CONTROL TECHNOLOGY CORPORATION

M3-61A DeviceNet Master Module

# M3-61A DeviceNet™ Master Module

*Blank*

**WARNING:** Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.



The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions. See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

DeviceNet™ is a trademark of Open DeviceNet Vendor Association, Inc. (ODVA)

| Model Number | Hardware Revision | Firmware Revision |
|---|---|---|
| M3-61A | All Revisions | >= M361AV0106<br>>= BF5300V059069R30 |

# TABLE OF CONTENTS

*Break*

CHAPTER 1

# [1] Overview



The 5300 series programmable automation controllers can be simultaneously connected to one or more fieldbus networks. Modbus master and slave communications are built into the CPU module. Modbus master and slave communications are supported on both the serial COM ports as well as the Ethernet ports. Additional fieldbus networks are supported via Model 5300 Fieldbus Modules that plug into the 5300 backplane. CTC currently offers Model 5300 modules for the following fieldbus networks:

| | |
|---|---|
| DeviceNet Master | M3-61A |
| DeviceNet Slave | M3-61B |
| EtherNet/IP Master | M3-61C |
| EtherNet/IP Slave | M3-61D |

Additional fieldbus modules are under development for popular fieldbus networks such as Profibus, CANOpen and others. To check on the release status of modules other than those listed above, contact CTC sales.

## *Model 5300 Fieldbus Module Architecture*

The CTC fieldbus modules contain two circuit cards. The first card is the universal fieldbus adapter, which handles all interfacing tasks between the Model 5300 controller and the second card, which is called the fieldbus interface adapter. The fieldbus interface adapter is developed by HMS. In adopting this architecture CTC teamed up with HMS (http://www.hms.se/ ) who is the industry leader in industrial networking cards. This allows CTC to provide a wide range of network interfaces. Additionally CTC benefits from HMS's large engineering staff which is focused on updating the fieldbus interfaces and making sure they are in compliance with the applicable ratings agencies.

## *M3-61A DeviceNet Master*

The M3-61A module provides DeviceNet Master support for the 5300 series controller. This includes bit oriented I/O and explicit messaging with support for mapping data such

as analog and explicit messages to/from the Model 5300 registers, as well as multiple modules/networks.

## Front Panel


LED – NS, Network Status, flashing red indicates bad config.
LED – MS, Module Status
LED – RS, Run Status, not used but set to green.

| LED | LED Status | Description |
|---|---|---|
| **Module status** | Off | No power or not initialized |
| | Green | Module status is OK |
| | Flashing red | Minor fault |
| | Red | Major fault |

Switch – 1, 2 baud rate (on = 1)

Switch – 3 to 8 MACID, 0 to 63 binary with switch 8 low bit.

| Baudrate (kBit/sec) | DIP 1-2 |
|---|---|
| 125 | 0 0 |
| 250 | 0 1 |
| 500 | 1 0 |
| Reserved | 1 1 |

USB & COM are used for re-flash of firmware and future optional RS232 serial port.

LED 1-4 are reserved for future use.

CH – CAN High (terminating 120 ohm resistors required)
SD – Shield/Drain
CL – CAN Low

CHAPTER 2

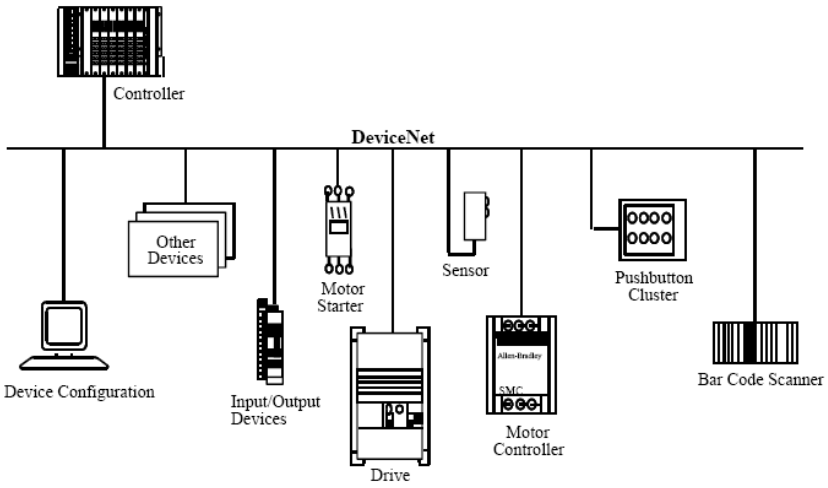
# [2] DeviceNet

DeviceNet is a fieldbus system used for industrial automation, normally for the control of valves, sensors and I/O units and other automation equipment. The DeviceNet communication link is based on a broadcast oriented, communications protocol, the Controller Area Network (CAN). This protocol has I/O response and high reliability even for demanding applications.

DeviceNet has a user organization, the Open DeviceNet Vendor Association (ODVA), which assists members on matters concerning DeviceNet. HMS is a member of ODVA and also represented as a member of the DeviceNet Conformance SIG.

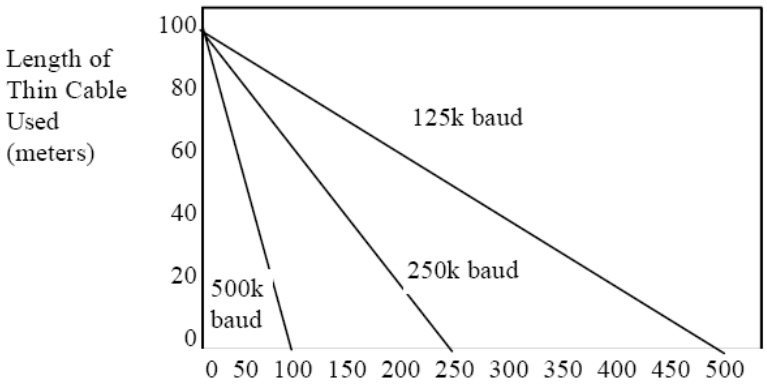
## *Network Overview*

The physical media for the fieldbus is a shielded copper cable composed of one twisted pair and two cables for the external power supply. The baud rate can be changed between 125k, 250k and 500k bit/sec. Each node in the network is given a MAC ID, which is a number between 0 and 63 and is used to address the node.



## *Technical Features of DeviceNet*

The maximum length of cable is dependent on the baud rate and DeviceNet cable that are used. Below is a diagram that shows the maximum allowed cable length in the network.




$L_{thick} + 5 \times L_{thin} = 500$ at 125Kbaud
$L_{thick} + 2.5 \times L_{thin} = 250$ at 250Kbaud
$L_{thick} + L_{thin} = 100$ at 500Kbaud

where $L_{thick}$ is the length of thick cable and Lthin is the length of thin cable.

### Summary: The Technical Features of DeviceNet

- DeviceNet specific cable (twisted pair)
- Access to intelligence present in low-level devices
- Master/Slave and Peer-to-Peer capabilities
- Trunkline-dropline configuration
- Support for up to 64 nodes
- Node removal without severing the network
- Simultaneous support for both network powered (sensors) and self powered (actuators) devices
- Use of sealed or open style connectors
- Protection from wiring errors
- Selectable data rates of 125k baud, 250k Baud, and 500k baud max. Trunk distance 500 meters and drop length 156 meters at 125k baud.
- Adjustable power configuration to meet individual application needs
- High current capability (up to 16 amps per supply)
- Operation with off-the-shelf power supplies
- Power taps that allow the connection of several power supplies from multiple vendors that comply with DeviceNet standards
- Built-in overload protection
- Power available along the bus; both signal and power lines contained in the trunkline
- Provisions for the typical request/response oriented network communications
- Provisions for the efficient movement of I/O data
- Fragmentation for moving larger bodies of information
- Duplicate MAC ID detection

## HMS AnyBus-M

The M3-61A uses the HMS AnyBus-M DeviceNet interface module to ensure full compliance. As such, the module will appear on the network with the following parameters:

| Description | Text string | Dec | Hex |
|---|---|---|---|
| Vendor ID | HMS Fieldbus Systems AB | 90 | 0x5A |
| Product type | Communications adapter | 12 | 0x0C |
| Product code | - | 14 | 0x0E |
| Product name | AnyBus-M DeviceNet | - | - |

The ANYBUS® M DeviceNet follows the DeviceNet standard that has been developed by ODVA. It is fully compatible with the DeviceNet specification rev. 2.0 Vol I and Vol II. The module operates according to the communication adapter profile (product type 12, see DeviceNet specification for more information). The module supports the I/O connections Bit strobe, Polled I/O, Change of state and Cyclic I/O data.

## DeviceNet Features

| | | | |
|---|---|---|---|
| Device Type: | Communication adapter | Master/Scanner: | Yes |
| Explicit peer-to-peer messaging: | Yes | I/O slave messaging: | |
| I/O peer-to-peer messaging: | No | Bit strobe | Yes |
| Configuration consistency value | Yes | Polling | Yes |
| Faulted node recovery: | No | Cyclic | Yes |
| Baud rates: | 125K, 250K, 500K | Change of state (COS) | Yes |

*Blank*

CHAPTER 3

# [3] Interface Basics



The M3-61A must be configured prior to operation. This consists of setting the network speed and Master MACID via dip switches as well as setting up the proper network configuration using a PC and device specific EDS (Electronic Data Sheet) files. The PC is then attached to the network and communicates with the M3-61A over CAN, setting up the proper scan list of devices. A configurator is used to define how the data received from the remote device is transferred to the M3-61A and how it is mapped into memory, and hence assigned as I/O registers. In this chapter we will review the basic architecture of the M3-61A as it operates within the Model 5300 system. Then in Chapter 6 we will cover an actual example.

## *Basic Architecture*

The M3-61A DeviceNet Master operates asynchronously to the main Model 5300 controller, constantly scanning and updating information as needed. In addition to its HMS AnyBus-M module there is a 60 MHZ ARM7 processor operating as an interface and high level controller. This processor handles the interface and mapping between the AnyBus-M DeviceNet data and that observed by the controller, as well as all the explicit message queuing, responses, error retries, and what to do with bit oriented data as it is received. Some examples are analog read/writes, mapping explicit message data contents and responses, etc. For now we will discuss the simplified I/O scanned data mapping.

There is a dual port memory device that exists between the Anybus-M and the ARM7 processor. Where data is mapped from a DeviceNet Slave determines its I/O reference from a Model 5300 application program perspective. A Model 5300 controller can access a total of 1024 inputs and 1024 outputs overall in a system. Convert that to bytes and you have 128 bytes of input information and 128 bytes of output information that you can scan with a Master controller. Any local Model 5300 I/O also counts towards this maximum.

If you reference a DeviceNet Slave device you will note that it can produce a certain number of bytes and consume a certain number (defined in its EDS file). To keep things simple, each byte produced will add 8 inputs to the Model 5300 and each byte consumed

will add 8 outputs. This directly maps to the input/output registers in the QuickStep Register Guide:

| | |
|---|---|
| 1001-1999 | **Digital Output Bits:** Alternate Access to Outputs 1-999: R/W, 0=off, 1=on. |
| 2001-3024 | **Digital Input Bits:** Alternate Access to Inputs 1-1024: R only, 0=open, 1=closed. |
| 3025 | **Digital Output Force Selection:** R/W, select Digital output to effect with Set/Clr, 1 based. (M3-20/21 only) |
| 3026 | **Digital Output Force Set:** R/W, set digital output (on), overriding application (M3-20/21 only) |
| 3027 | **Digital Output Force Clr:** R/W, clear digital output (off), overriding application (M3-20/21 only) |
| 3028 | **Digital Input Force Selection:** R/W, select Digital input to effect with Set/Clr, 1 based. (M3-20/21 only) |
| 3029 | **Digital Input Force Set:** R/W, set digital input simulation (M3-20/21 only) |
| 3030 | **Digital Input Force Clr:** R/W, clear digital input simulation (M3-20/21 only) |
| 10001-10032 | **Digital Output Integer:** Access Outputs as a 32-bit number: R/W |
| 10101-10164 | **Digital Output Short:** Access Outputs as a 16-bit number: R/W |
| 10201-10328 | **Digital Output Byte:** Access Outputs as an 8-bit number: R/W |
| 11001-11032 | **Digital Input Integer:** Access Inputs as a 32-bit number: R only |
| 11101-11164 | **Digital Input Short:** Access Inputs as a 16-bit number: R only |
| 11201-11328 | **Digital Input Byte:** Access inputs as an 8-bit number: R only |

*The Force registers are available but have not been tested in the current firmware revision.*

Thus let's reference a simple device such as the Automation Direct DL105. This device has 10 bits of produced data and 8 bits of consumed, simply 10 inputs and 8 outputs.

http://www.facts-eng.com/manuals/f1dvnetm.pdf

Upon referencing their provided EDS file it would be observed that the RX bytes on a poll are 2 (16 bits read but only the first 10 really produced data) and the TX bytes are 1 (8 bits consumed). From a Model 5300 perspective if no local digital input or output modules were installed then register 1001 would be the first output and 2001 the first input on the DL105. If Model 5300 I/O modules are installed then the DeviceNet I/O adds on after the last local I/O. For example a single M3-10 type module has 16 in, 16 out, causing first DL105 output to now be register 1017. As other modules are added note that the 2 bytes received for input data still occupy 16 bits even though 10 bits is all that is usable, the other 6 bits will be a constant 0. Same for unused outputs, should a device not have an evenly divisible number of outputs then you will still consume the output space on the Model 5300, and those bits will not operate (example: device provides 5 outputs and 1 byte consumed, hence outputs 6, 7, and 8 are do nothing outputs, next DeviceNet Slave outputs would begin after the last output, 8). Reference the next section for actual mapping examples.

CHAPTER 4

# [4] DeviceNet Network Setup Overview



This chapter provides a high level overview of the steps necessary to set up a DeviceNet network to properly communicate with the Model 5300 automation controller. The following chapters will cover these steps in detail.

## *Installation*

**Hardware Installation:**

1. M3-61A: Set the baud rate to match the network and set a unique MAC ID. Install the module into the Model 5300 rack. It must be installed in a slot after any M3-40 series modules, thus keep M3-40 modules to the left of the M3-61A module.
2. DeviceNet Devices: Set the baud rate to match the network and set a unique ID.
3. Network cabling: Connect up to 63 devices to the M3-61A following proper DeviceNet network wiring standards. Note that there is no internal network terminator on the M3-61A module, so proper network termination is required.

