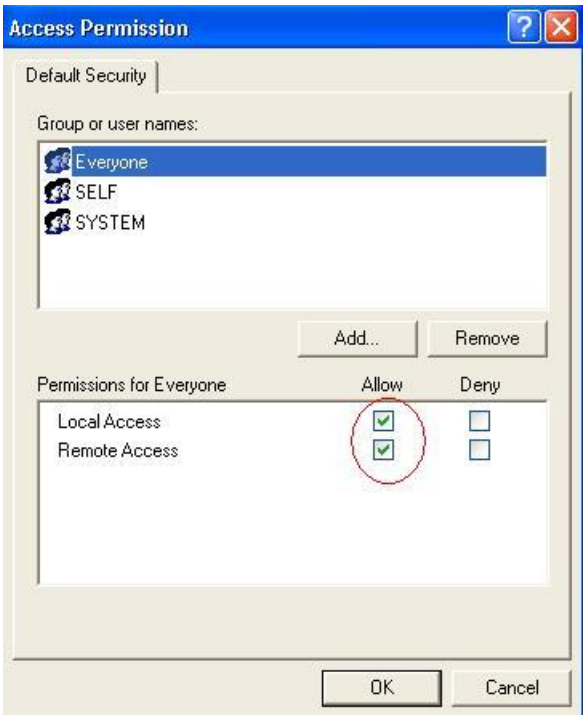


Step 5: Click on the "Edit Limits…" of "Access Permissions" button to set.

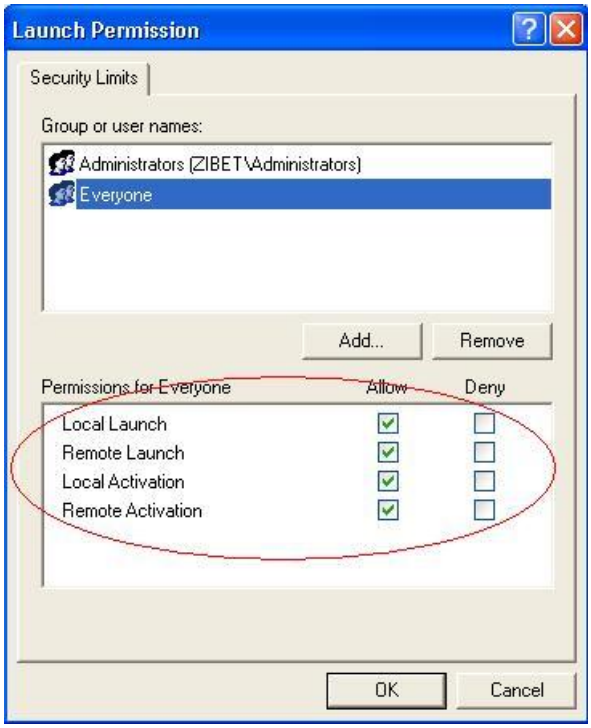




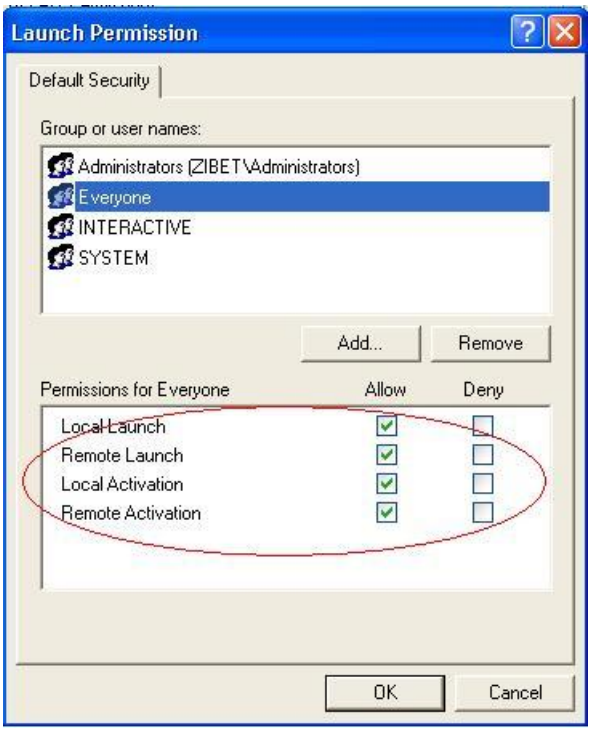
Step 6: Click on the "Edit Default…" of "Access Permissions" button to set.



Step 7: Click on the "Edit Limits…" of "Launch and Activation Permissions" button to set.

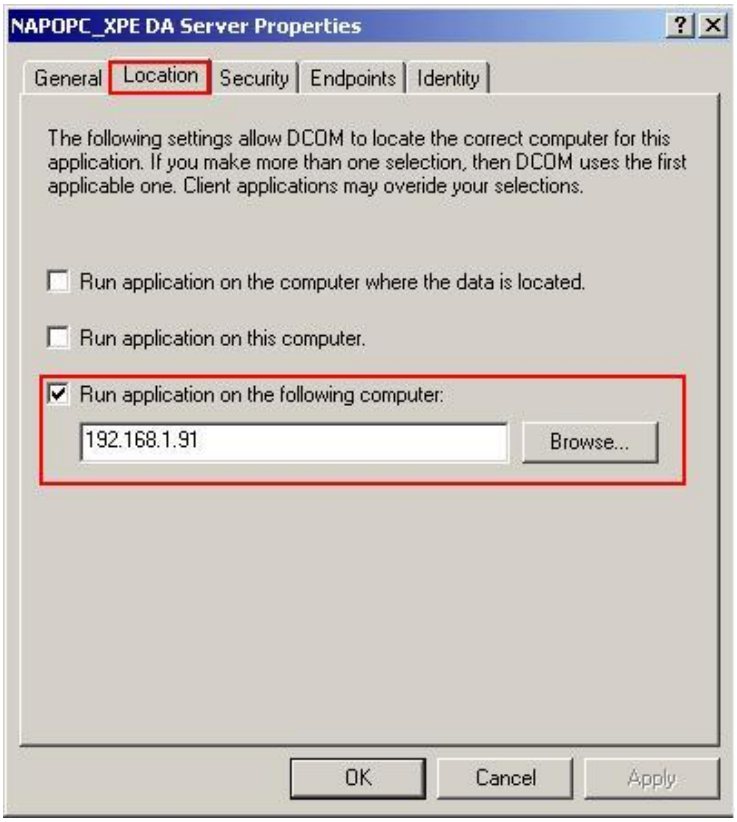


Step 8: Click on the "Edit Default…" of "Launch and Activation Permissions" button to set.



Step 9: Right click on the "NAPOPC_XPE DA Server" of "DCOM Config" button and select "Properties".

Step 10: Select the "Location" tab page and check "Run application on the following computer". And enter the Server IP here.



Step 11: Select the "Identity" tab page and check "The interactive user"

Step 12: Restart XPC

3.2.4 Configuring On the Client Site (XP-8000-CE6)

System Requirement

OS version:

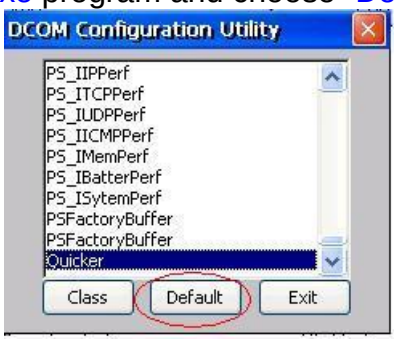
XP-8000-CE6 OS 1.3.1.0 or later

Program:

NAPOPC_CE6
DCOMCnfg.exe
NTLMUser.exe

Configuring DCOM

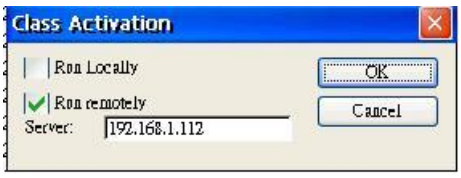
Step 1: Run the dcomcnfg.exe program and choose "Default".



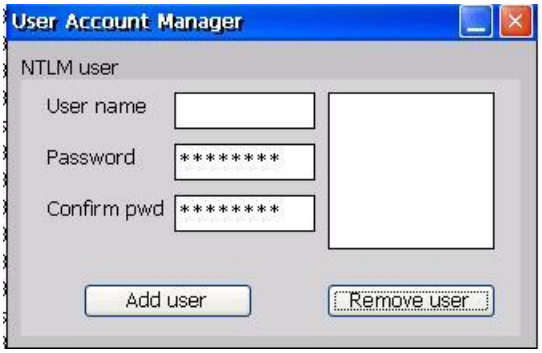
Step 2: Select the "Access" button to add an account which is identical to the account on the server site.

Step 3: Select the "Launch" button to add an account which is identical to the account on the server site as above.

Step 4: Select "Class" button of "DCOM Configuration Utility" to setup "Class Activation". Uncheck "Run Locally" and check "Run remotelly". Enter IP address of server site.

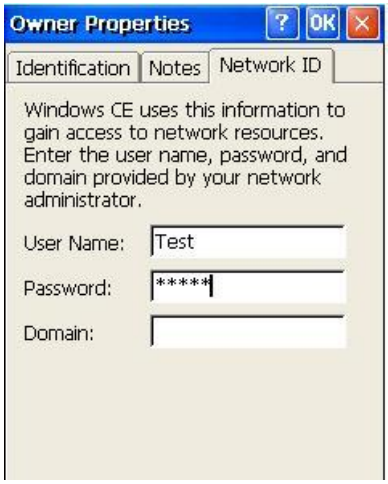


Step 5: Execute NTLMUser.exe



Step 6: Fill out "User name", "Password", and "Confirm pwd" which is identical to the account on the server site.

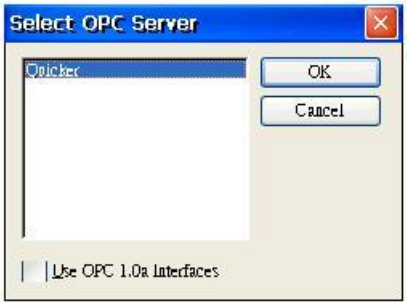
Step 7: Select "Control Panel" → "Owner Properties" → "Network ID" and fill out the User name/Password which is identical to the account on the server site.



Step 8: Run XPAC Utility to save and reboot.

Step 9: Execute OPC client for testing.


File
OPC
View
Help
Tag
Value
Qual Flag


Select OPC Server
Opicker
OK
Cancel
Use OPC 1.0a Interfaces

4 The Application of NAPOPC_CE6

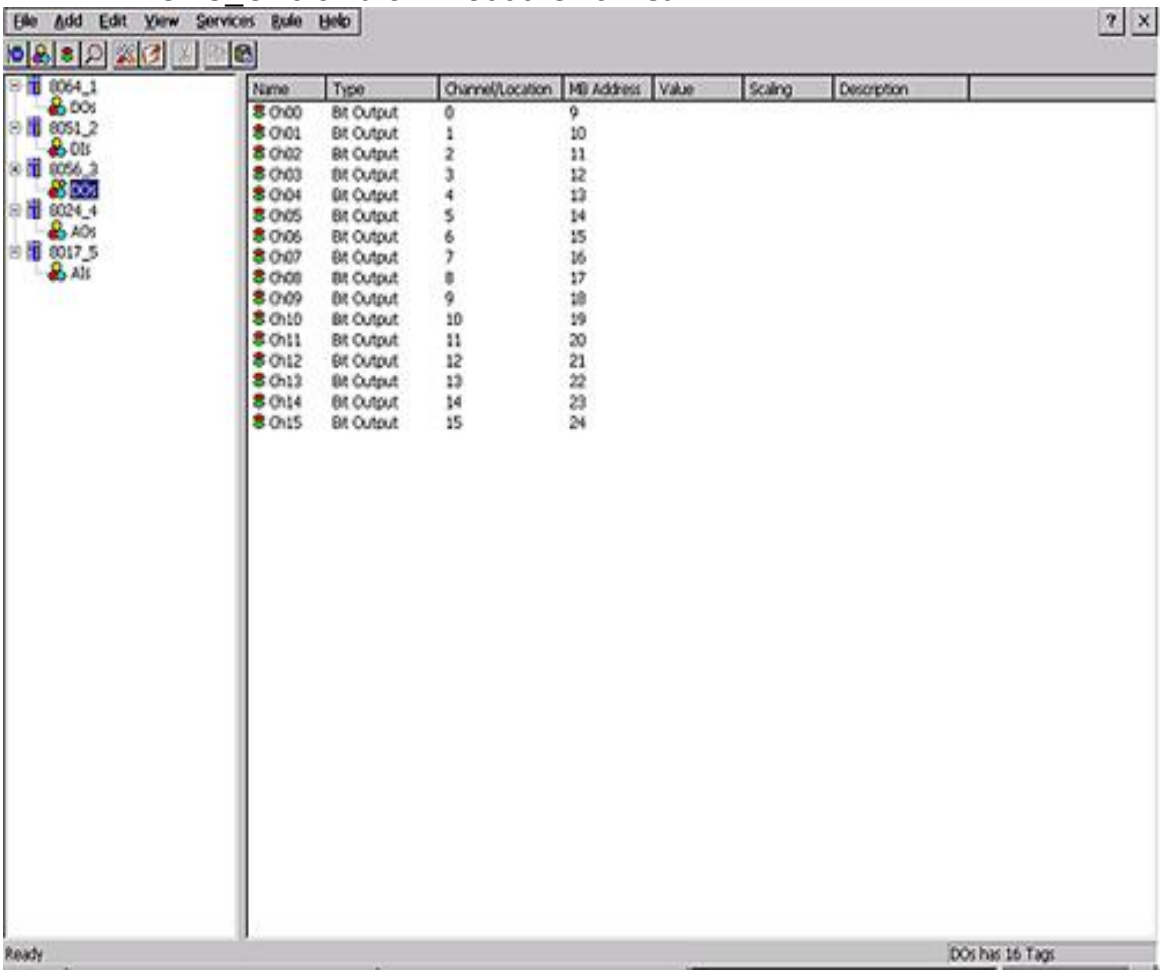
User can develop an incredible application combining with OPC client, Modbus RTU/TCP client, NAPOPC_ST, and NAPOPC_XPE. If using "Rule Script" inside the NAPOPC_CE6, user can not only save lots of time developing system, but also create a more stable and safer system. The five sections below describe the timing and method to apply in different kind of situation.

4.1 NAPOPC_CE6 with OPC Client

NAPOPC_CE6 is designed as OPC based architecture, therefore it supports OPC client naturally. Many WinCE based OPC clients in the world can apply with it. Please refer to its user manual for detail information. The following sections show you how "InduSoft Web Studio Version 6.0" connects to Quicker.

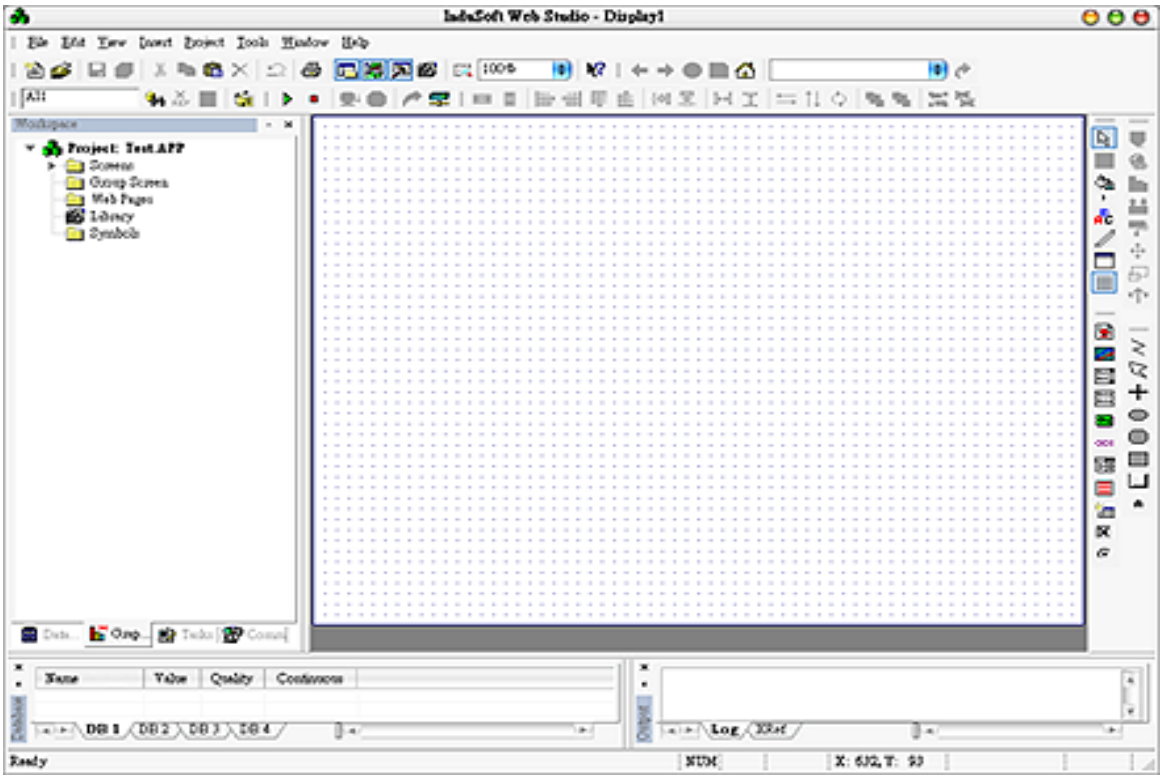
InduSoft Web Studio is a powerful, integrated collection of automation tools that includes all the building blocks needed to develop human machine interfaces (HMIs), supervisory control and data acquisition (SCADA) systems, and embedded instrumentation and control applications. Web Studio runs in native Windows NT, 2000, XP and CE.Net 5.0 environments and conforms to industry standards such as Microsoft DNA, OPC, DDE, ODBC, XML, SOAP and ActiveX. For more information please visit: http://www.indusoft.com/

Step 1: Before using the InduSoft OPC Client module, you need to configure the NAPOPC_CE6 on the XP-8000-CE6 first.