After powering up the Model 5300 controller you may access the remote administrative screens via telnet and verify module installation. Below shows two M3-61A modules installed in slots 4 and 6:

```
BlueFusion/>get versions

*Local 5300 Serial Number = 10086477
  DNS Name: CTC_BF_10086477  DHCP active: YES
  Group Name:
  IP Address = 12.40.53.40 MAC Address = 00C0CB99E84D
  Total:  DIN = 56  DOUT = 48  AIN = 2  AOUT = 2  MOTION = 0
  Base Firmware Revisions:
    Quickstep ARM9 Application                  V05.00.90R45
    Quickstep ARM9 Monitor                      V01.49 A
    CT webHMI Enabled
  Available Racks:
    1. 8 slots local.
    2. Not installed.
    3. Not installed.
    4. Not installed.
  Slot Firmware Revisions:
    R1.01(01). M3-20C - 5VDC SNK 16DI 16DO 8CNT       (00000000) V01.05
    R1.02(02). M3-20C - 5VDC SNK 16DI 16DO 8CNT       (00000000) V01.05
    R1.03(03). Empty                                  (00000000) V00.00
    R1.04(04). M3-61A - DeviceNet Master              (00000000) V01.02
    R1.05(05). Empty                                  (00000000) V00.00
    R1.06(06). M3-61A - DeviceNet Master              (00000000) V01.02
    R1.07(07). Empty                                  (00000000) V00.00
    R1.08(08). Empty                                  (00000000) V00.00
```

*A configuration file must be loaded for proper operation, reference Chapter 7: XML Configuration File & I/O Declarations.*

## DeviceNet Network Configuration

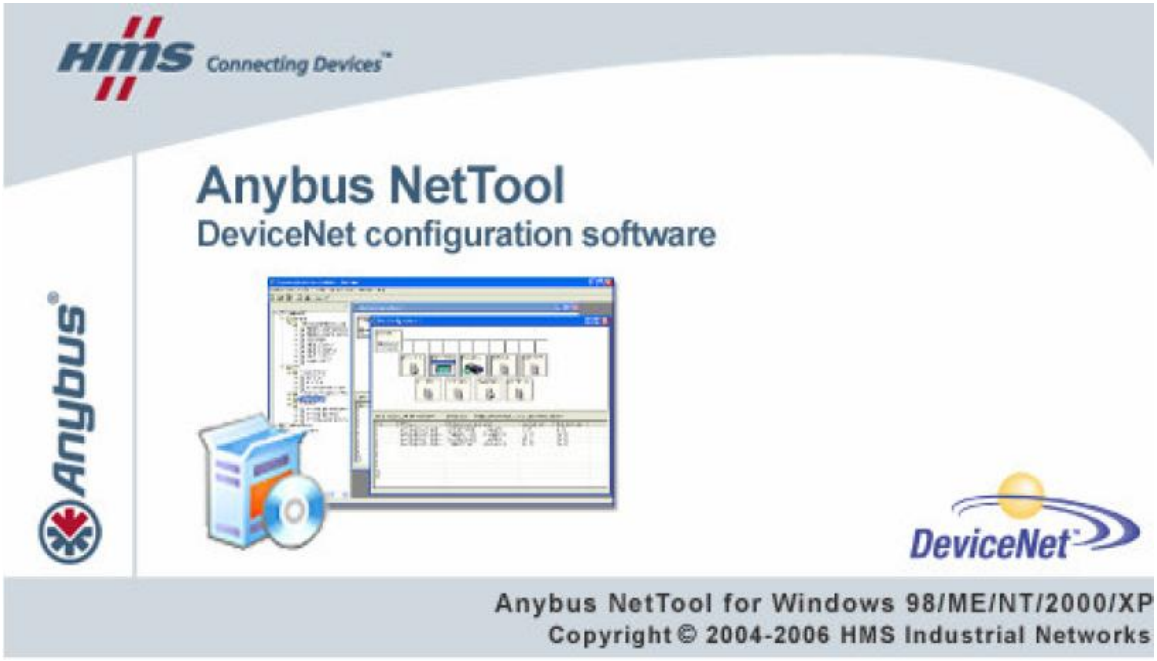
Before device information can be accessed, each node must be properly mapped to the M3-61A DeviceNet master module.

There are currently two ways to configure the M3-61A, both using independent PC based applications. A third method, integrated within Quickbuilder, will be available at a later date. The first way to configure the M3-61A module is to use Rockwell's RSNetWorx for DeviceNet. This program is run on a PC in the Windows environment. The program needs a physical link to the DeviceNet network where the M3-61A module is connected and also the DeviceNet Slaves that the module will communicate with are connected. This physical link can be a serial adapter (for example 1770-KFD), a PCI or ISA card (for example 1784 scanner), or a PCMCIA interface. Reference the web link below for their documentation:

http://literature.rockwellautomation.com/idc/groups/literature/documents/gr/dnet-gr001_-en-e.pdf

The preferred method of configuration, and the one currently used by CTC, is provided by HMS. This configurator is known as NetTools (NetTool-DN-D, CTC Part#067-018020) and comes with a serial to DeviceNet network converter.

http://www.anybus.com/products/products.asp?PID=98&ProductType=Anybus%20NetTool




This manual assumes you know how to use this tool. Some examples are provided but it is important to first reference the documentation available for this software product prior to attempting configuration:

http://www.anybus.com/upload/98-7256-NetTool%20DN%20User%20Manual%201_0.pdf

A good application example is also provided:

http://www.anybus.com/upload/98-0005-How%20to%20configure%20DeviceNet%20with%20NetTool%20for%20DeviceNet_1.10.pdf

The NetTool Configurator consists of a DeviceNet network adapter and a PC software application. This adapter makes it possible to connect a serial port from a PC to the DeviceNet network and configure and monitor DeviceNet nodes remote from any point on the DeviceNet network. Using this tool, the following steps are taken to configure a device:

1. Identify a network and the attached devices
2. Configure the I/O data from each device
3. Map the I/O data to the M3-61A
4. Configure the M3-61A master module

An example showing how two devices are configured on the network is given in *Chapter 6: NetTools Example*. Installation instructions, over and above that provided by HMS, are provided in *Chapter 5: NetTools Installation*.

*When using the serial to NetTools DeviceNet converter it has been observed that after a power cycle or initial boot you must attempt to connect to the converter a few times if using a USB to serial converter cable. It will show a timeout error on the first couple of attempts.*

*When downloading a scan list to the M3-61A, make sure you first place it in idle mode, then do the scan list download, return to either set mode to run and/or cycle power. It has been observed that some DeviceNet Clients need power cycling of either the master and/or slave after configuration changes.*

## Configuration of M3-61A Within the 5300 Controller

After all of the network device characteristics have been set up on the M3-61A, the Model 5300 must be made aware of these devices and how it should map/interface with the device. This is done via a special ASCII configuration file called the XML Configuration File. Each M3-61A in the Model 5300 rack requires its own file. Creation of this file is covered in a later chapter, but the steps involved are outlined below:

1. Using a text editor or XML editor, load a blank template provided by CTC
2. Edit the Digital and Analog I/O definitions. This maps the DeviceNet I/O into the Model 5300 controller. Note that DeviceNet I/O numbering must start after the local I/O.
3. Optionally: Edit Explicit Message definitions.
4. Optionally: Set up high speed dual port registers – provides very high speed access to I/O or parameters mapped to these registers between the Model 5300 CPU and the M3-61A. There are 256 of these registers per M3-61A.
5. Download XML Configuration File to the M3-61A (as explained in the *XML Configuration File Storage* section found at the end of *Chapter 7: XML Configuration File & I/O Declarations*).

CHAPTER 5

# [5] NetTools Installation

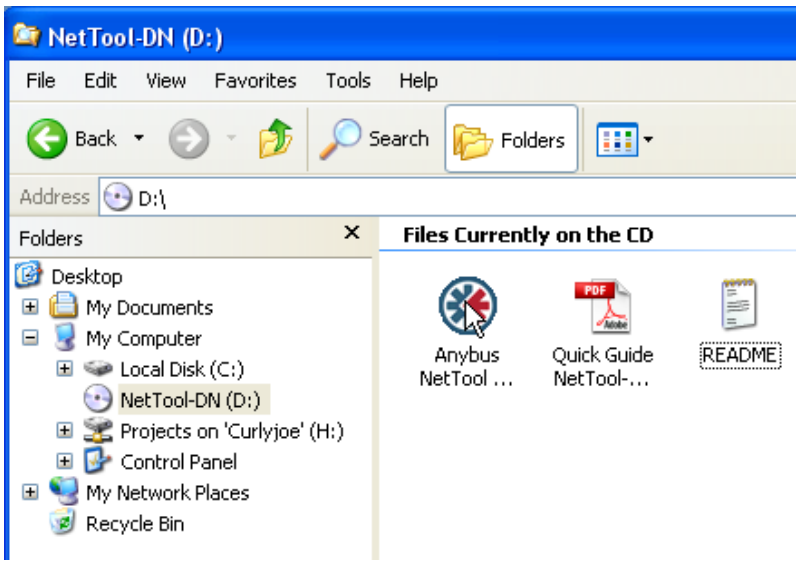


This section includes installation instructions for the HMS Anybus-NetTool-DN-D toolset. It should be used in addition to the instructions provided by HMS and/or as a reference should questions arise. Screen captures of each installation step are provided to simplify the process. Additionally the importing of an EDS file is covered as well as initially going online to a DeviceNet network.

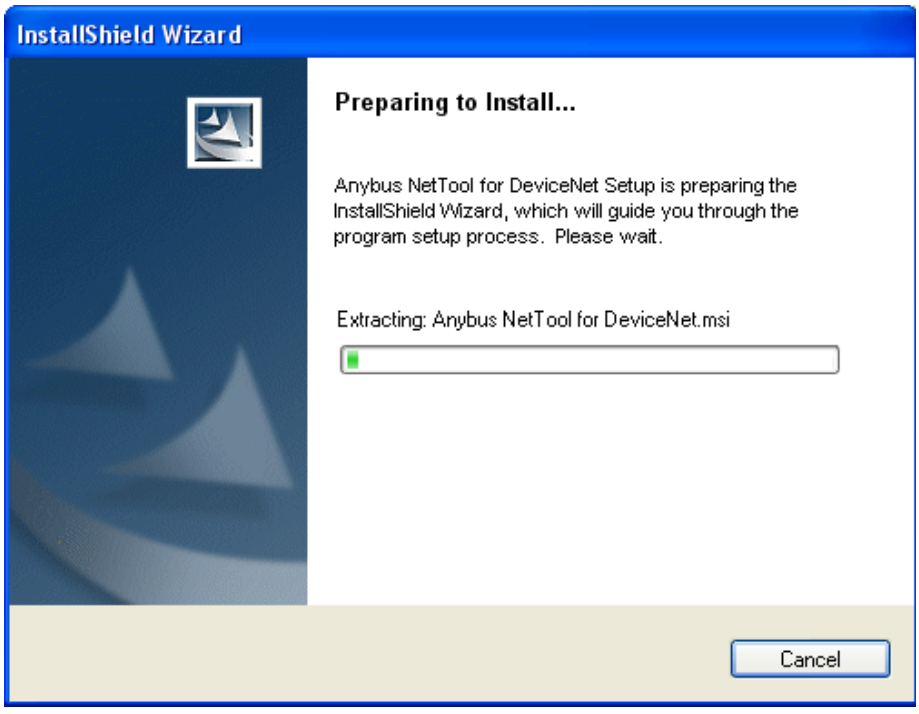
The Anybus-NetTool-DN-D is designed for use with Anybus-M Embedded cards or Anybus X-gateways with DeviceNet Scanner. Functions include full scanner and adapter configuration, on/off line configuration, online diagnostics, auto EDS file generator and online parameter editor. Anybus-NetTool-DN-D includes an RS232<->DeviceNet adapter. This adapter makes it possible to connect a serial port from a PC to the DeviceNet network and configure and monitor DeviceNet nodes remote from any point on the DeviceNet network.

## *Installation*

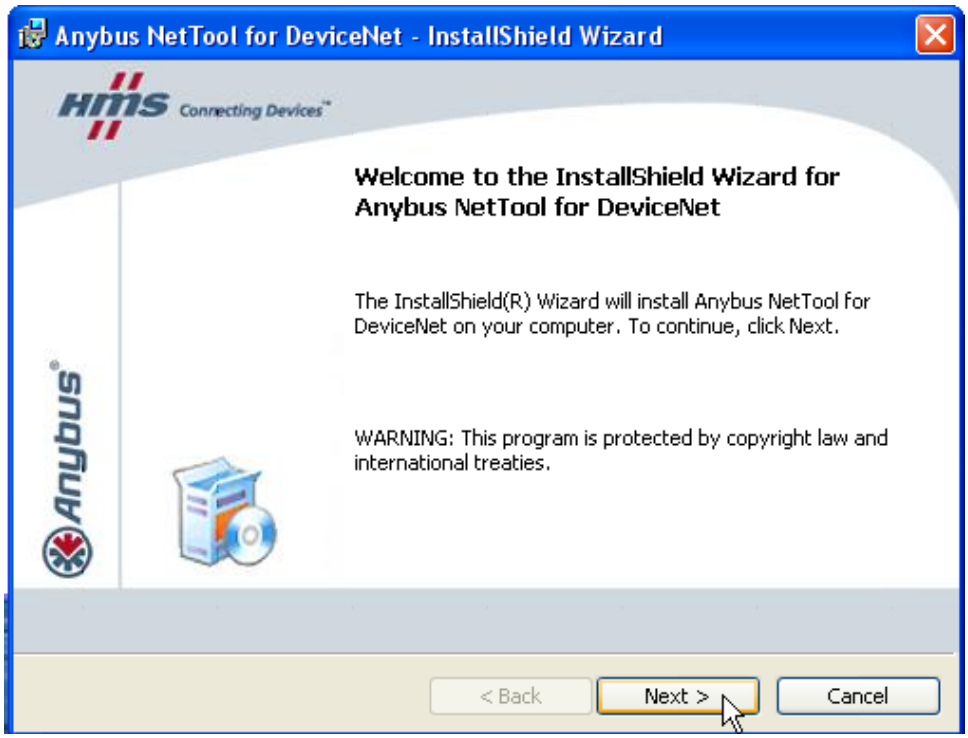
Insert the CD into the PC drive. Explore the files on the drive and double click the Anybus NetTools Setup Icon:

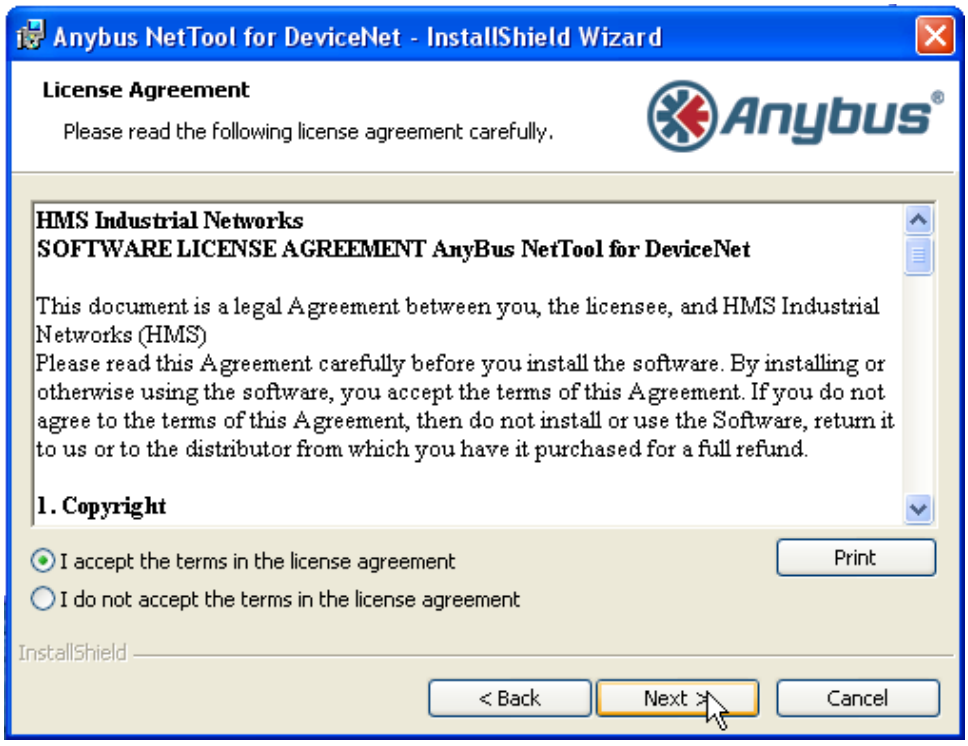



The package will begin to prepare for installation:

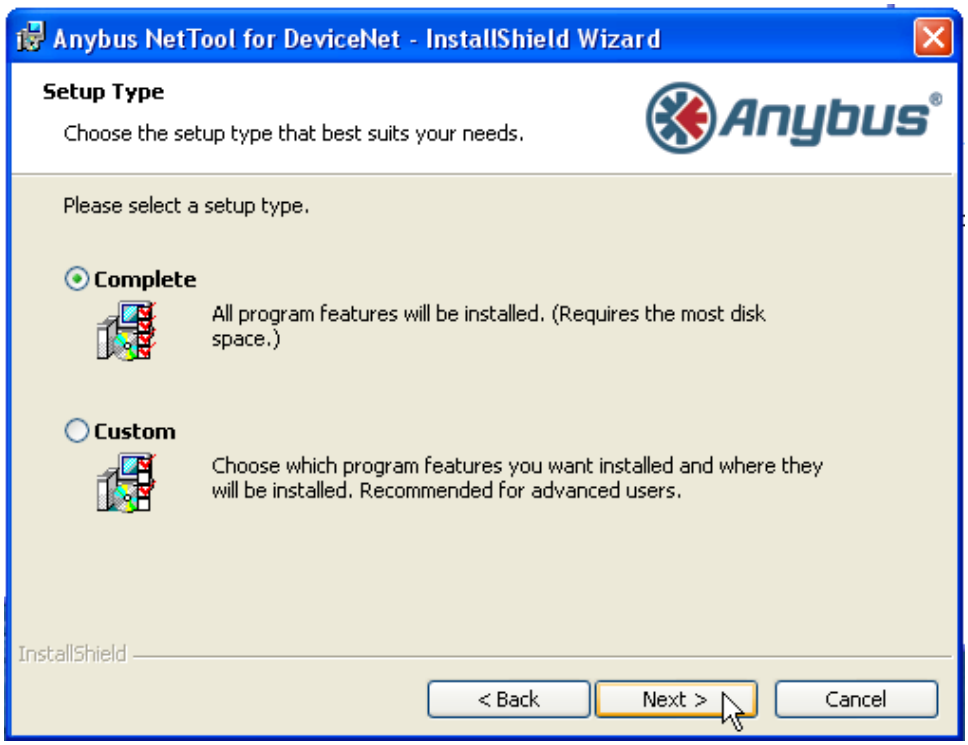


Click Next on the Welcome screen, accept the license and click Next once again:

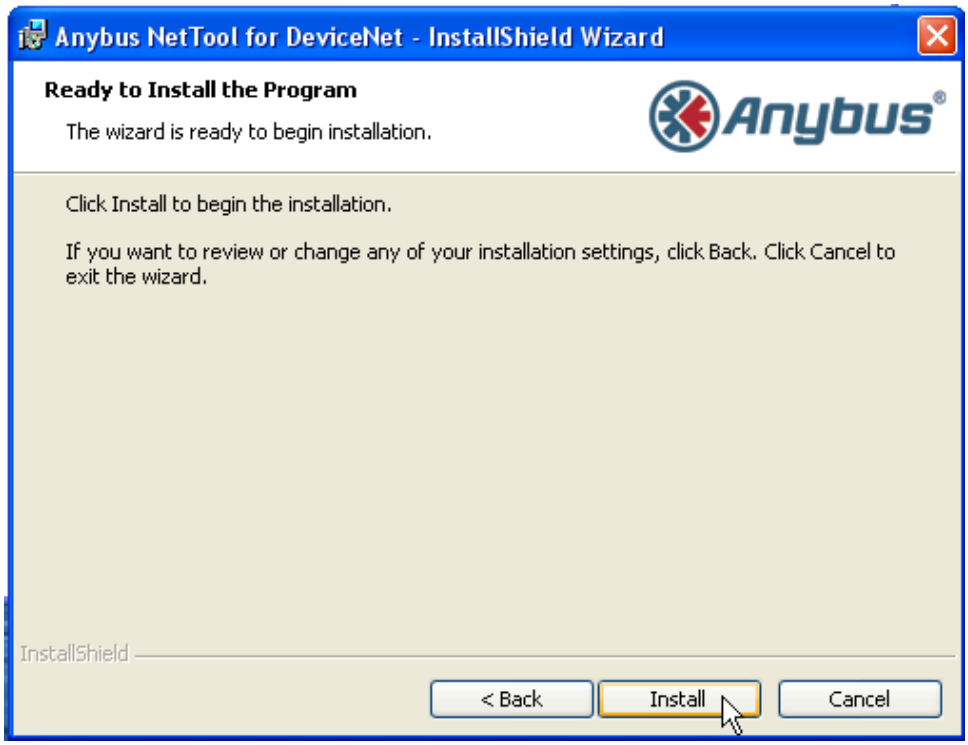


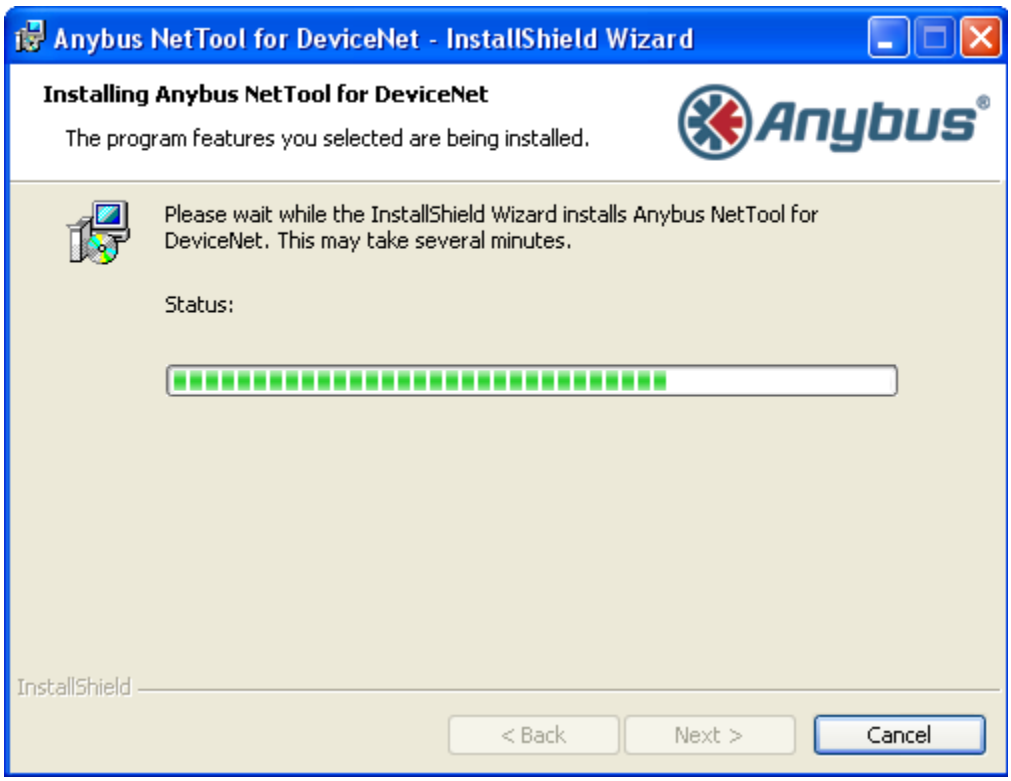
Click Next on the Complete installation:



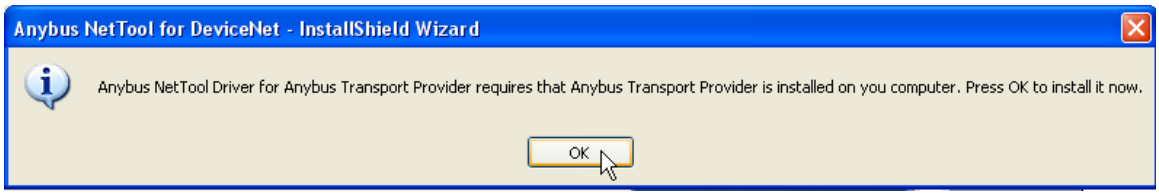

Click Next on Ready to Install the Program:

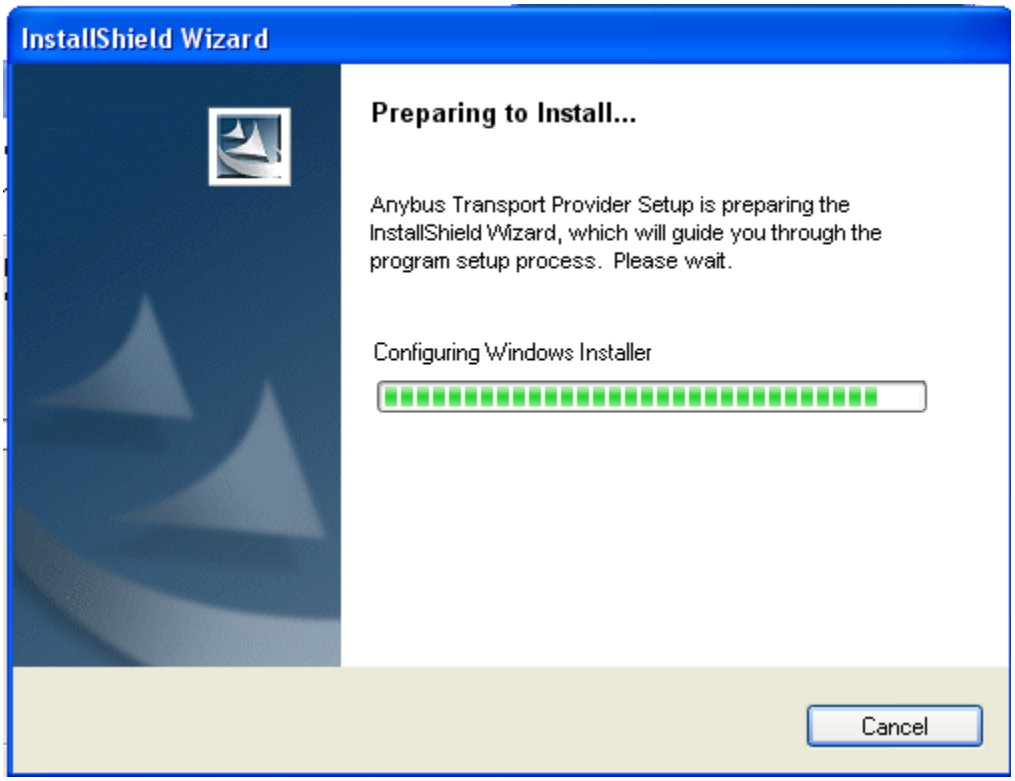


The program will begin installing:

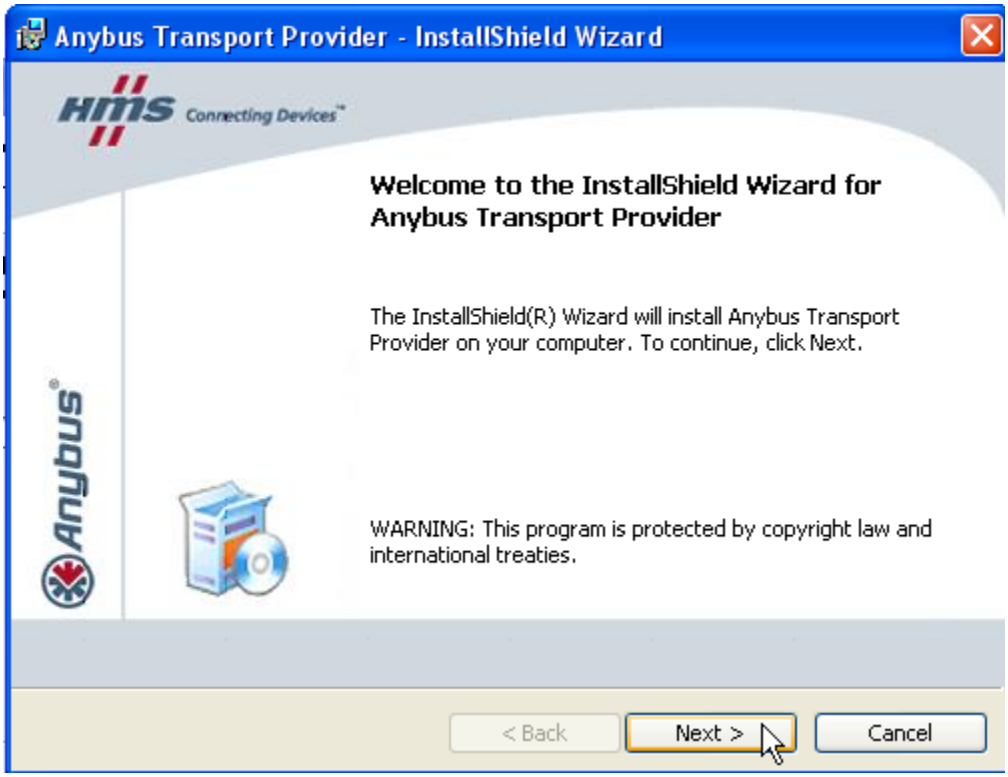



Click OK to begin installing the Anybus Transport drivers:

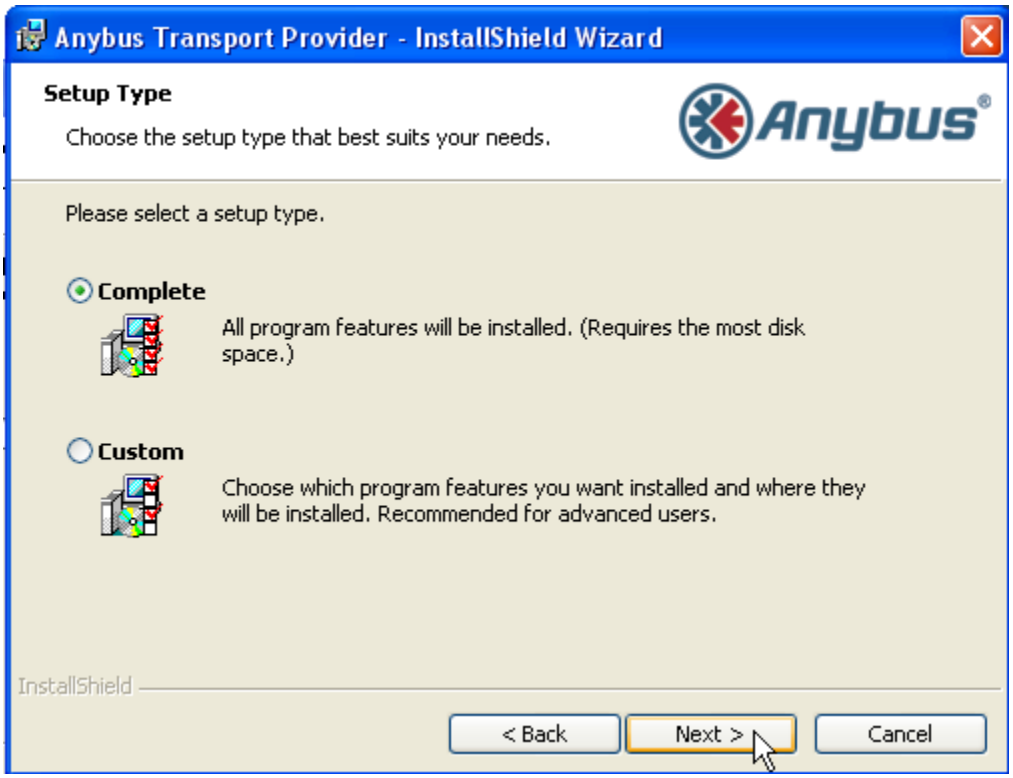


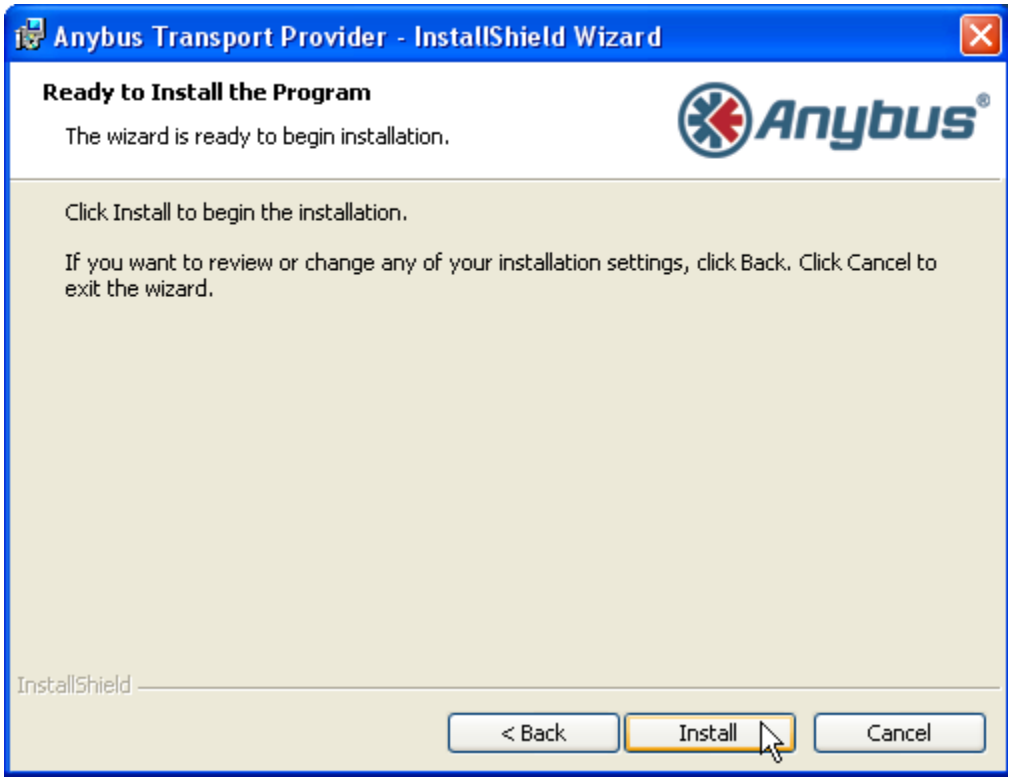



Click Next on the Transport Provider Welcome screen:

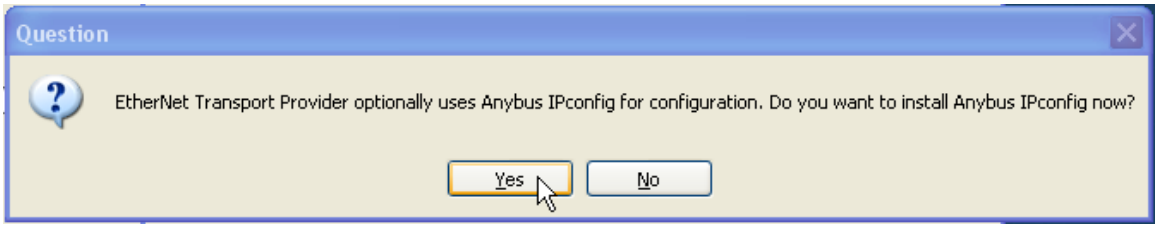


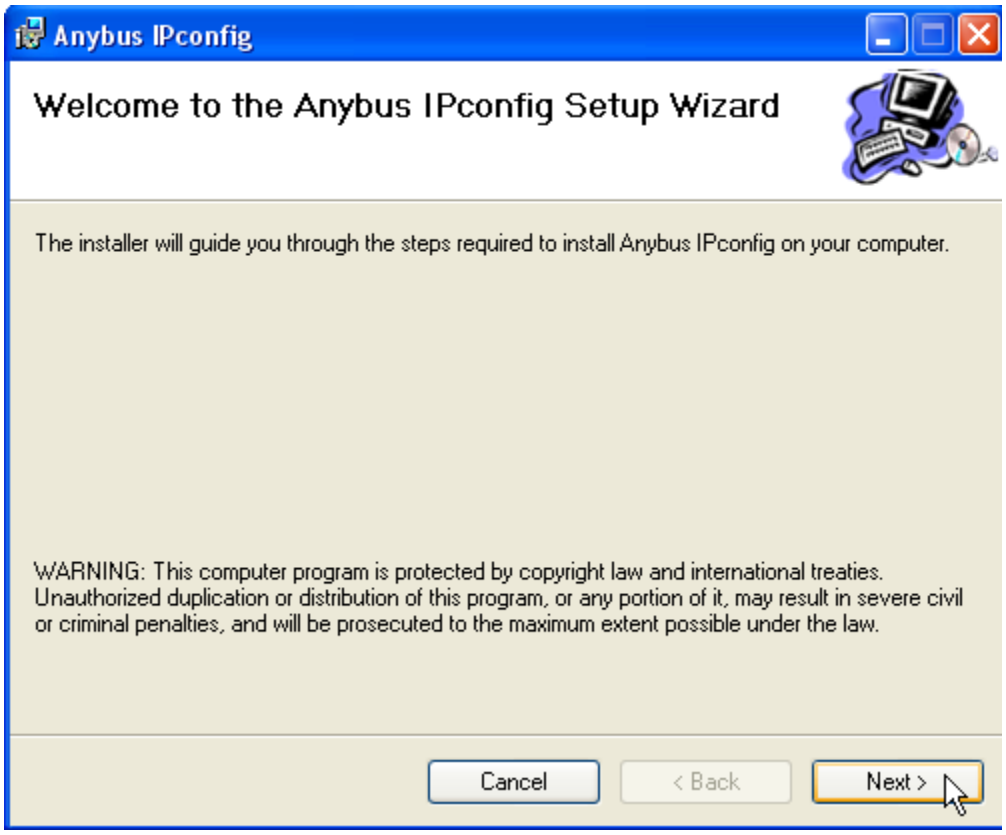
Click Next with Complete selected and Install on the Ready to Install the Program screen:



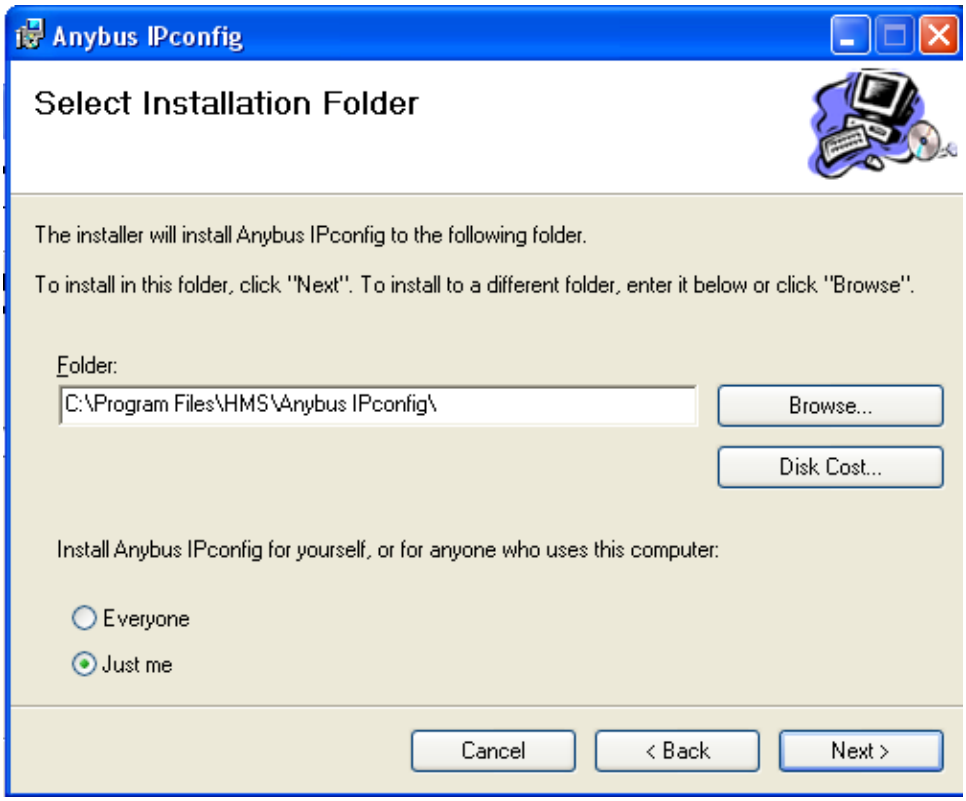



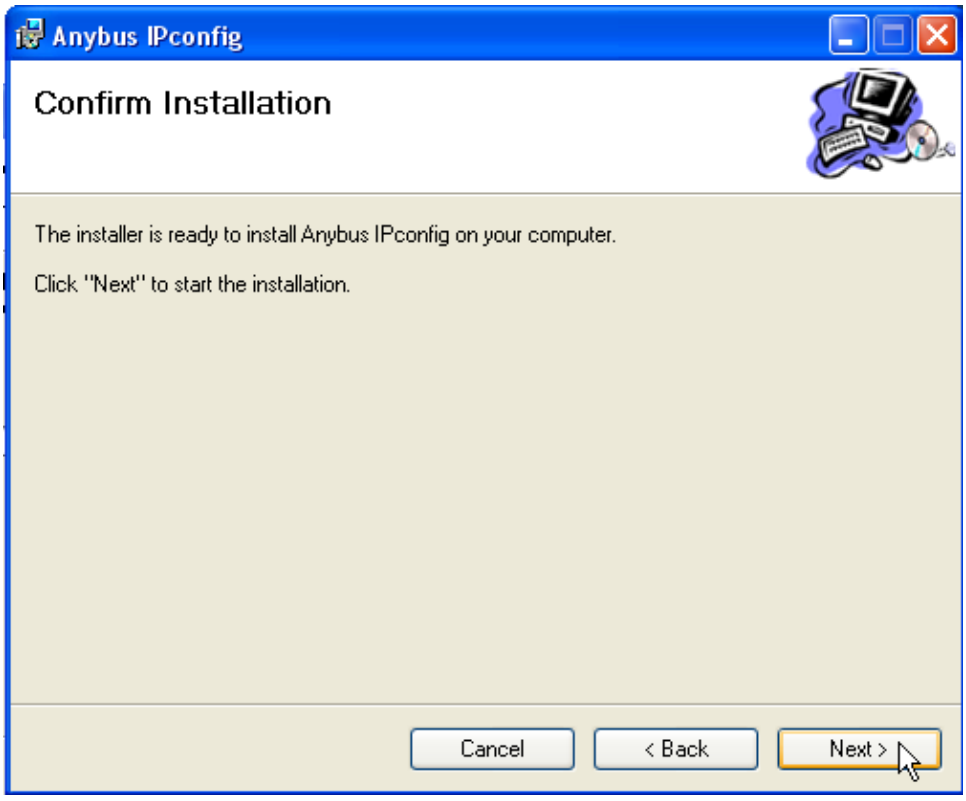
Click Yes for EtherNet Transport even though not using, followed by Next on the Welcome screen:



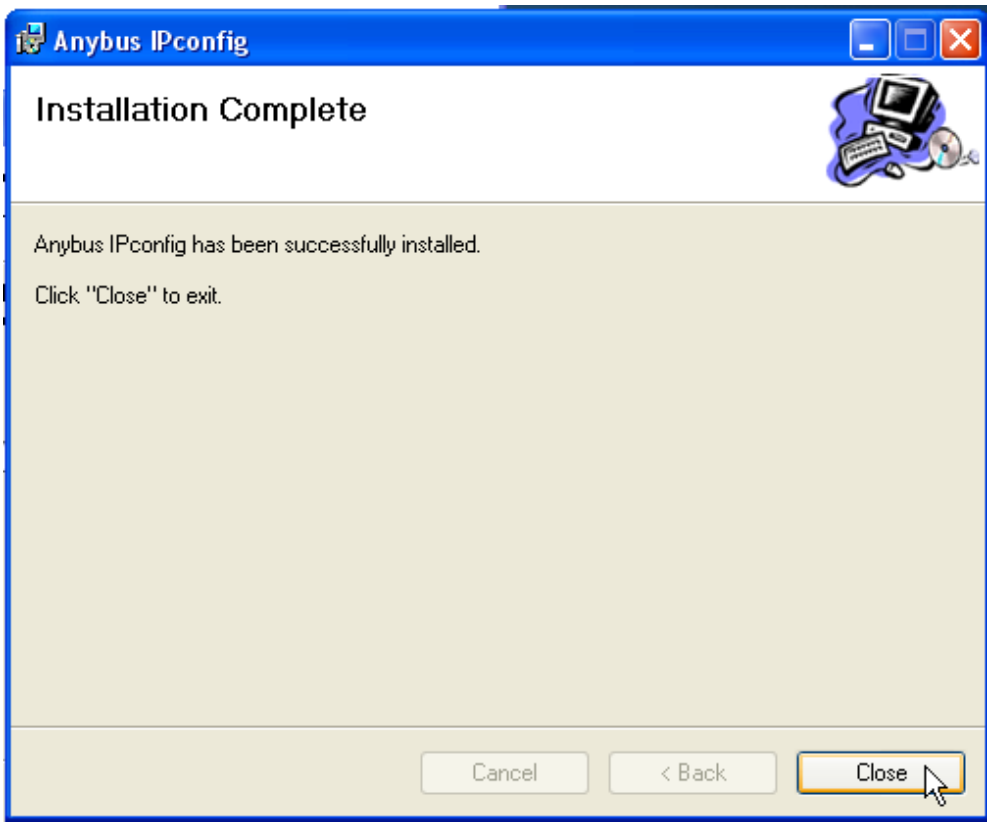



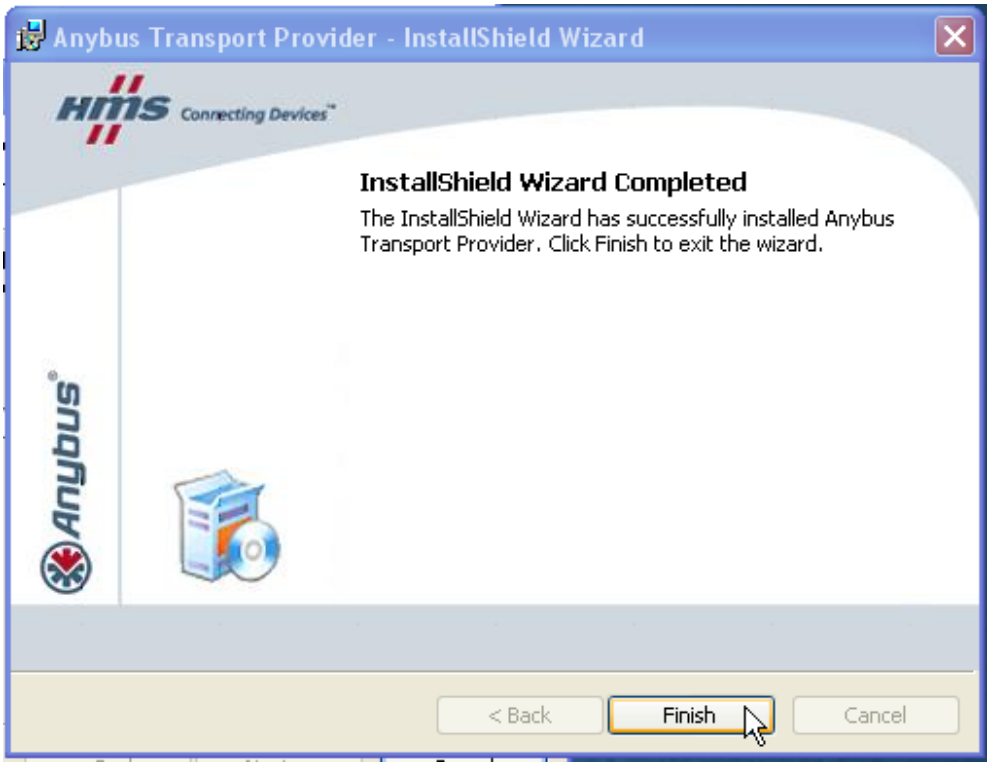
Select the folder for installation and who will use this on your computer, followed by Next, confirm installation:

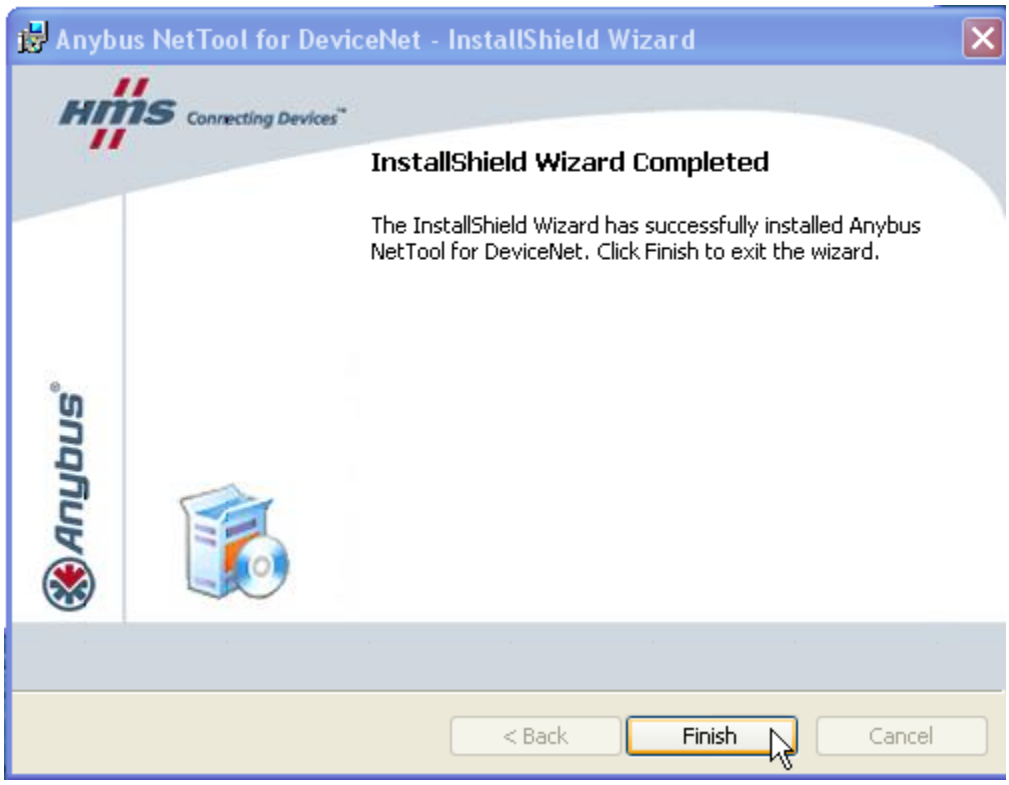


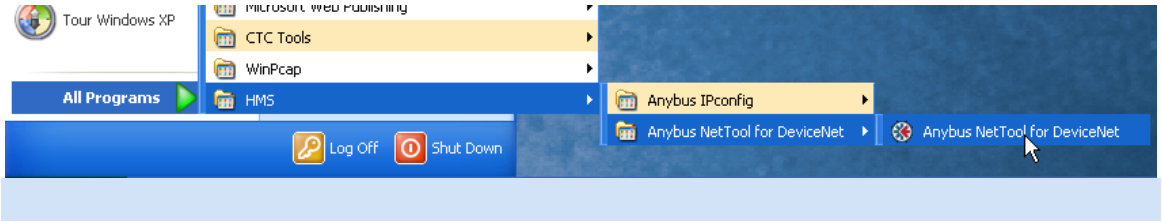
Once installation is complete, click Close followed by the Finish dialog for each transport:



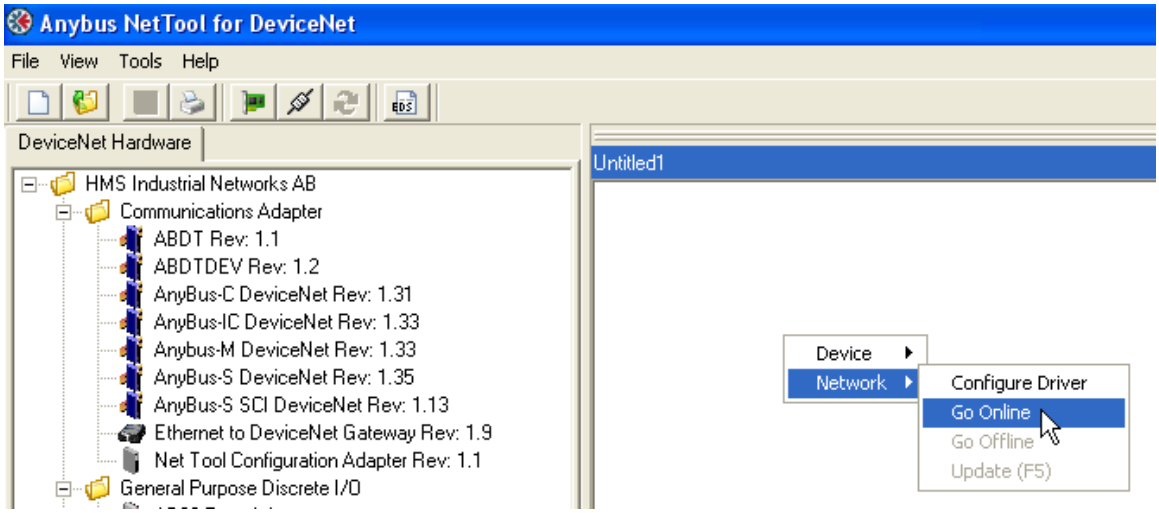



Reboot your PC prior to proceeding. Once rebooted install any USB to Serial Converters that are needed. Attach the NetTools serial cable and module with power, and then proceed to invoke the tool:

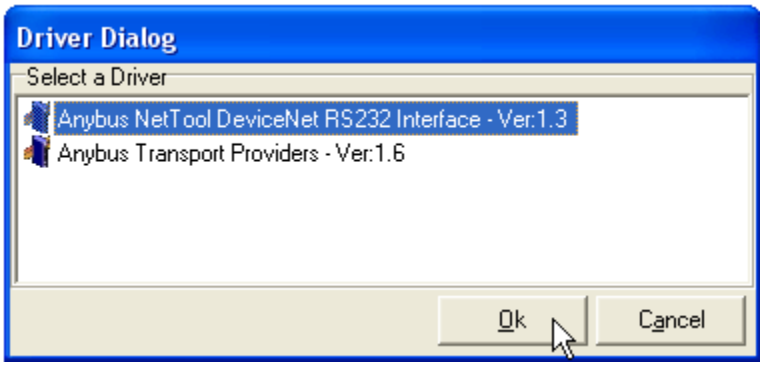


## *Online Configuration & EDS File Importing*

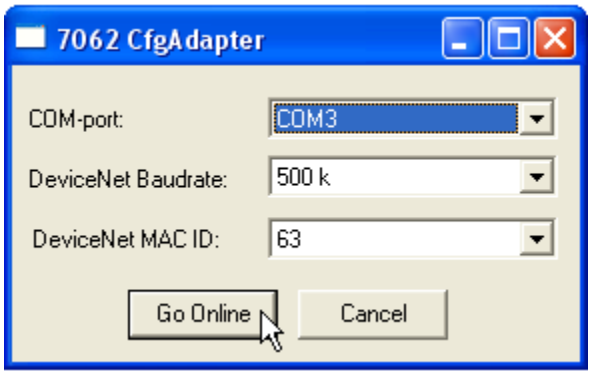
Right click the network screen to the right of the device hardware list. The Tools menu can also be used but for some reason upon first installation it is grayed out and the right clicking on the network screen is needed. Select Go Online.



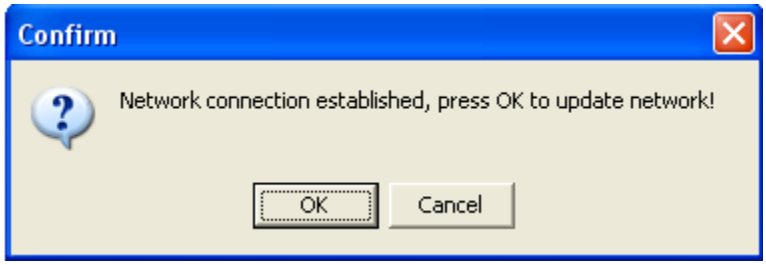

Highlight the RS232 driver and click OK:




Select the proper serial port and MACID used. The MAC ID is for the PC, so make sure it is not used on the network:

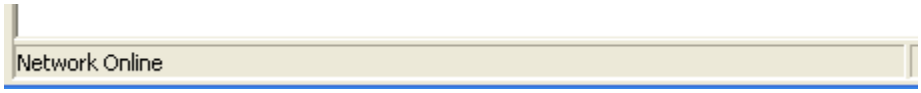

7062 CfgAdapter
COM-port:
COM3
DeviceNet Baudrate:
500 k
DeviceNet MAC ID:
63
Go Online
Cancel

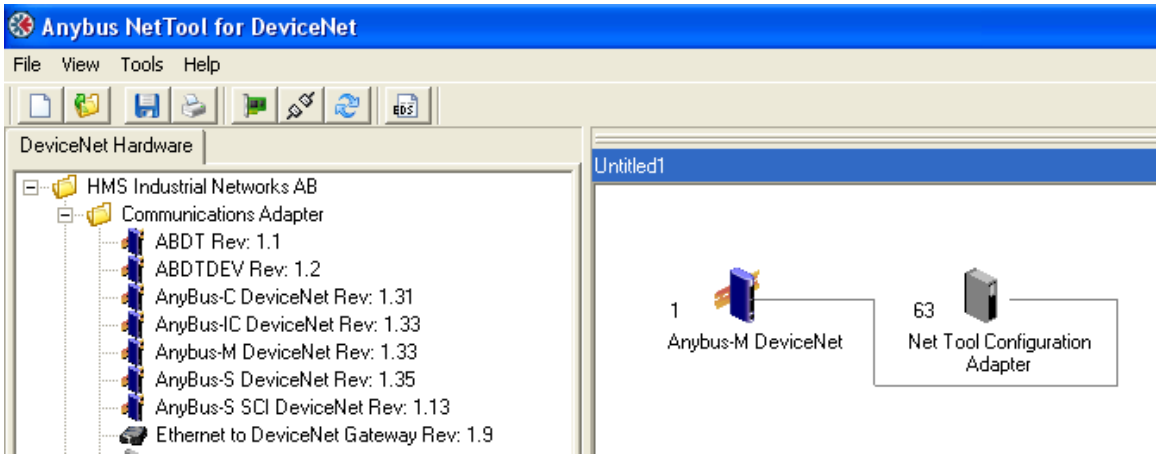
Once Go Online is selected the below box should appear which verifies the serial port connection to the Anybus Serial/CAN converter module and that at least one node can be found:



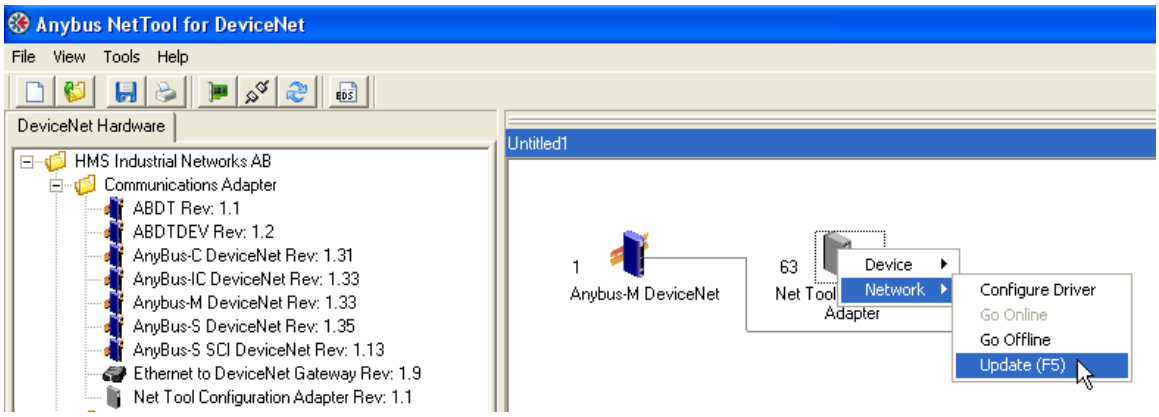

The network is scanned for additional devices:


Network Online status means at least one device is present and the network icon should appear, in this case a DeviceNet Master at address 1:

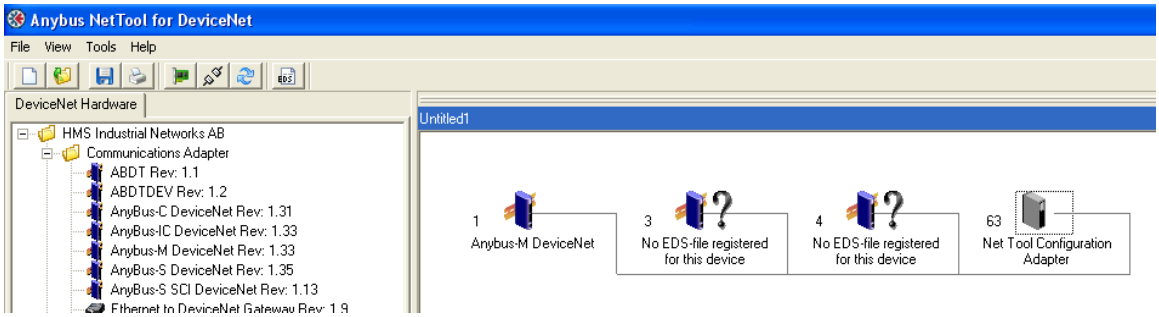


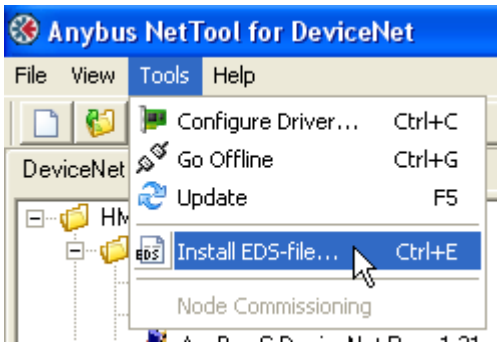
As a test 2 Wago controllers are plugged in and the NetTool adapter is requested to update the network configuration::