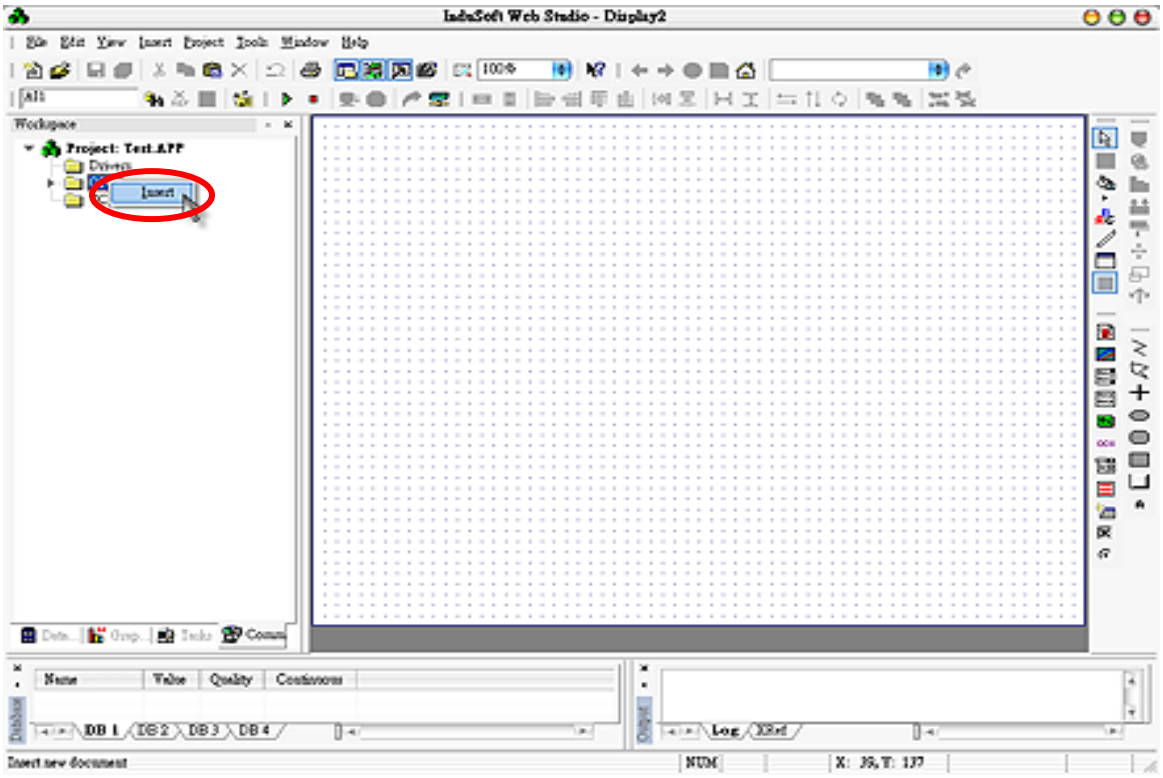


Step 2: Run InduSoft Web Studio version 6.0

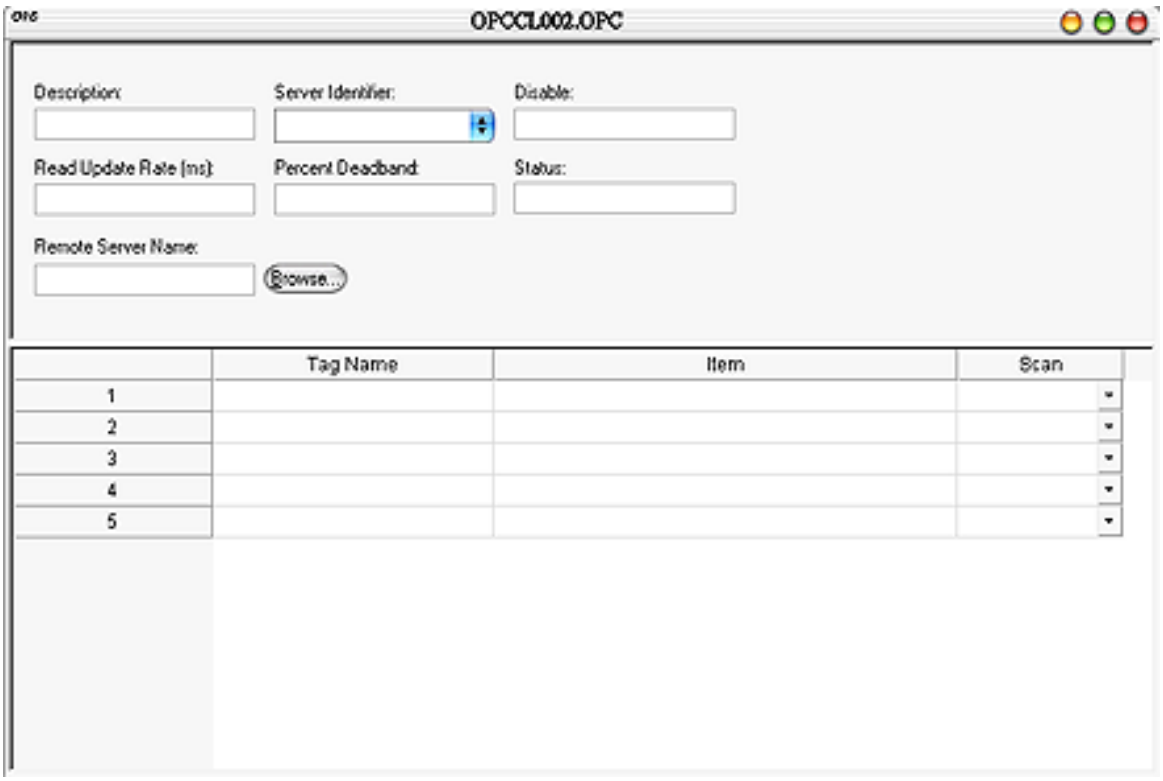
Step 3: Create a new project.



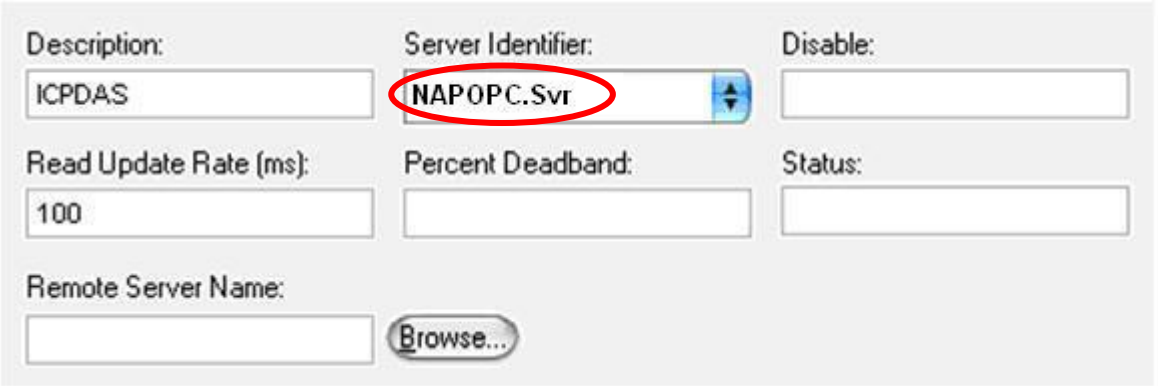
Step 4: In the Studio Workspace window, click the OPC tab, right-click the OPC folder, and click "Insert":



Step 5: OPC Attributes window pops up.



Step 6: Click on the Server Identifier: Write "NAPOPC.Svr".

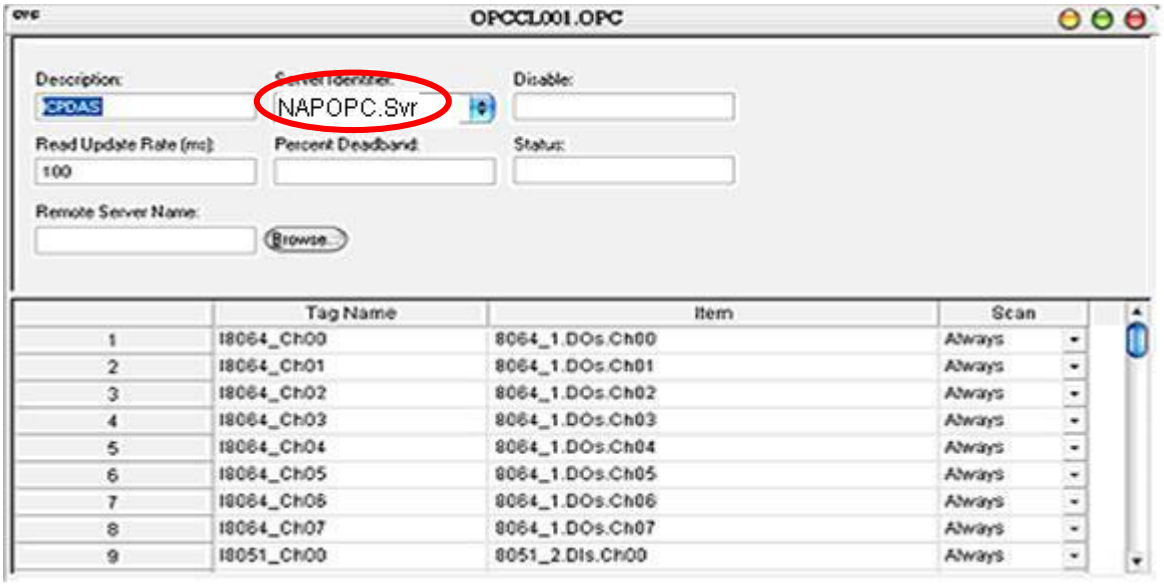


The configuration table for OPC has the following entries:

- Description: this field is used for documentation only. The OPC Client module ignores it.
- Server Identifier: this field should contain the name of the server you want to connect. If the server is installed in the computer, its name can be selected through the list box.
- Disable: this field should contain a tag or a constant. If its value is different of zero, the communication with the OPC server is disabled.
- Update Rate: this field indicates how often the server will update this group in milliseconds. If it is zero indicates the server should use the fastest practical rate.
- Percent Deadband: this field indicates the percent change in an item value that will cause a notification by the server. It's only valid for analog items.
- Tag Name: these fields should contain the tags linked to the server items.
- Item: these fields should contain the name of the server's items

Step 7: In the first cell of the Tag Name column type the tag name created in database.

Step 8: In the first cell of the item, you have to write it the same as the NAPOPC_CE6 configuration. Please refer to the demo at "CD:\Compact Flash\NAPOPC_CE6\Demo\InduSoft\Full"

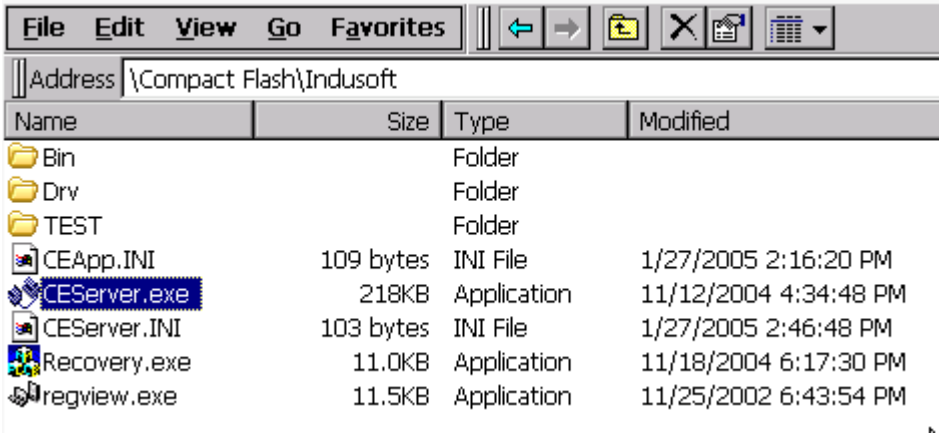


Step 9: Repeat the step between 7 and 8 to add more tags.

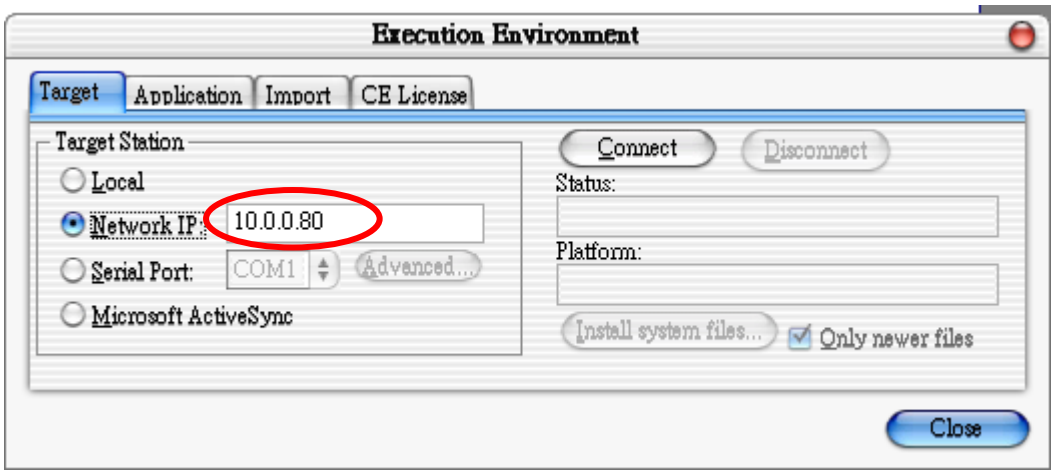
Step 10: Creating a Text String for the Input/Output Dynamic. Click the Text icon on the Object Editing toolbar. Position the crosshairs in the MAIN.SCR. Press the"#" key three times to display "###" in the gray square.

Step 11: Click the Text Input/Output property icon on the Object Editing toolbar. *Text I/O* appears in the drop-down menu of the Object Properties window. In the Tag/Expression field type the tag name you want to link.

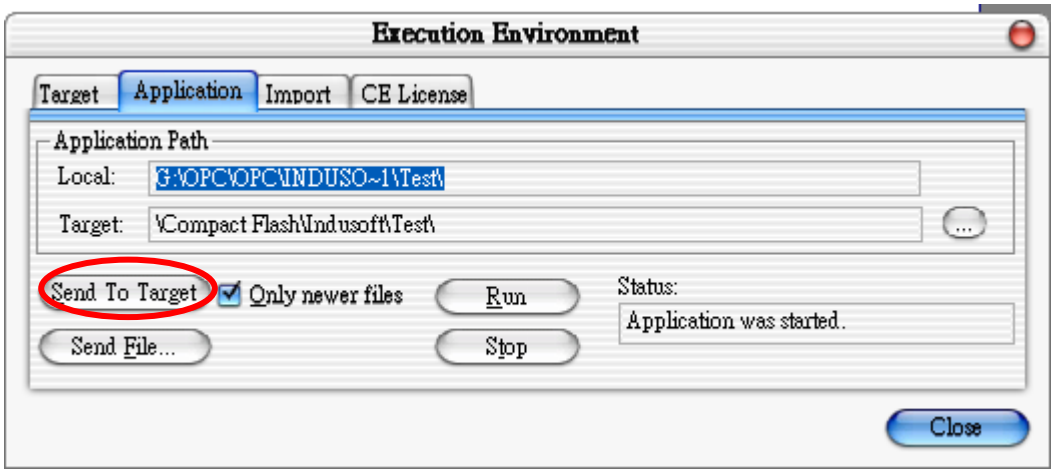
Step 12: After you finish the configuration. Execute the InduSoft Remote Agent by clicking "Compact Flash\Indusoft\CEServer.exe"



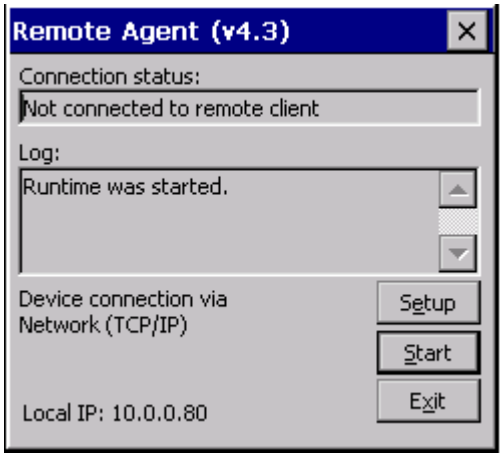
Step 13: Click "Project → Execution Environment" then select "Network IP" to press the IP of XP-8000-CE6.

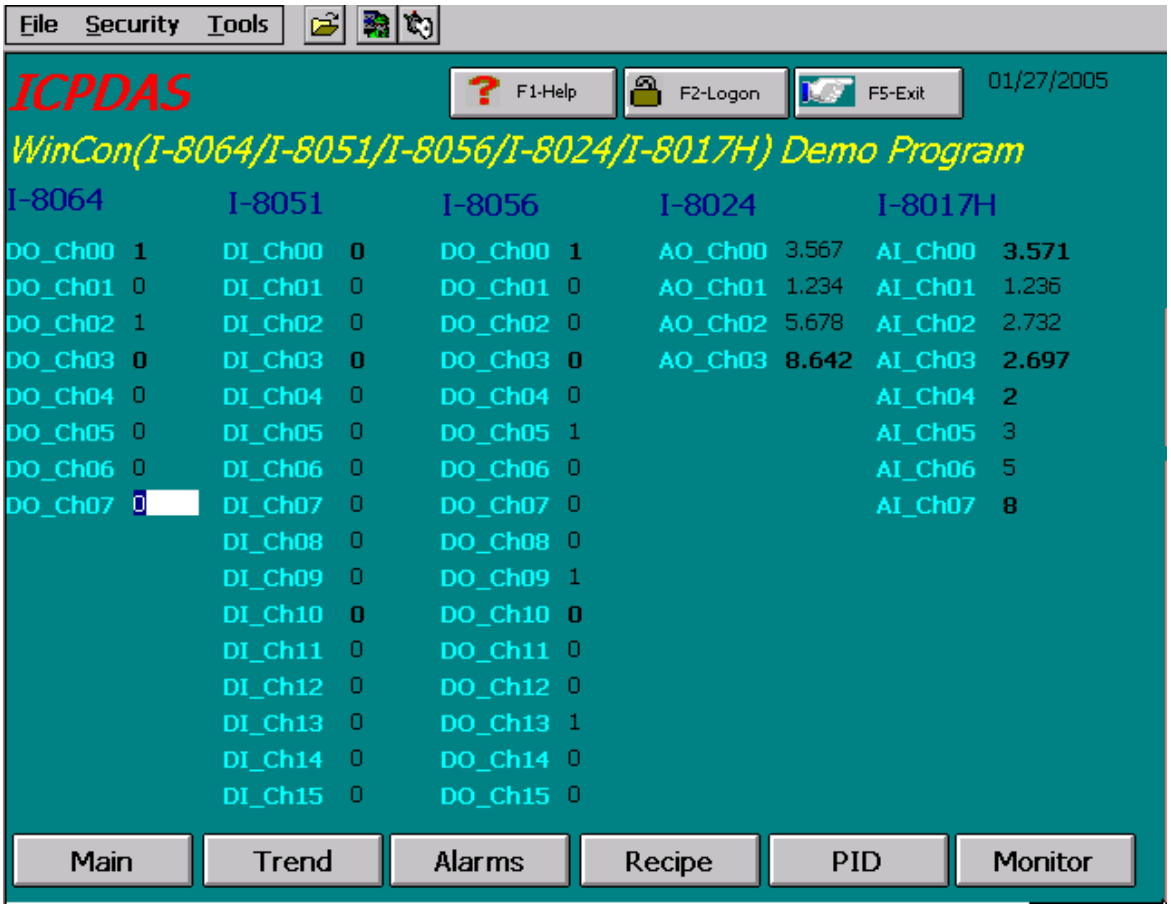


Step 14: Click "Connect" then select "Application → Send to Target"



Step 15: Execute your application by clicking "Start". After that, you will see your runtime HMI.





4.2 NAPOPC_CE6 with Modbus RTU/TCP Client

If the third party software which supports Modbus RTU/TCP client wants to connect to NAPOPC_CE6, just remember to check the services "Modbus RTU" and "Modbus TCP". Please refer to the user manual of the third party made for setting. And for NAPOPC_CE6, please refer to the section "1.2.11 Services Setup".

4.2.1 Supported Modbus Commands

The Modbus protocol establishes the format for the master's query by placing into the device (or broadcast) address, a function code defining the requested action, any data to be sent, and an error checking field. The slave's response message is also constructed using the Modbus protocol. It contains fields confirming the action

taken, any data to be returned, and an error-checking field. If an error occurred in receipt of the message, or if the slave is unable to perform the requested action, the slave will construct an error message and send it as its response.

Code Description I/O Unit Min Max					
Code	**Description**	**I/O**	**Unit**	**Min**	**Max**
01(0x01)	Read Coil	Status In	Bit	1	2000(0x7D0)
02(0x02)	Read Discrete Inputs	Status In	Bit	1	2000(0x7D0)
03(0x03)	Read Holding Registers	Registers In	Word	1	125(0x7D)
04(0x04)	Read Input Registers	Registers In	Word	1	125(0x7D)
05(0x05)	Write Single Coil	Coil Out	Bit	1	1
06(0x06)	Write Single Register	Register Out	Word	1	1
15(0x0F)	Write Multiple Coils	Coils Out Bit	Bit	1	800
16(0x10)	Write Multiple registers	Registers Out Word	Word	1	100

4.3 NAPOPC_CE6 with NAPOPC_ST/NAPOPC_XPE

You can construct a complete control system from top to bottom via NAPOPC_CE6 combining with NAPOPC_ST/NAPOPC_XPE and SCADA software. Please refer to the "1.2.11 Services Setup" to set up NAPOPC_CE6 services depending on which communication way that NAPOPC_ST/NAPOPC_XPE used. NAPOPC_CE6 provides three ways, "Modbus TCP", "Modbus RTU", and "RPC Server", to communicate with NAPOPC_ST/NAPOPC_XPE. At NAPOPC_ST/ NAPOPC_XPE site, please refer to the "Adding A New Modbus TCP Controller", " Adding A New Modbus RTU Controller" and "Adding A New RPC Controller" in the NAPOPC_ST/NAPOPC_XPE user manual.

4.4 NAPOPC_CE6 with User Application

Users can develop their own application program with VC++, VB.NET, or VC#.NET and share data with NAPOPC_CE6 via Quicker API. User can use the Modbus RTU/TCP services, or just use the share memory inside NAPOPC_CE6 to exchange data between different programs. We do not focus on the programming skill of VC++/VB.NET/VC#.NET. We just focus on the Quicker API below.

4.4.1 Quicker API for VC++ Developer

Step 1:

Install XPacSDK_CE.msi. You can find it at:
CD root\ICPDAS\XP-8000-CE6\SDK\PlatformSDK (in the companion CD)
CF Card root\SDK\PlatformSDK (in the companion compact flash)
ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/sdk/platformsdk/

Step 2:

Start Microsoft Visual Studio 2005/2008.

Step 3:

Create a new VC++ project with choosing "Smart Device" option and follow the wizard to finish creating project procedure.

Step 4:

#include "WinConAgent.h"

Step 5:

Refer to the following functions to design your own program

Step 6:

Build your project with release mode.