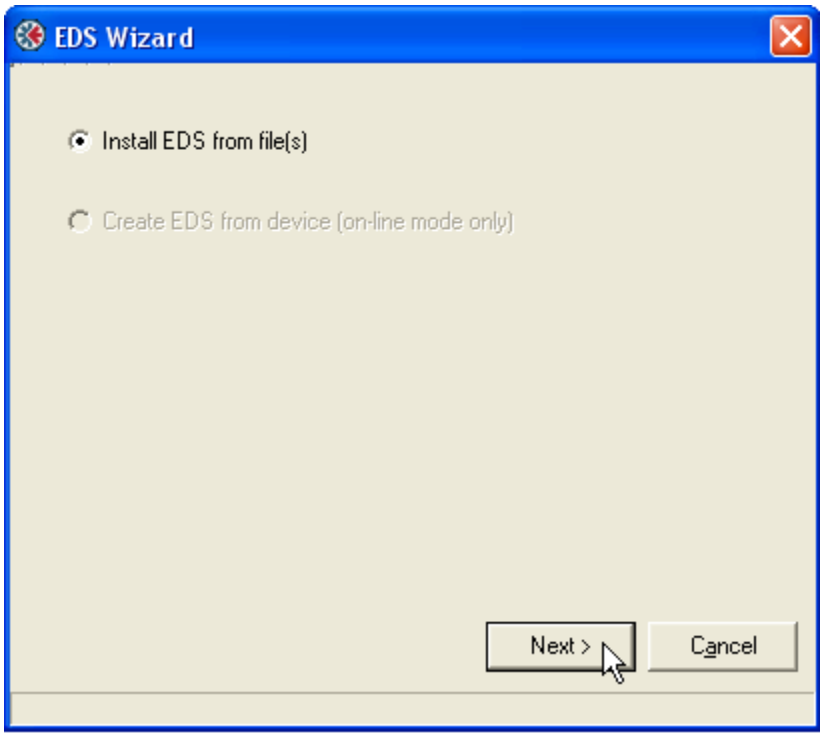

The devices are shown with a '?', meaning theEDS file has not been installed but the devices are communicating:

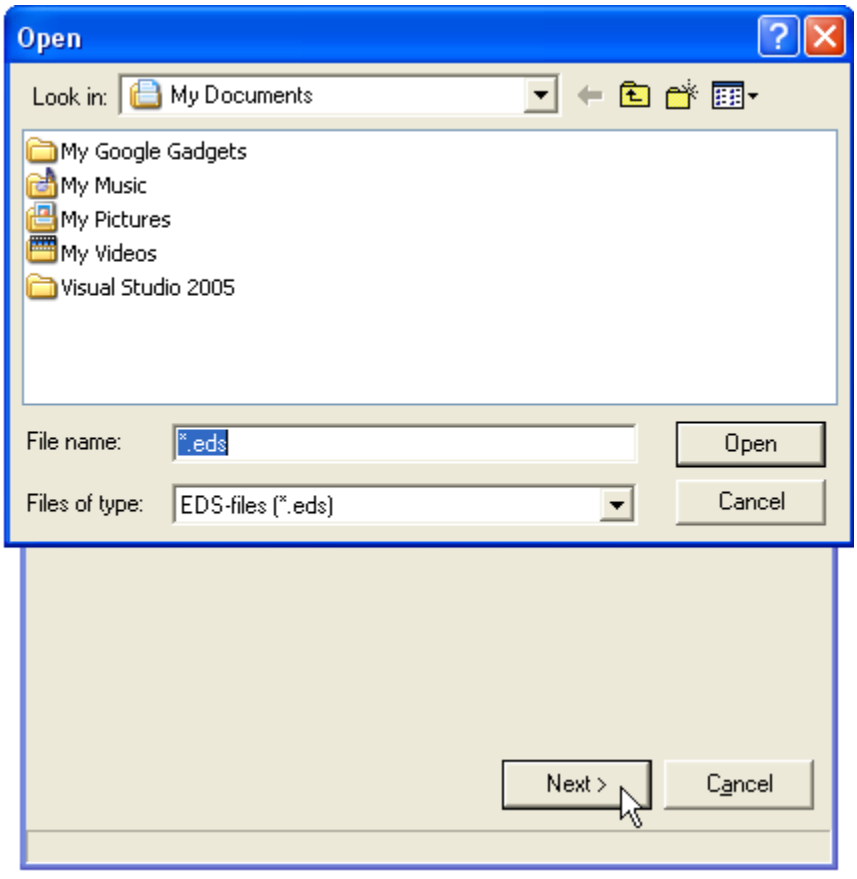


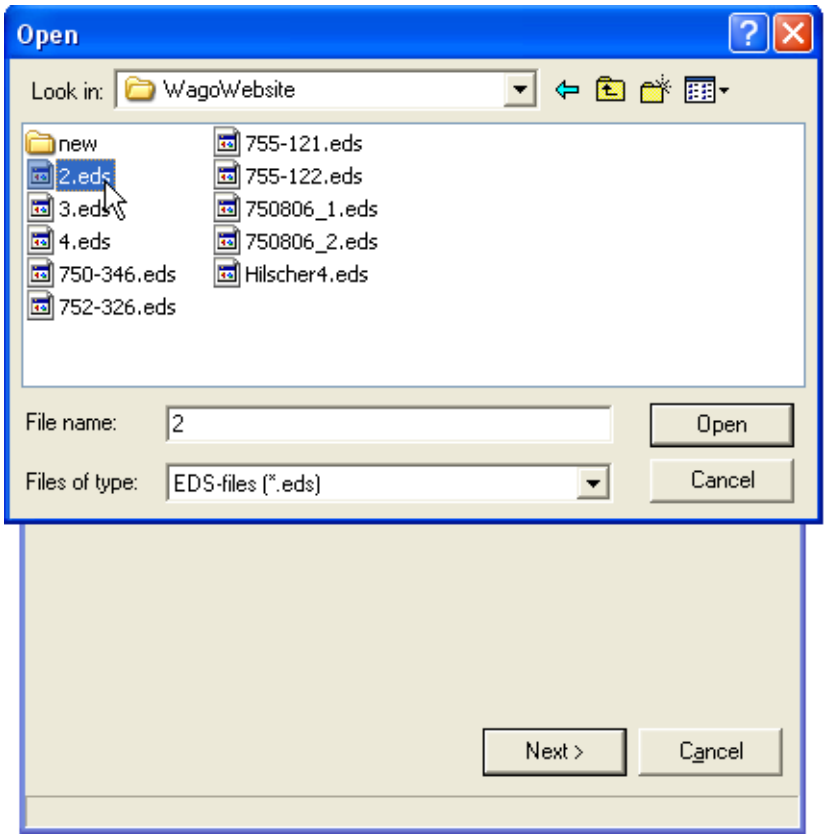

The EDS file can be installed by accessing the Tools menu and selecting Install EDS-file:



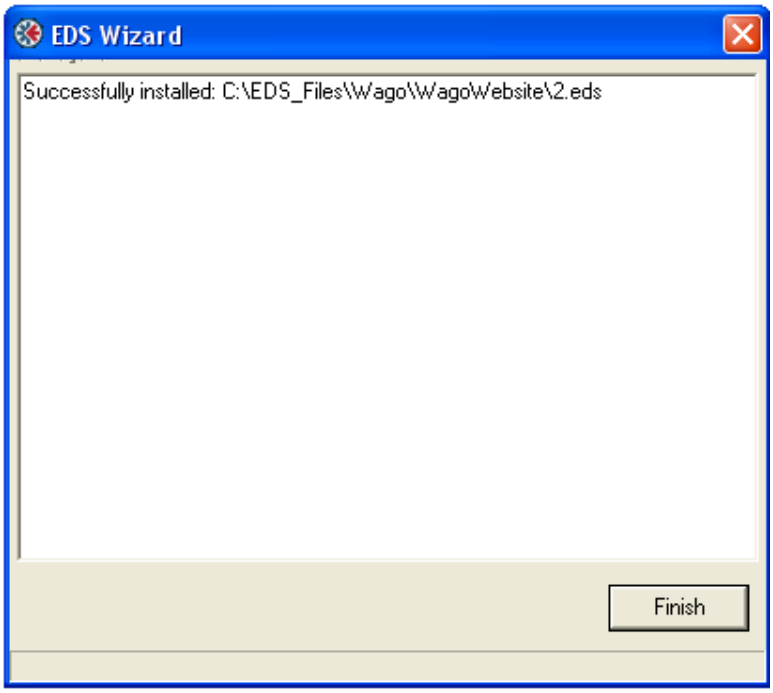
Click Next to install an existing EDS file, locate it in the file browser window and click Open:



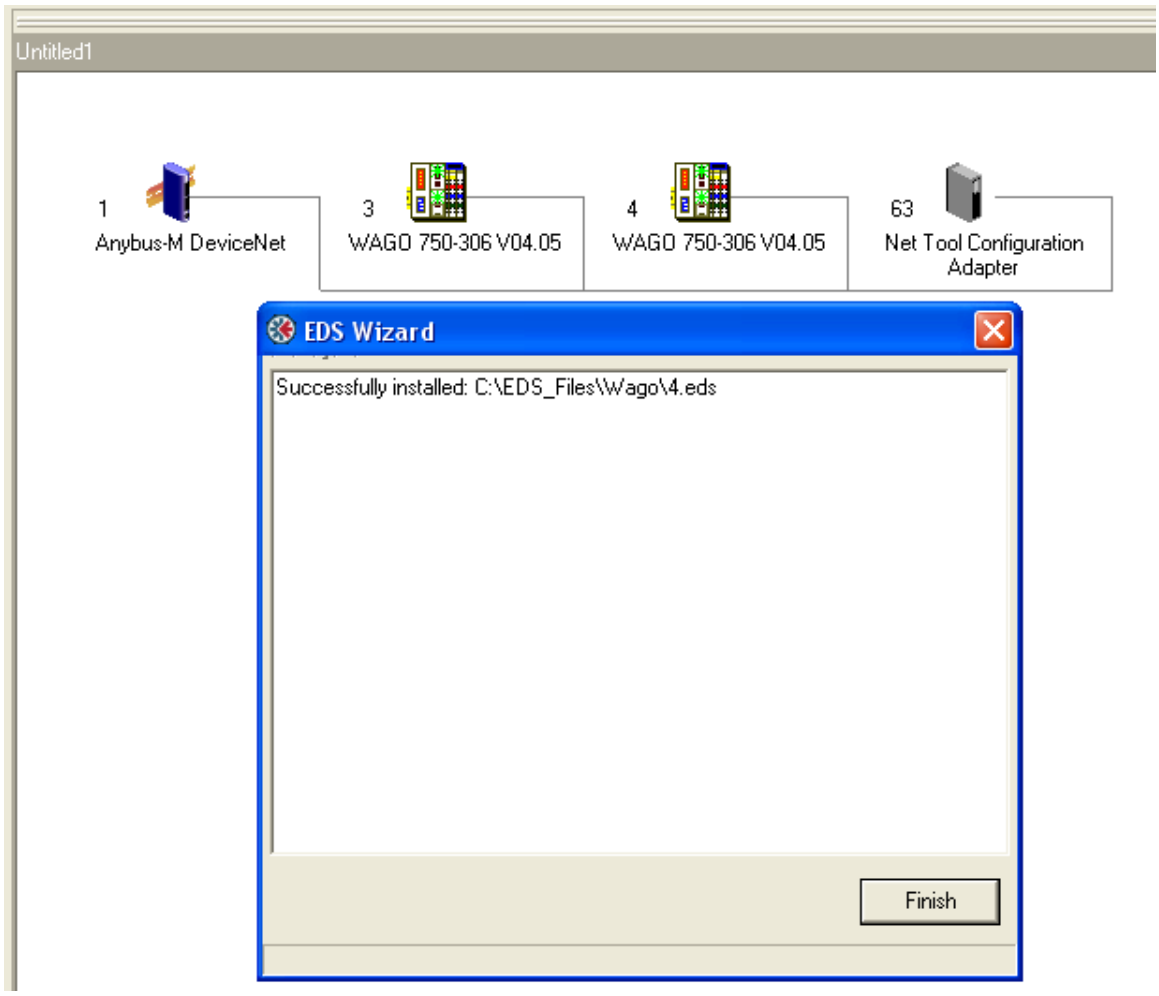



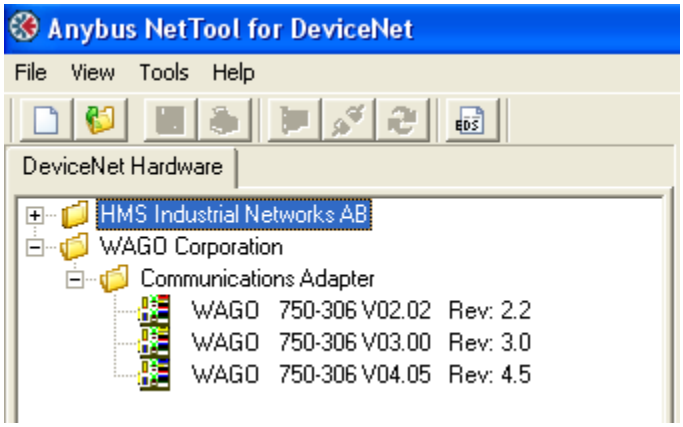



If the EDS file installs successfully the following will appear; a poorly formatted file will give errors:




With the EDS file now loaded the '?' goes away and the controllers are identified:


Untitled1
1
Anybus-M DeviceNet
3
WAGO 750-306 V04.05
4
WAGO 750-306 V04.05
63
Net Tool Configuration Adapter
EDS Wizard
Successfully installed: C:\EDS_Files\Wago\4.eds
Finish


Anybus NetTool for DeviceNet
File View Tools Help
DeviceNet Hardware
HMS Industrial Networks AB
WAGO Corporation
Communications Adapter
WAGO 750-306 V02.02 Rev: 2.2
WAGO 750-306 V03.00 Rev: 3.0
WAGO 750-306 V04.05 Rev: 4.5

*Blank*

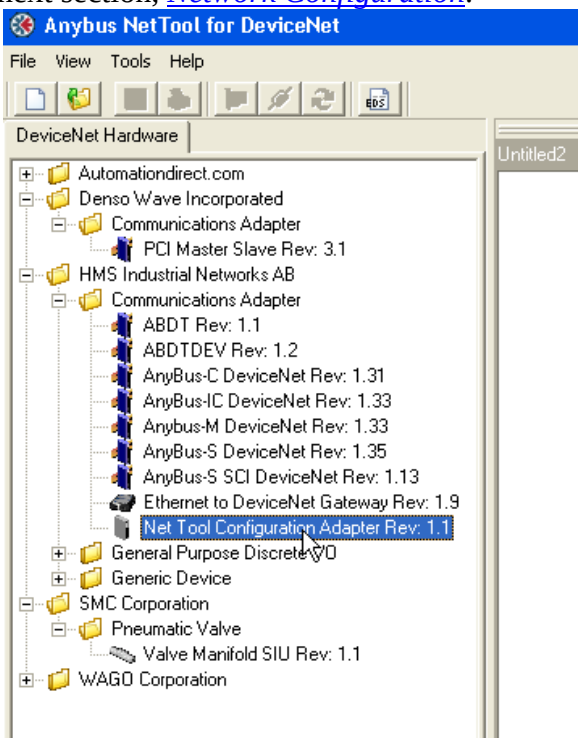
CHAPTER 6

# [6] NetTools Example

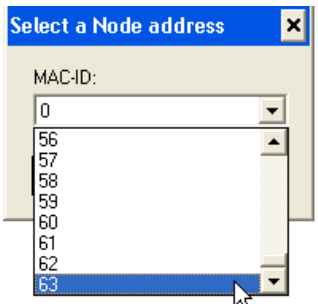
This section assumes an understanding of NetTools as well as how to install the needed EDS files. It is assumed that you have referenced the NetTools documentation for installation and initial setup. The setup and configuration of a Denso Robot and SMC valve module is demonstrated in this chapter as well as how that I/O would map into the Model 5300 register set.