Note: Quicker.dll and VC++ application program must be copied to the same folder in the XP-8000-CE6

System Function
unsigned char StartQuicker(unsigned char iMode) unsigned char StopQuicker(void) unsigned char GetVersion()

QuickerIO Function

unsigned char GetDIO(unsigned short iMBAddr, unsigned char *iRecv, unsigned char iAttribute);
unsigned char GetAIO_Short(unsigned short iMBAddr, short *iRecv, unsigned char iAttribute);
unsigned char GetAIO_Long(unsigned short iMBAddr, flong *iRecv, unsigned char iAttribute);
unsigned char GetAIO_Float(unsigned short iMBAddr, float *iRecv, unsigned char iAttribute);
unsigned char GetAIO_Word(unsigned short iMBAddr, unsigned short *iRecv, unsigned char iAttribute);
unsigned char GetAIO_DWord(unsigned short iMBAddr, unsigned long *iRecv, unsigned char iAttribute);
unsigned char SetDO(unsigned short iMBAddr, unsigned char iSend);
unsigned char SetAO_Short(unsigned short iMBAddr, short *iSend);
unsigned char SetAO_Long(unsigned short iMBAddr, long *iSend);
unsigned char SetAO_Float(unsigned short iMBAddr, float *iSend);
unsigned char SetAO_Word(unsigned short iMBAddr, unsigned short *iSend);
unsigned char SetAO_DWord(unsigned short iMBAddr, unsigned long *iSend);

Modbus Function
unsigned char MBSetCoil(unsigned short iMBAddress, unsigned char iStatus, unsigned char iAttr) unsigned char MBGetCoil(unsigned short iMBAddress, unsigned char *iStatus, unsigned char iAttr) unsigned char MBSetReg(unsigned short iMBAddress, short iStatus, unsigned char iAttr) unsigned char MBGetReg(unsigned short iMBAddress, short *iStatus, unsigned char iAttr) unsigned char MBSetReg_Long(unsigned short iMBAddress, long iStatus, unsigned char iAttr) unsigned char MBGetReg_Long(unsigned short iMBAddress, long *iStatus, unsigned char iAttr) unsigned char MBSetReg_DWord(unsigned short iMBAddress, unsigned long iStatus, unsigned char iAttr) unsigned char MBGetReg_DWord(unsigned short iMBAddress, unsigned long *iStatus, unsigned char iAttr)

UserShare Function

unsigned char UserSetCoil(unsigned short iUserAddress, unsigned char iStatus);
unsigned char UserGetCoil(unsigned short iUserAddress, unsigned char *iStatus);
unsigned char UserSetReg_Str(unsigned short iUserAddress, char *iStatus);
unsigned char UserGetReg_Str(unsigned short iUserAddress, char *iStatus);
unsigned char UserSetReg_Float(unsigned short iUserAddress, float *iStatus);
unsigned char UserGetReg_Float(unsigned short iUserAddress, float *iStatus);
unsigned char UserSetReg_Short(unsigned short iUserAddress, short *iStatus);
unsigned char UserGetReg_Short(unsigned short iUserAddress, short *iStatus);
unsigned char UserSetReg_Long(unsigned short iUserAddress, long *iStatus);
unsigned char UserGetReg_Long(unsigned short iUserAddress, long *iStatus);

4.4.1.1 System Function

This group provides three functions for users to start and stop the "NAPOPCSvr_CE6.exe" and get NAPOPC_CE6 version before using "QuickerIO Function" and "Modbus Function".

StartQuicker

This function launches the NAPOPC_CE6 with different mode.

Syntax

```
[VC++]
unsigned char StartQuicker(unsigned char iMode)
```

```
[VB.NET/VC#.NET]
byte Quicker.System.StartQuicker(byte iMode)
```

Parameters

iMode

[in] The decimal number of kernel mode. It is always 1 now. It will provide another mode in the future.

Return Values

0 indicates success. If the NAPOPC_CE6 has been run, the function will return mode number. (Please refer to the Appendix 2.1)

Remarks

You **have to** call this function to launch the NAPOPC_CE6 before using the QuickerIO and Modbus functions.

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Start up the NAPOPC_CE6 with mode 1
if (StartQuicker(1) == 0){
        AfxMessageBox(_T("Start NAPOPC_CE6 successfully!"));
}
else{
        AfxMessageBox(_T("NAPOPC_CE6 has been started!"));
}
```

[VB.NET]

```
Quicker.System.StartQuicker(1)
```

[VC#.NET]

```
Quicker.System.StartQuicker(1)
```

StopQuicker

This function stops the NAPOPC_CE6.

Syntax

[VC++]

```
unsigned char StopQuicker(void)
```

[VB.NET/VC#.NET]

```
byte Quicker.System.StopQuicker()
```

Parameters

Return Values

0 indicates success. **WCA_Stop** means NAPOPC_CE6 has been stopped. **WCA_NOT_MASTER** means not the main AP which calls NAPOPC_CE6 (Please refer to the Appendix 2.1)

Remarks

NAPOPC_CE6 only can be stopped by the AP which launched it.

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Stop the NAPOPC_CE6
if(StopQuicker() == 0){
        AfxMessageBox(_T("Stop NAPOPC_CE6 successfully!"));
}
else if(StopQuicker() == WCA_Stop){
        AfxMessageBox(_T("NAPOPC_CE6 has been stopped!"));
}
else{
        AfxMessageBox(_T("Can not terminate the NAPOPC_CE6!"));
}
```

[VB.NET]

```
Quicker.System.StopQuicker()
```

[VC#.NET]

```
Quicker.System.StopQuicker()
```

GetVersion

This function gets the NAPOPC_CE6 version.

Syntax

[VC++]

```
unsigned char GetVersion(void)
```

[VB.NET/VC#.NET]

```
byte Quicker.System.GetVersion()
```

Parameters

Return Values

The return value means the version value. Ex. 209 means v2.09.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the NAPOPC_CE6 version
unsigned char iQversion;
iQversion = GetVersion();
```

[VB.NET]

```
Dim iQversion As Byte
iQversion = Quicker.System.GetVerison()
```

[VC#.NET]

```
byte iQversion = 0;
iQversion = Quicker.System.GetVersion();
```

4.4.1.2 QuickerIO Function

This group provides 12 functions for users to Get/Set data which's modbus address is mapping from 1 to 1000 in NAPOPCSvr_CE6. The data which's modbus address is mapping from 1 to 1000 can be accessed by OPC client and modbus master via NAPOPC_CE6.

GetDIO

This function can get a single digital I/O status from a specific modbus address.

Syntax

```
[VC++]
unsigned char GetDIO(unsigned short iMBAddr, unsigned char *iRecv,
                     unsigned char iAttribute
```

```
[VB.NET/VC#.NET]
byte GetDIO(ushort iMBAddr, out byte iRecv, byte iAttribute)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iRecv

[out] The digital status of specific tag. 1 means ON. 0 means OFF.

iAttribute

[in] Assign which kind of digital status you want get. 1 means digital input. 0 means digital output.

Return Values

0 indicates success. **WCA_ATT_ERROR** means the iAttibute is neither 0 nor 1.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the digital I/O status
//Get the digital input status from modbus address 1
unsigned char iRecvIn;
GetDIO(1,&iRecvIn,1);
//Get the digital output status from modbus address 2
unsigned char iRecvOut;
GetDIO(2,&iRecvOut,0);
```

[VB.NET]

```
Dim m_GetDIOVal As Byte
Quicker.QuickerIO.GetDIO(7, m_GetDIOVal, 0)
```

[VC#.NET]

```
byte m_GetDIOVal;
Quicker.QuickerIO.GetDIO(7,out m_GetDIOVal, 0);
```

GetAIO_Short

This function can get a single analog I/O value from a specific modbus address.

Syntax

```
[VC++]
unsigned char GetAIO_Short(unsigned short iMBAddr, short *iRecv,
                           unsigned char iAttribute)
```

```
[VB.NET/VC#.NET]
byte GetAIO_Short(ushort iMBAddr, out short fRecv, byte iAttribute)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iRecv

[out] The analog value of specific tag.

iAttribute

[in] Assign which kind of analog value you want get.

Return Values

0 indicates success. **WCA_ATT_ERROR** means the iAttibute is neither 0 nor 1.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the analog I/O value
//Get the analog input value from modbus address 1
short sRecvIn;
GetAIO_Short(1,&sRecvIn,1);
//Get the analog output value from modbus address 2
short sRecvOut;
GetAIO_Short(2,&sRecvOut,0);
```

[VB.NET]

```
Dim m_GetAIOVal As short
Quicker.QuickerIO.GetAIO_Short(7, m_GetAIOVal, 0)
```

[VC#.NET]

```
short m_GetAIOVal;
Quicker.QuickerIO.GetAIO_Short(7,out m_GetAIOVal, 0);
```

GetAIO_Long

This function can get a single analog I/O value from a specific modbus address.

Syntax

[VC++]
unsigned char GetAIO_Long(unsigned short iMBAddr, long *iRecv, unsigned char iAttribute)
[VB.NET/VC#.NET]
byte GetAIO_Long(ushort iMBAddr, out long fRecv, byte iAttribute)

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iRecv

[out] The analog value of specific tag.

iAttribute

[in] Assign which kind of analog value you want get.

Return Values

0 indicates success. **WCA_ATT_ERROR** means the iAttibute is neither 0 nor 1.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the analog I/O value
//Get the analog input value from modbus address 1
long lRecvIn;
GetAIO_Long(1,&fRecvIn,1);
//Get the analog output value from modbus address 2
long lRecvOut;
GetAIO_Long(2,&fRecvOut,0);
```

[VB.NET]

```
Dim m_GetAIOVal As long
Quicker.QuickerIO.GetAIO_Long(7, m_GetAIOVal, 0)
```

[VC#.NET]

```
long m_GetAIOVal;
Quicker.QuickerIO.GetAIO_Long(7,out m_GetAIOVal, 0);
```

GetAIO_Float

This function can get a single analog I/O value from a specific modbus address.

Syntax

[VC++]

```
unsigned char GetAIO_Float(unsigned short iMBAddr, float *iRecv,
                           unsigned char iAttribute)
```

[VB.NET/VC#.NET]

```
byte GetAIO_Float(ushort iMBAddr, out float fRecv, byte iAttribute)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iRecv

[out] The analog value of specific tag.

iAttribute

[in] Assign which kind of analog value you want get.

Return Values

0 indicates success. **WCA_ATT_ERROR** means the iAttibute is neither 0 nor 1.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the analog I/O value
//Get the analog input value from modbus address 1
float fRecvIn;
GetAIO_Float(1,&fRecvIn,1);
//Get the analog output value from modbus address 2
float fRecvOut;
GetAIO_Float(2,&fRecvOut,0);
```

[VB.NET]

```
Dim m_GetAIOVal As Single
Quicker.QuickerIO.GetAIO_Float(7, m_GetAIOVal, 0)
```

[VC#.NET]

```
float m_GetAIOVal;
Quicker.QuickerIO.GetAIO_Float(7,out m_GetAIOVal, 0);
```

GetAIO_Word

This function can get a single analog I/O value from a specific modbus address.

Syntax

[VC++]

```
unsigned char GetAIO_Word(unsigned short iMBAddr, unsigned short *iRecv,
                          unsigned char iAttribute)
```

[VB.NET/VC#.NET]

```
byte GetAIO_Word(ushort iMBAddr, out ushort fRecv, byte iAttribute)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iRecv

[out] The analog value of specific tag.

iAttribute

[in] Assign which kind of analog value you want get.

Return Values

0 indicates success. **WCA_ATT_ERROR** means the iAttibute is neither 0 nor 1.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the analog I/O value
//Get the analog input value from modbus address 1
unsigned short usRecvIn;
GetAIO_Word(1,&fRecvIn,1);
//Get the analog output value from modbus address 2
unsigned short usRecvOut;
GetAIO_Word(2,&usRecvOut,0);
```

[VB.NET]

```
Dim m_GetAIOVal As UInt16
Quicker.QuickerIO.GetAIO_Word(7, m_GetAIOVal, 0)
```

[VC#.NET]

```
ushort m_GetAIOVal;
Quicker.QuickerIO.GetAIO_Word(7,out m_GetAIOVal, 0);
```

GetAIO_DWord

This function can get a single analog I/O value from a specific modbus address.

Syntax

[VC++]

```
unsigned char GetAIO_DWord(unsigned short iMBAddr, unsigned long *iRecv,
                           unsigned char iAttribute)
```

[VB.NET/VC#.NET]

```
byte GetAIO_DWord(ushort iMBAddr, out ulong fRecv, byte iAttribute)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iRecv

[out] The analog value of specific tag.

iAttribute

[in] Assign which kind of analog value you want get.

Return Values

0 indicates success. **WCA_ATT_ERROR** means the iAttibute is neither 0 nor 1.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get the analog I/O value
//Get the analog input value from modbus address 1
unsigned long ulRecvIn;
GetAIO_DWord(1,&ulRecvIn,1);
//Get the analog output value from modbus address 2
unsigned long ulRecvOut;
GetAIO_DWord(2,&ulRecvOut,0);
```

[VB.NET]

```
Dim m_GetAIOVal As UInt64
Quicker.QuickerIO.GetAIO_DWord(7, m_GetAIOVal, 0)
```

[VC#.NET]

```
ulong m_GetAIOVal;
Quicker.QuickerIO.GetAIO_DWord(7,out m_GetAIOVal, 0);
```

SetDO

This function can set a single digital output status to a specific modbus address

Syntax

```
[VC++]
unsigned char SetDO(unsigned short iMBAddr, unsigned char iSend)
```

```
[VB.NET/VC#.NET]
byte SetDO(ushort iMBAddr, byte iSend)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iSend

[in] The digital status of specific tag. 1 means ON. 0 means OFF.

Return Values

0 indicates success.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set the digital output ON to modbus address 1
SetDO(1,1);
```

[VB.NET]

```
Dim m_SetDOVal As Byte
Quicker.QuickerIO.SetDO(1, m_SetDOVal)
```

[VC#.NET]

```
byte m_SetDOVal;
Quicker.QuickerIO.SetDO(1, m_SetDOVal);
```

SetAO_Short

This function can set a single analog output value to a specific modbus address

Syntax

```
[VC++]
unsigned char SetAO_Short(unsigned short iMBAddr, short *iSend)
```

```
[VB.NET/VC#.NET]
byte SetAO_Short(ushort iMBAddr, out short iSend)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iSend

[out] The analog value of specific tag.

Return Values

0 indicates success.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set the analog output value as 42 to modbus address 1
SetAO_Short(1,42);
```

[VB.NET]

```
Quicker.QuickerIO.SetAO_Short(1, 42)
```

[VC#.NET]

```
Quicker.QuickerIO.SetAO_Short(1, 42);
```

SetAO_Long

This function can set a single analog output value to a specific modbus address

Syntax

[VC++]

```
unsigned char SetAO_Long(unsigned short iMBAddr, long *iSend)
```

[VB.NET/VC#.NET]

```
byte SetAO_Long(ushort iMBAddr, out long iSend)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iSend

[out] The analog value of specific tag.

Return Values

0 indicates success.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set the analog output value as 2323 to modbus address 1
SetAO_Long(1,2323);
```

[VB.NET]

```
Quicker.QuickerIO.SetAO_Long(1, 2323)
```

[VC#.NET]

```
Quicker.QuickerIO.SetAO_Long(1, 2323);
```

SetAO_Float

This function can set a single analog output value to a specific modbus address

Syntax

[VC++]

```
unsigned char SetAO_Float(unsigned short iMBAddr, float *iSend)
```

[VB.NET/VC#.NET]

```
byte SetAO_Float(ushort iMBAddr, out float iSend)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iSend

[out] The analog value of specific tag.

Return Values

0 indicates success.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set the analog output value as 5.5 to modbus address 1
SetAO_Float(1,5.5);
```

[VB.NET]

```
Quicker.QuickerIO.SetAO_Float(1, 5.5)
```

[VC#.NET]

```
Quicker.QuickerIO.SetAO_Float(1, 5.5);
```

SetAO_Word

This function can set a single analog output value to a specific modbus address

Syntax

[VC++]

```
unsigned char SetAO_Word(unsigned short iMBAddr, unsigned short *iSend)
```

[VB.NET/VC#.NET]

```
byte SetAO_Word(ushort iMBAddr, out ushort iSend)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iSend

[out] The analog value of specific tag.

Return Values

0 indicates success.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set the analog output value as 222 to modbus address 1
SetAO_Word(1,222);
```

[VB.NET]

```
Quicker.QuickerIO.SetAO_Word(1, 222)
```

[VC#.NET]

```
Quicker.QuickerIO.SetAO_Word(1, 222);
```

SetAO_DWord

This function can set a single analog output value to a specific modbus address

Syntax

```
[VC++]
unsigned char SetAO_DWord(unsigned short iMBAddr, unsigned long *iSend)
```

```
[VB.NET/VC#.NET]
byte SetAO_DWord(ushort iMBAddr, out ulong iSend)
```

Parameters

iMBAddr

[in] The modbus address of specific tag in the NAPOPC_CE6. The range is from 1 to 1000.

iSend

[out] The analog value of specific tag.

Return Values

0 indicates success.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set the analog output value as 2323 to modbus address 1
SetAO_DWord(1,2323);
```

[VB.NET]

```
Quicker.QuickerIO.SetAO_DWord(1, 2323)
```

[VC#.NET]

```
Quicker.QuickerIO.SetAO_DWord(1, 2323);
```

4.4.1.3 Modbus Function

This group provides 8 functions to user to add their own variables into NAPOPC_CE6 for sharing the values to modbus client via modbus service of NAPOPC_CE6. If user create internal device and create internal tag, this data can not only be accessed by modbus client but also OPC client via NAPOPC_CE6.

MBSetCoil

The function can set a coil value into NAPOPC_CE6.

Syntax

```
[VC++]
unsigned char MBSetCoil(unsigned short iMBAddress, unsigned char iStatus,
                        unsigned char iAttr)
```

```
[VB.NET/VC#.NET]
byte MBSetCoil(ushort iMBAddress, byte iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to set into. The range of modbus address is from 1001 to 20999.

iStatus

[in] The coil status of specific modbus address. 1 means ON. 0 means OFF.

iAttr

[in] Assign which kind of coil you want set. 1 means input coil which will be requested by modbus function number 2. 0 means output coil which will be requested by modbus function number 1/5/15.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

//Set input coil status ON at address 1

[VC++]

```
MBSetCoil(1,1,1);
```

[VB.NET]

```
Quicker.Modbus.MBSetCoil(1, 1, 1)
```

[VC#.NET]

```
Quicker.Modbus.MBSetCoil(1, 1, 1);
```

MBGetCoil

The function can get a coil value from a specific modbus address.

Syntax

```
[VC++]
unsigned char MBGetCoil(unsigned short iMBAddress, unsigned char *iStatus,
                        unsigned char iAttr)
```

```
[VB.NET/VC#.NET]
byte MBGetCoil(ushort iMBAddress, out byte iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to get from. The range of modbus address is from 1001 to 20999.

iStatus

[out] The coil status of specific modbus address. 1 means ON. 0 means OFF.

iAttr

[in] Assign which kind of coil you want get. 1 means input coil which will be requested by modbus function number 2. 0 means output coil which will be requested by modbus function number 1/5/15.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get input coil status from address 1
unsigned char iStatus;
MBGetCoil(1,&iSatus,1);
```

[VB.NET]

```
Dim m_MBGetCoilVal As Byte
Quicker.Modbus.MBGetCoil(1, m_MBGetCoilVal, 1)
```

[VC#.NET]

```
byte m_MBGetCoilVal;
Quicker.Modbus.MBGetCoil(1,out m_MBGetCoilVal, 1);
```

MBSetReg

The function can set a register value into NAPOPC_CE6.

Syntax

```
[VC++]
unsigned char MBSetReg(unsigned short iMBAddress, short iStatus,
                       unsigned char iAttr)
```

```
[VB.NET/VC#.NET]
byte MBSetReg(ushort iMBAddress, short iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to set into. The range of modbus address is from 1001 to 20999.

iStatus

[in] The register value of specific modbus address.

iAttr

[in] Assign which kind of register you want set. 1 means input register which will be requested by modbus function number 4. 0 means output register which will be requested by modbus function number 3/6/16.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set input register value 123 at address 1
MBSetReg(1,123,1);
```

[VB.NET]

```
Quicker.Modbus.MBSetReg(1, 123, 1)
```

[VC#.NET]

```
Quicker.Modbus.MBSetReg(1, 123, 1) ;
```

MBGetReg

The function can get a register value from a specific modbus address.

Syntax

```
[VC++]
unsigned char MBGetReg(unsigned short iMBAddress, short *iStatus,
                       unsigned char iAttr)
```

```
[VB.NET/VC#.NET]
byte MBGetReg(ushort iMBAddress, out short iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to get from. The range of modbus address is from 1001 to 20999.

iStatus

[out] The register value of specific modbus address.

iAttr

[in] Assign which kind of register you want get. 1 means input register which will be requested by modbus function number 4. 0 means output register which will be requested by modbus function number 3/6/16.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get input register value from address 1
short iSataus;
MBGetReg(1,&iSatus,1);
```

[VB.NET]

```
Dim m_MBGetRegVal As short
Quicker.Modbus.MBGetReg(1, m_MBGetRegVal, 1)
```

[VC#.NET]

```
short m_MBGetRegVal;
Quicker.Modbus.MBGeReg(1,out m_MBGetRegVal, 1);
```

MBSetReg_Long

The function can set a register value into NAPOPC_CE6.

Syntax

[VC++]

```
unsigned char MBSetReg_Long(unsigned short iMBAddress, long iStatus,
                            unsigned char iAttr)
```

[VB.NET/VC#.NET]

```
byte MBSetReg(ushort iMBAddress, int iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to set into. The range of modbus address is from 1001 to 20999.

iStatus

[in] The register value of specific modbus address.

iAttr

[in] Assign which kind of register you want set. 1 means input register which will be requested by modbus function number 4. 0 means output register which will be requested by modbus function number 3/6/16.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set input register value 123 at address 1
MBSetReg_Long(1,123,1);
```

[VB.NET]

```
Quicker.Modbus.MBSetReg_Long(1, 123, 1)
```

[VC#.NET]

```
Quicker.Modbus.MBSetReg_Long(1, 123, 1) ;
```

MBGetReg_Long

The function can get a register value from a specific modbus address.

Syntax

```
[VC++]
unsigned char MBGetReg_Long(unsigned short iMBAddress, long *iStatus,
                            unsigned char iAttr)
```

```
[VB.NET/VC#.NET]
byte MBGetReg_Long(ushort iMBAddress, out int iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to get from. The range of modbus address is from 1001 to 20999.

iStatus

[out] The register value of specific modbus address.

iAttr

[in] Assign which kind of register you want get. 1 means input register which will be requested by modbus function number 4. 0 means output register which will be requested by modbus function number 3/6/16.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get input register value from address 1
long iSataus;
MBGetReg_Long(1,&iSatus,1);
```

[VB.NET]

```
Dim m_MBGetRegVal As Integer
Quicker.Modbus.MBGetReg_Long(1, m_MBGetRegVal, 1)
```

[VC#.NET]

```
int m_MBGetRegVal;
Quicker.Modbus.MBGeReg_Long(1,out m_MBGetRegVal, 1);
```

MBSetReg_DWord

The function can set a register value into NAPOPC_CE6.

Syntax

[VC++]

unsigned char MBSetReg_DWord(unsigned short iMBAddress, unsigned long iStatus, unsigned char iAttr)

[VB.NET/VC#.NET]

byte MBSetReg(ushort iMBAddress, uint iStatus, byte iAttr)

Parameters

iMBAddress

[in] The modbus address which you want to set into. The range of modbus address is from 1001 to 20999.

iStatus

[in] The register value of specific modbus address.

iAttr

[in] Assign which kind of register you want set. 1 means input register which will be requested by modbus function number 4. 0 means output register which will be requested by modbus function number 3/6/16.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set input register value 123 at address 1
MBSetReg_DWord(1,123,1);
```

[VB.NET]

```
Quicker.Modbus.MBSetReg_DWord(1, 123, 1)
```

[VC#.NET]

```
Quicker.Modbus.MBSetReg_DWord(1, 123, 1) ;
```

MBGetReg_DWord

The function can get a register value from a specific modbus address.

Syntax

[VC++]

```
unsigned char MBGetReg_DWord(unsigned short iMBAddress, unsigned long
*iStatus, unsigned char iAttr)
```

[VB.NET/VC#.NET]

```
byte MBGetReg_DWord(ushort iMBAddress, out uint iStatus, byte iAttr)
```

Parameters

iMBAddress

[in] The modbus address which you want to get from. The range of modbus address is from 1001 to 20999.

iStatus

[out] The register value of specific modbus address.

iAttr

[in] Assign which kind of register you want get. 1 means input register which will be requested by modbus function number 4. 0 means output register which will be requested by modbus function number 3/6/16.

Return Values

0 indicates success. **WCA_MBADDR_OVER** means the iMBAddress over the range. The legal range is from number 1001 to number 20999. **WCA_MBATTR_ERROR** means the iAttr is neither 1 nor 0.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get input register value from address 1
unsigned long iSataus;
MBGetReg_DWord(1,&iSatus,1);
```

[VB.NET]

```
Dim m_MBGetRegVal As UInt32
Quicker.Modbus.MBGetReg_DWord(1, m_MBGetRegVal, 1)
```

[VC#.NET]

```
uint m_MBGetRegVal;
Quicker.Modbus.MBGeReg_DWord(1,out m_MBGetRegVal, 1);
```

4.4.1.4 UserShare Function

These functions allow users to add their own variables into share memory block for sharing the values with different application program. The data using these functions can not be accessed by modbus client and OPC client.

UerSetCoil

The function can set an unsigned char variable into share memory block.

Syntax

```
[VC++]
unsigned char UserSetCoil(unsigned short iUserAddress, unsigned char iStatus)
```

```
[VB.NET/VC#.NET]
byte UserSetCoil(ushort iUserAddress, byte iStatus)
```

Parameters

iUserAddress

[in] The address which you want to set into. The range of address is from 1 to 19999.

iStatus

[in] unsigned char variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set coil value into address 1
UserSetCoil(1,1);
```

[VB.NET]

```
Quicker.UserShare.UserSetCoil(1, 1)
```

[VC#.NET]

```
Quicker.UserShare.UserSetCoil(1, 1);
```

UserGetCoil

The function can get an unsigned char variable from share memory block.

Syntax

```
[VC++]
unsigned char UserGetCoil(unsigned short iUserAddress, unsigned char *iStatus)
```

```
[VB.NET/VC#.NET]
byte UserGetCoil(ushort iUserAddress, out byte iStatus)
```

Parameters

iUserAddress

[in] The address which you want to get from. The range of address is from 1 to 19999.

iStatus

[out] The pointer to an unsigned char variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get coil value from address 1
unsigned char iStatus;
UserGetCoil(1,&iSatus);
```

[VB.NET]

```
Dim m_UserGetCoilVal As Byte
Quicker.UserShare.UserGetCoil(1, m_UserGetCoilVal)
```

[VC#.NET]

```
byte m_UserGetCoilVal;
Quicker.UserShare.UserGetCoil(1,out m_UserGetCoilVal);
```

UserSetReg_Str

The function can set a string variable into share memory block.

Syntax

[VC++]
unsigned char UserSetReg_Str(unsigned short iUserAddress, char *iStatus)
[VB.NET/VC#.NET]
byte UserSetReg_Str(ushort iUserAddress, char[] cSetStr)

Parameters

iUserAddress

[in] The address which you want to set into. The range of address is from 1 to 1024.

iStatus

[out] char variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 1024.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set string KKK into address 1
char *SetString;
CString m_USAValStr;
m_USAValStr = _T("KKK");
SetString =  (LPSTR)(LPCTSTR)m_USAValStr;
UserSetReg_Str(1,SetString);
```

[VB.NET]

```
Dim Rtn As Byte
Dim UserSetRegStrVal As String

 Rtn = Quicker.UserShare.UserSetReg_Str(1, UserSetRegStrVal.ToCharArray())
```

[VC#.NET]

```
byte Rtn;
string UserSetRegStrVal;
Rtn = Quicker.UserShare.UserSetReg_Str(1, UserSetRegStrVal.ToCharArray());
```

UserGetReg_Str

The function can get a string variable from share memory block.

Syntax

[VC++]

```
unsigned char UserGetReg_Str(unsigned short iUserAddress, char *iStatus)
```

[VB.NET/VC#.NET]

```
byte UserGetReg_Str(ushort iUserAddress, byte[] cGetStr)
```

Parameters

iUserAddress

[in] The address which you want to get from. The range of address is from 1 to 1024.

iStatus

[out] The pointer to a long variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 1024.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get string from modbus address 1
char iStatus[256];
UserGetReg_Str(1,iStatus);
```

[VB.NET]

```
Dim UserGetStr(256) As Byte
Dim Rtn As Byte
Rtn = Quicker.UserShare.UserGetReg_Str(1, UserGetStr)
```

[VC#.NET]

```
byte Rtn;
byte[] UserGetStr = new byte[256];
Rtn = Quicker.UserShare.UserGetReg_Str(1, UserGetStr);
```

UserSetReg_Float

The function can set a float variable into share memory block.

Syntax

[VC++]

```
unsigned char UserSetReg_Float(unsigned short iUserAddress, float *iStatus)
```

[VB.NET/VC#.NET]

```
byte UserSetReg_Float(ushort iUserAddress, out float iStatus)
```

Parameters

iUserAddress

[in] The address which you want to set into. The range of address is from 1 to 19999.

iStatus

[out] float variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set register value 2.5 into address 1
UserSetReg_Float(1,2.5);
```

[VB.NET]

```
Dim Rtn As Byte
Dim UserSetRegFloatVal As Single
Rtn = Quicker.UserShare.UserSetReg_Float(1, UserSetRegFloatVal)
```

[VC#.NET]

```
byte Rtn;
float RegFloat;
Rtn = Quicker.UserShare.UserSetReg_Float(1,out RegFloat);
```

UserGetReg_Float

The function can get a float variable from share memory block.

Syntax

[VC++]

```
unsigned char UserGetReg_Float(unsigned short iUserAddress, float *iStatus)
```

[VB.NET/VC#.NET]

```
byte UserGetReg_Float(ushort iUserAddress, out float iStatus)
```

Parameters

iUserAddress

[in] The address which you want to get from. The range of address is from 1 to 19999.

iStatus

[out] The pointer to a float variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get register value from address 1
float iStatus;
UserGetReg_Float(1,&iSatus);
```

[VB.NET]

```
Dim Rtn As Byte
Dim m_UserGetRegFloatVal As Single
Rtn = Quicker.UserShare.UserGetReg_Float(1, m_UserGetRegFloatVal)
```

[VC#.NET]

```
byte Rtn;
float m_UserGetRegFloatVal;
Rtn = Quicker.UserShare.UserGetReg_Float(1,out m_UserGetRegFloatVal);
```

UserSetReg_Short

The function can set a short variable into share memory block.

Syntax

```
[VC++]
unsigned char UserSetReg_Short(unsigned short iUserAddress, short *iStatus)
```

```
[VB.NET/VC#.NET]
byte UserSetReg_short(ushort iUserAddress, out int iStatus)
```

Parameters

iUserAddress

[in] The address which you want to set into. The range of address is from 1 to 19999.

iStatus

[out] short variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set register value 222 into address 1
UserSetReg_Short(1,222);
```

[VB.NET]

```
Dim Rtn As Byte
Dim UserSetRegShortVal As Integer
Rtn = Quicker.UserShare.UserSetReg_Short(1, UserSetRegShortVal)
```

[VC#.NET]

```
byte Rtn;
int RegShort;
Rtn = Quicker.UserShare.UserSetReg_Short(1,out RegShort);
```

UserGetReg_Short

The function can get a short variable from share memory block.

Syntax

[VC++]

```
unsigned char UserGetReg_Short(unsigned short iUserAddress, short *iStatus)
```

[VB.NET/VC#.NET]

```
byte UserGetReg_Float(ushort iUserAddress, out short iStatus)
```

Parameters

iUserAddress

[in] The address which you want to get from. The range of address is from 1 to 19999.

iStatus

[out] The pointer to a short variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get register value from address 1
short iStatus;
UserGetReg_Short(1,&iSatus);
```

[VB.NET]

```
Dim Rtn As Byte
Dim m_UserGetRegShortVal As Integer
Rtn = Quicker.UserShare.UserGetReg_Short(1, m_UserGetRegShortVal)
```

[VC#.NET]

```
byte Rtn;
short m_UserGetRegShortVal;
Rtn = Quicker.UserShare.UserGetReg_Short(1,out m_UserGetRegShortVal);
```

UserSetReg_Long

The function can set a long variable into share memory block.

Syntax

[VC++]

```
unsigned char UserSetReg_Long(unsigned short iUserAddress, long *iStatus)
```

[VB.NET/VC#.NET]

```
byte UserSetReg_Long(ushort iUserAddress, out long iStatus)
```

Parameters

iUserAddress

[in] The address which you want to set into. The range of address is from 1 to 19999.

iStatus

[out] long variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Set register value 112233 into address 1
UserSetReg_Long(1,112233);
```

[VB.NET]

```
Dim Rtn As Byte
Dim UserSetRegLongVal As Integer
Rtn = Quicker.UserShare.UserSetReg_Long(1, UserSetRegLongVal)
```

[VC#.NET]

```
byte Rtn;
int RegLong;
Rtn = Quicker.UserShare.UserSetReg_Long(1,out RegLong);
```

UserGetReg_Long

The function can get a long variable from share memory block.

Syntax

[VC++]

```
unsigned char UserGetReg_Long(unsigned short iUserAddress, long *iStatus)
```

[VB.NET/VC#.NET]

```
byte UserGetReg_Long(ushort iUserAddress, out long iStatus)
```

Parameters

iUserAddress

[in] The address which you want to get from. The range of address is from 1 to 19999.

iStatus

[out] The pointer to a long variable.

Return Values

0 indicates success. **WCA_USERADDR_OVER** means the iUserAddress over the range. The legal range is from number 1 to number 19999.

Remarks

Requirements

Runs on	Versions	Defined in	Include	Link to
XP-8000-CE6	1.3.1.0 and later	Quicker.lib	WinConAgent.h	

Example

[VC++]

```
//Get register value from address 1
long iStatus;
UserGetReg_Long(1,&iSatus);
```

[VB.NET]

```
Dim Rtn As Byte
Dim m_UserGetRegLongVal As Integer
Rtn = Quicker.UserShare.UserGetReg_Long(1, m_UserGetRegLongVal)
```

[VC#.NET]

```
byte Rtn;
int m_UserGetRegLongVal;
Rtn = Quicker.UserShare.UserGetReg_Long(1,out m_UserGetRegLongVal);
```

4.4.2 Quicker API for VB.NET/VC#.NET Developer

Step 1:

Create a smart device project

Step 2:

[Add Reference] ->QuickerNet.dll

Step 3:

Refer to the function prototype of QuickerNet.dll by Object Browser

Step 4:

Call the functions in the QuickerNet.dll (Please refer to the Quicker_VB.NET_Demo /Quicker_VC#.NET_Demo)

Step 5:

Build your project and copy it and relative library into XP-8000-CE6

Note: Quicker.dll, QuickerNet.dll, and VB.NET/VC#.NET application program must be copied to the same folder in the XP-8000-CE6

4.5 NAPOPC_CE6 with Rule Script

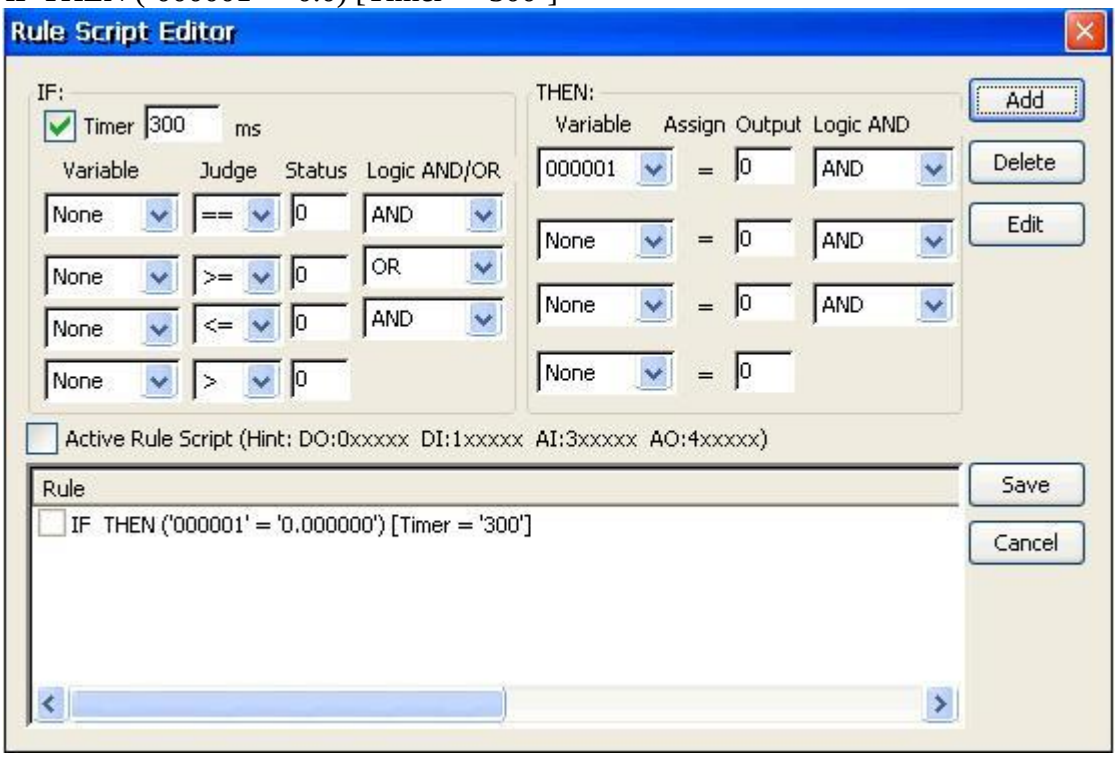
NAPOPC_CE6 provides "Rule Script Editor" to user for editing the rules. This function is based on the instinctive design style to develop rule list. The program designers can easily implement their logic via "IF…THEN…" syntax into rule list to achieve the purpose of chain reaction control. The "Rule Script" is suitable within the non-critical situation. Using this function can not only avoid typing error but also save developing time.

4.5.1 Rule Script Syntax

Rule script syntax is very instinctive as well. In the "IF" area, the relation between timer and other variables is "AND". The triggered frequency of the rule is decided by the timer of each rule. If the rule has timer and the "THEN" area has "0xxxxx" variable, the "0xxxxx" variable will frequently "ON/OFF" switch like blinking function.

Ex 1:

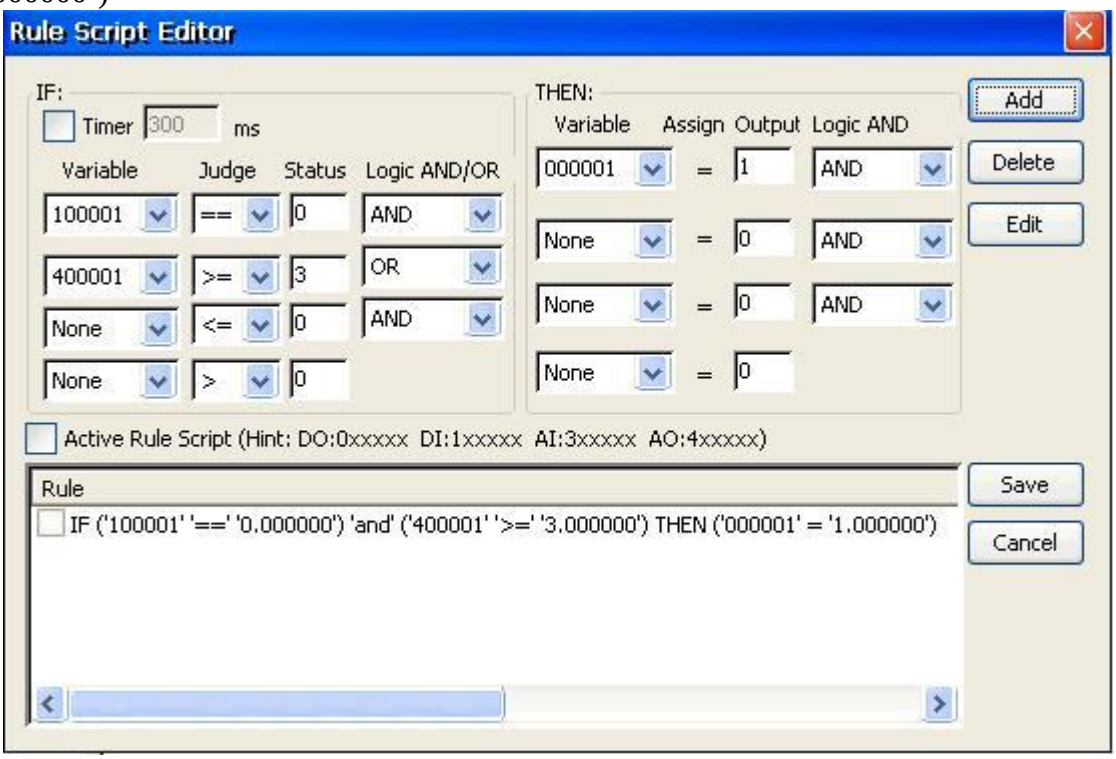
IF THEN ('000001' = 0.0) [Timer = '300']



Which means the variable "000001" will do "ON/OFF" switch every 300ms.

Ex 2:

IF ('100001' '==' '0.000000') and ('400001' '==' '3.000000') THEN ('000001' '=' '1.000000')



Which means the variable "000001" will do "ON" when variable "100001" is "0" and variable "400001" is "3". For more advanced application, user can use the variable in the "Internal device" to be a temporary buffer to chain each rule.

Appendix A – Error list and description

Code Description I/O Unit Min Max		
Code	**Define**	**Description**
0	WCA_OK	OK
102	WCA_Stop	ScanKernel has been stopped
103	WCA_SLOTNO_OVER	Slot number must be 1 - 8
104	WCA_ATT_ERROR	Attribute number error. It should be 1 or 0
105	WCA_COMNO_OVER	COM port No. must be 2 or 3
106	WCA_SLAVENO_OVER	Slave number must be 1 - 256
107	WCA_NOT_MASTER	Not the main AP which calls ScanKernel
108	WCA_MBADDR_OVER	Modbus DIO address must be 449 – 2048, AIO address must be 225 - 2048
109	WCA_MBATTR_ERROR	Modbus attribute must be 1 or 0
110	WCA_USERADDR_OVER	User defined address must be 1 - 8192
111	WCA_USERRATTR_ERROR	User defined register value must be -32768 to 32767