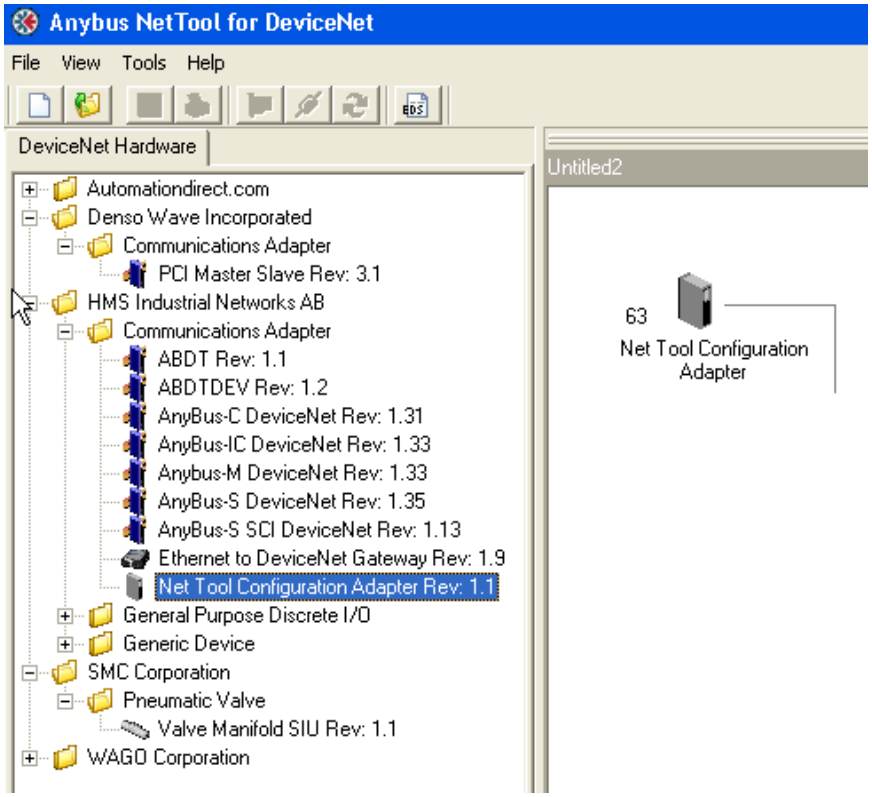
## *Initial Device Discovery*

Begin by invoking your NetTool software and preparing a new network configuration. Typically a configuration will be done online which would begin by dragging and dropping the HMS NetTool Configuration Adapter onto a new project. If configuring offline proceed to the next section, *Network Configuration*.

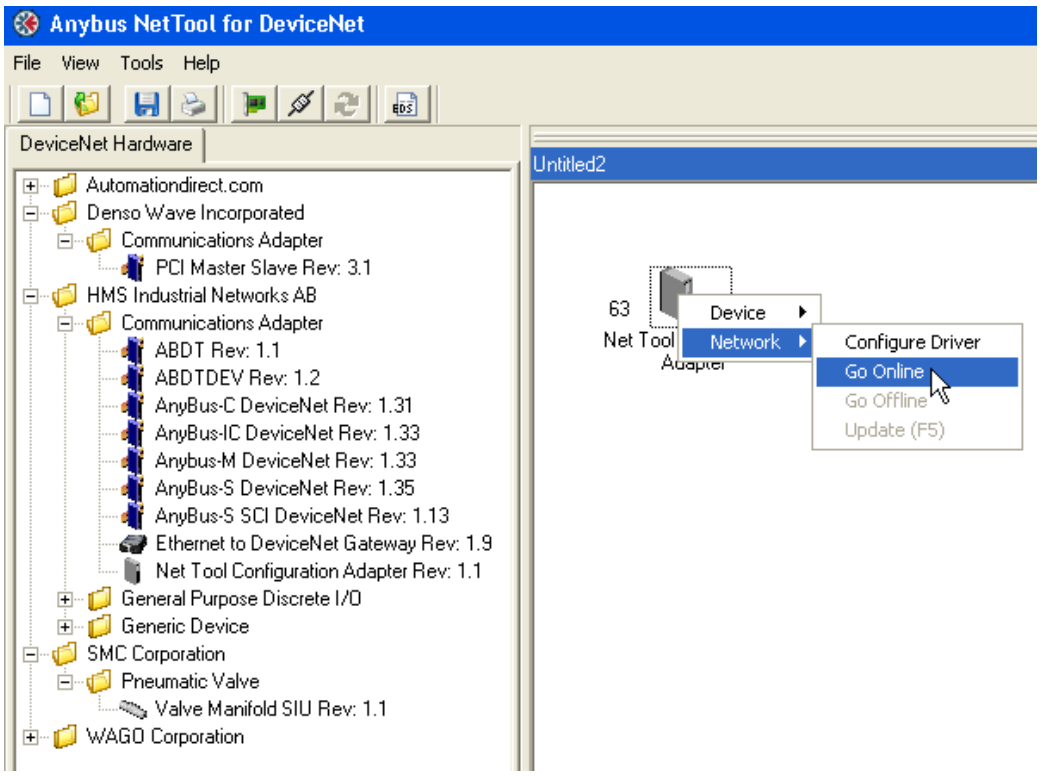


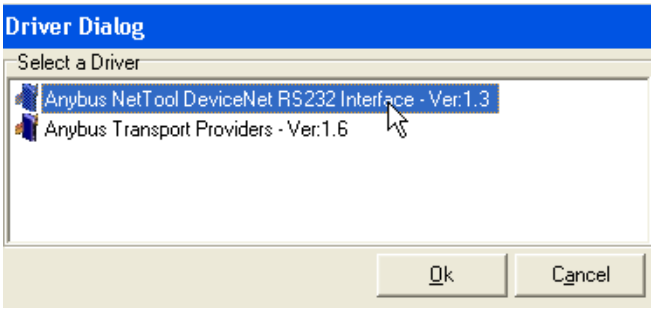
Suggest a MACID of 63, but any unused ID can be used.



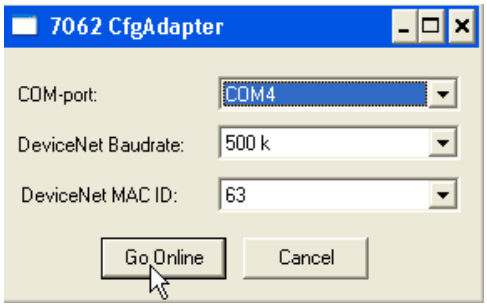




With the Configuration Adapter appearing you may now attempt to go online and check what devices are available for configuration. Right click the NetTool Configuration Adapter, clicking Go Online. Once online go back and click Update and all identified DeviceNet nodes (Slave and Master) will automatically appear similar to the screens which will be shown in this section. You may also configure offline and save the configuration to a file but it is best to ensure initial connectivity and EDS file matching.




Select the serial port that the RS232/DeviceNet converter is connected to as well as the DeviceNet Baud Rate to use and MACID of the converter (must be an unused MACID).

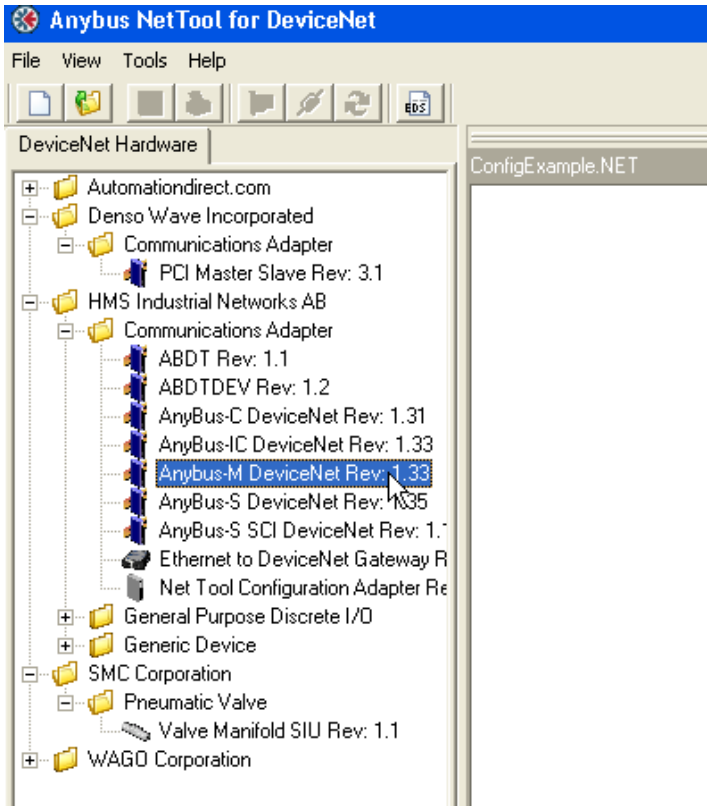


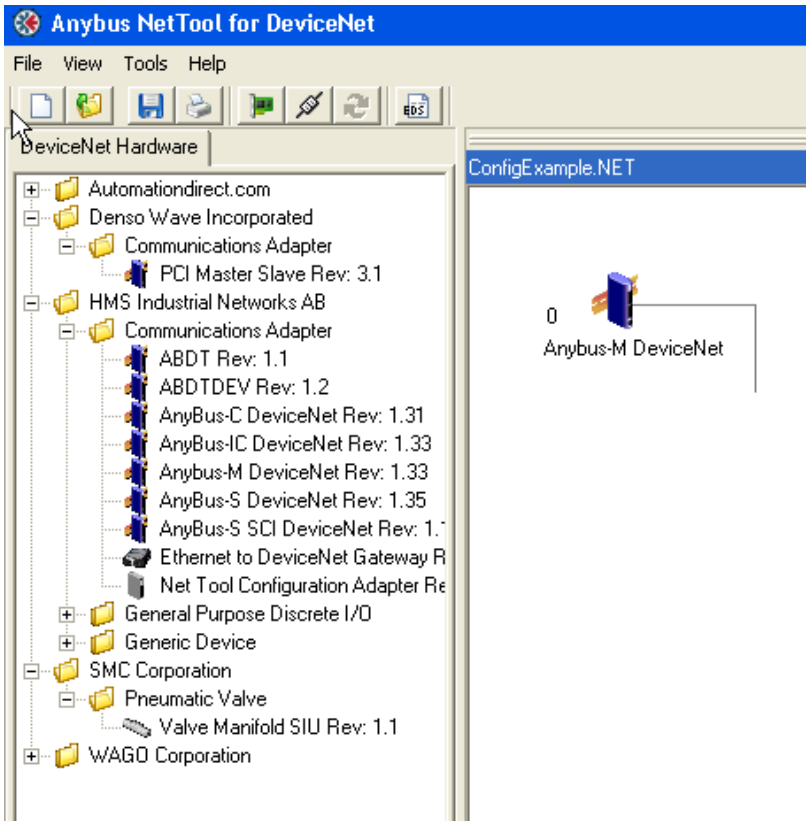

All nodes should automatically appear in the network window, ready for configuration after the Update menu item is selected.

## Network Configuration

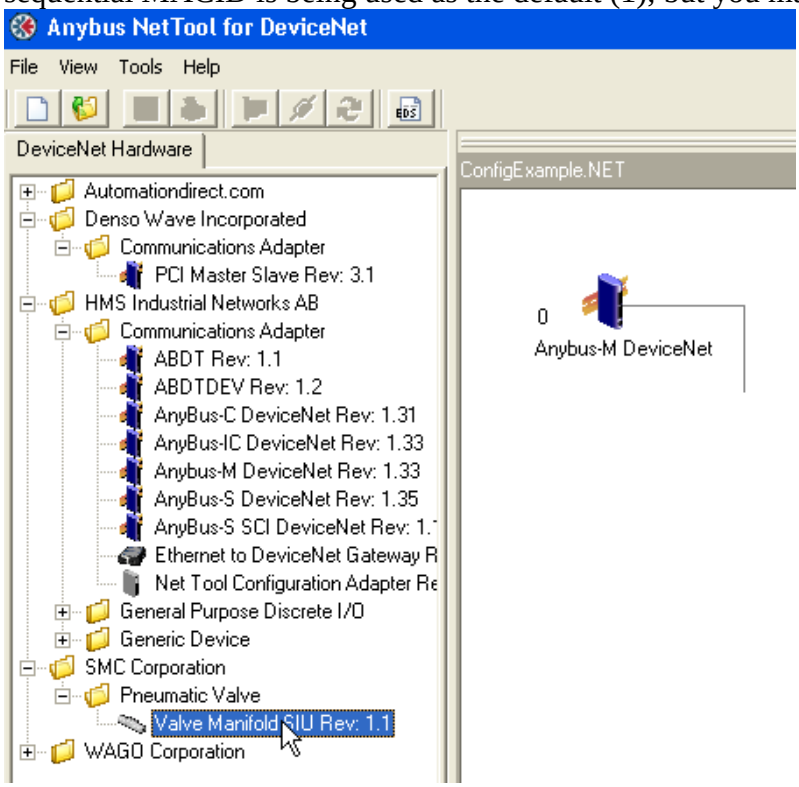
This configuration is being done offline but assuming you are connected to the real network it should appear the same, except the NetTool Configuration Adapter is used to insert the devices using the Update menu selection. Thus only if offline do you need to drag and drop the device from the available hardware menu to the network box on the right being configured. If using online configuration the resulting screens appearing below will at a minimum portray what should have appeared during the Update.

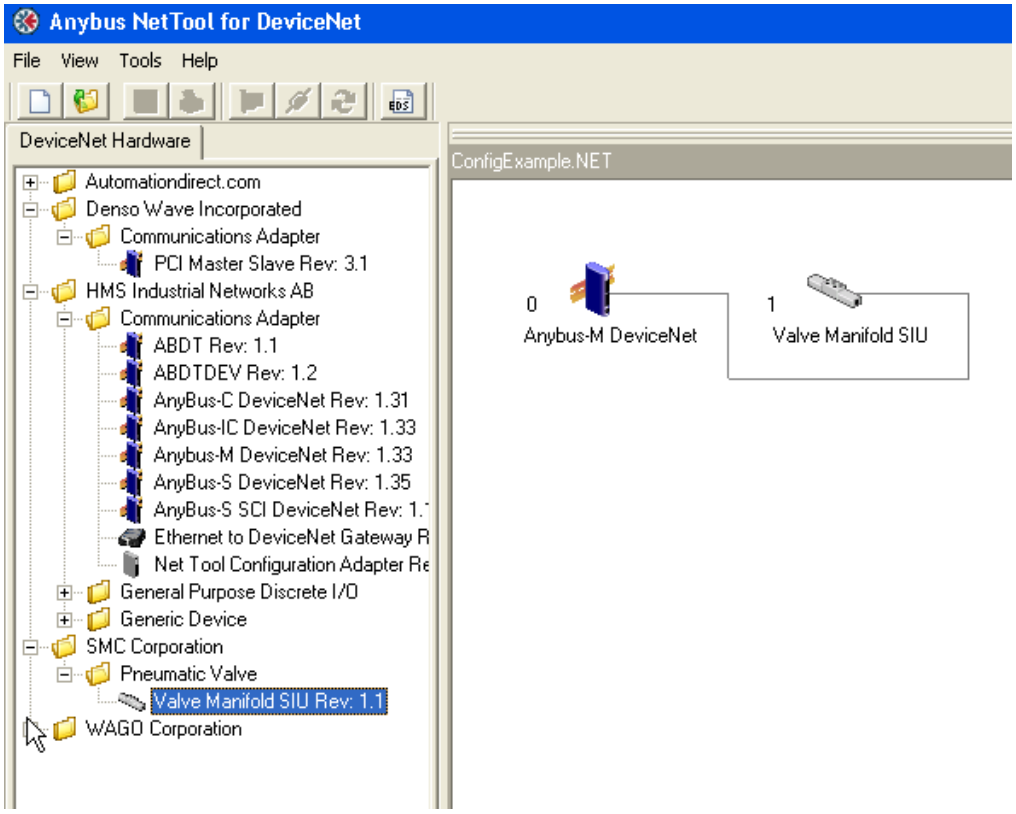
Begin by dragging the M3-61A Anybus-M DeviceNet module to the right side window:



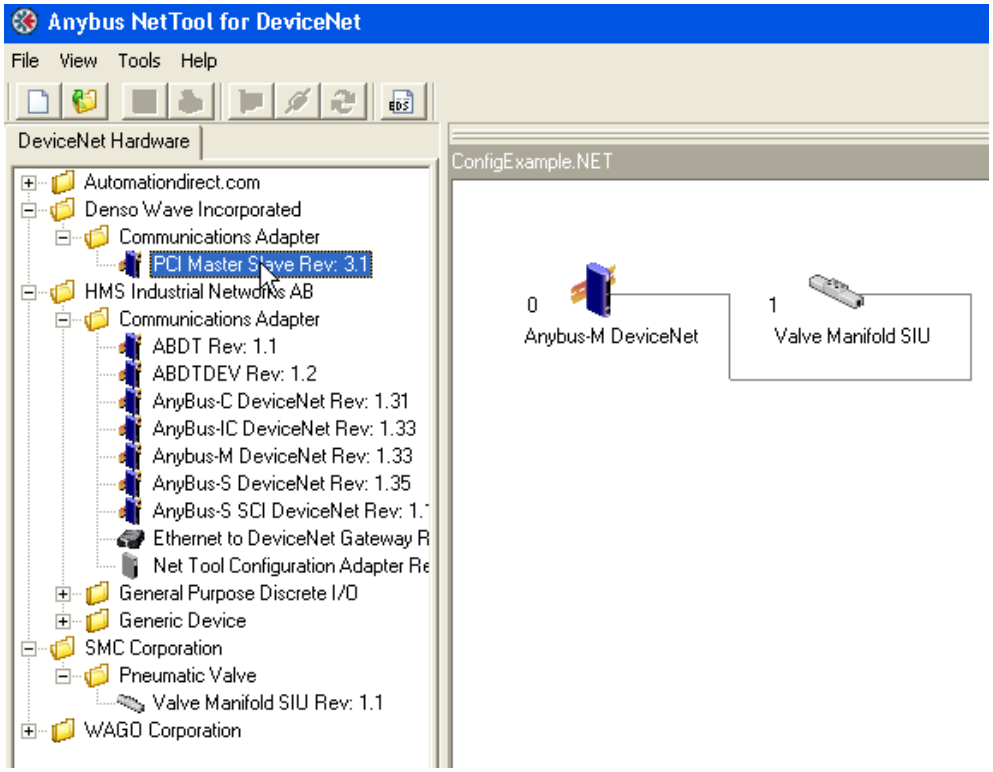


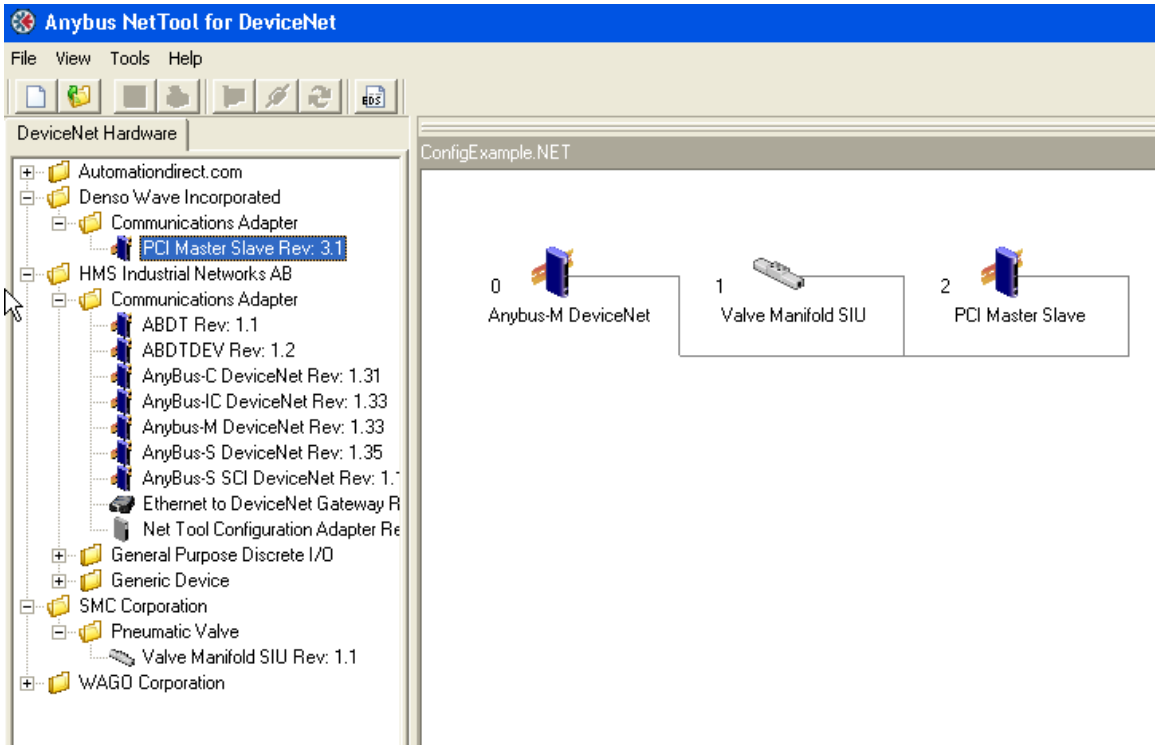

Next drag the SMC Corporation Valve Manifold to the right window. The next sequential MACID is being used as the default (1), but you may set as needed:



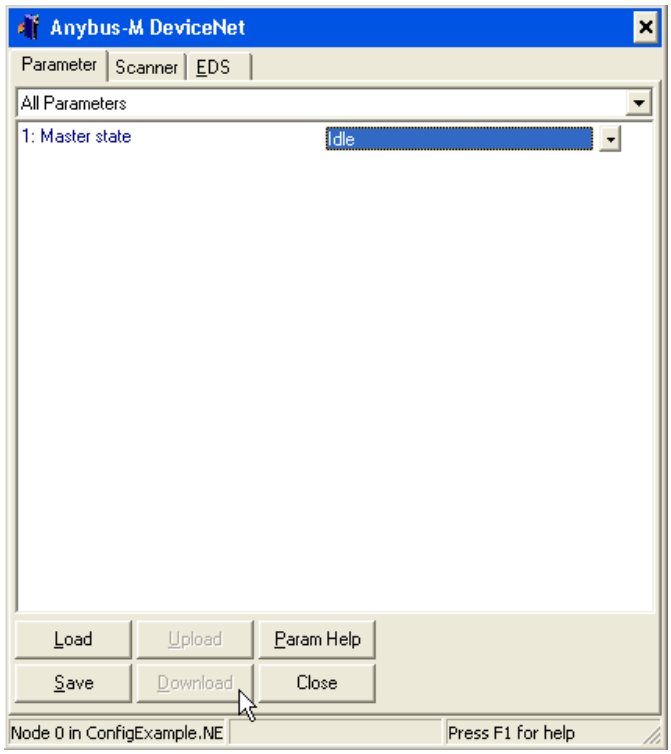



Next drag the Denso PCI Master Slave Device to the network window. The next sequential MACID is being used as the default (2), but you may set as needed:

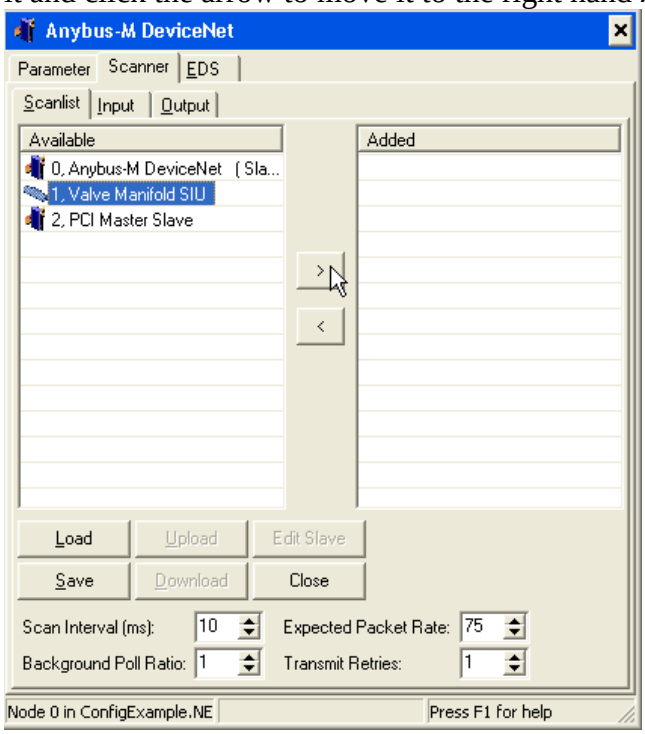




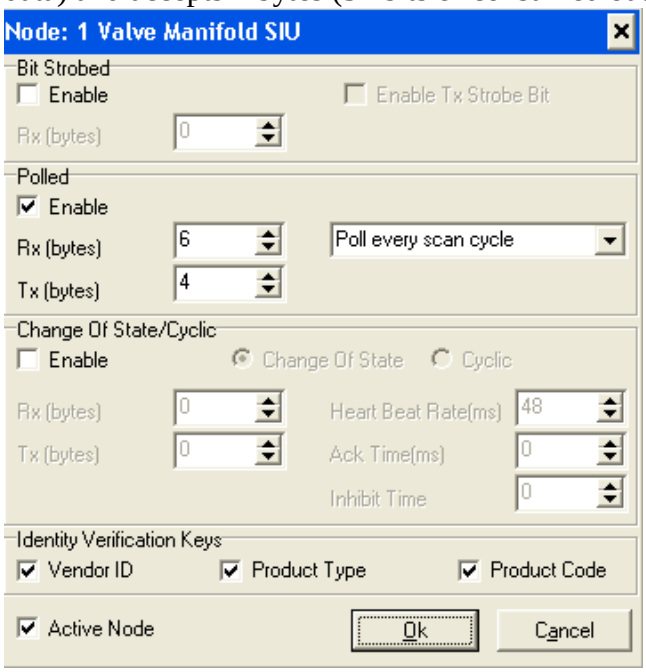

Double click the Anybus-M DeviceNet node in order to get into the scan list configuration screens. If not in Idle mode, use the pull down box, setting it to Idle and then click the Download box, which sets programming mode (only if online):

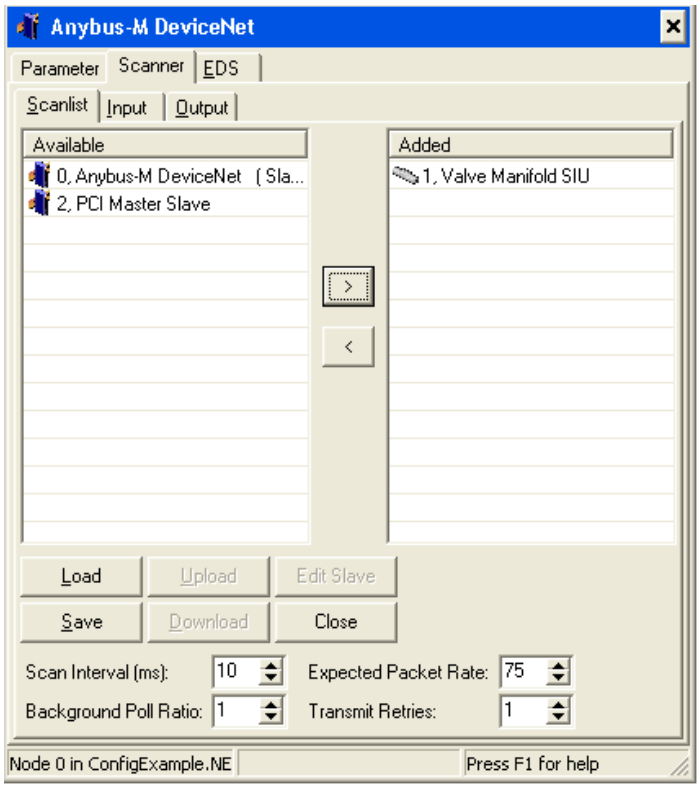


We will now begin to configure the individual DeviceNet slave nodes. Select the Scanner tab to view those available. Beginning with the SMC Valve module, highlight it and click the arrow to move it to the right hand Added side.

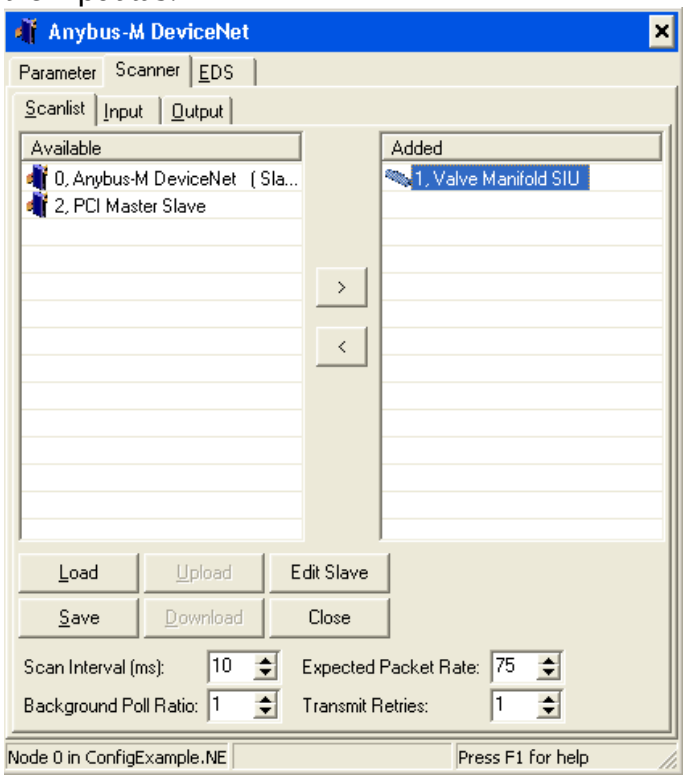


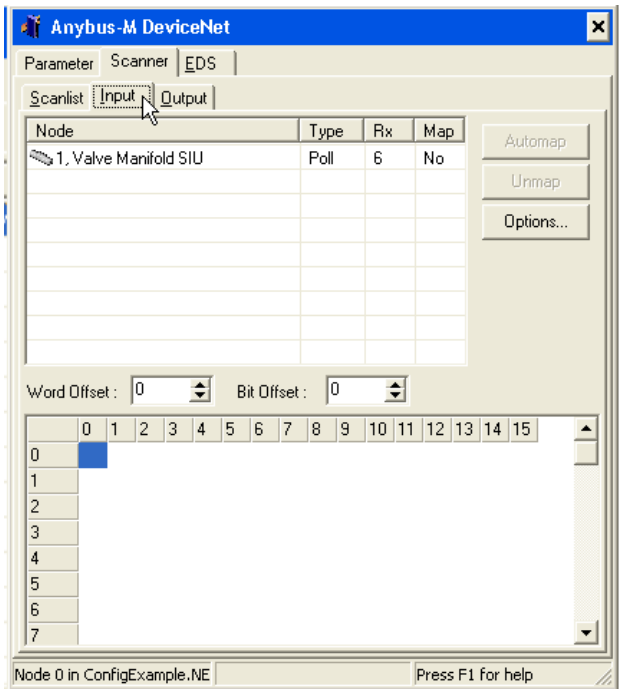

Upon adding, the dialog below will appear requesting the scan information to be entered. All three types are supported by the Master and each block of data returned will be mapped as unique I/O. For the purpose of this example we are assuming Polled only. The dialog below shows that the SMC device returns 6 bytes (48 bits of produced input data) and accepts 4 bytes (32 bits of consumed output data):



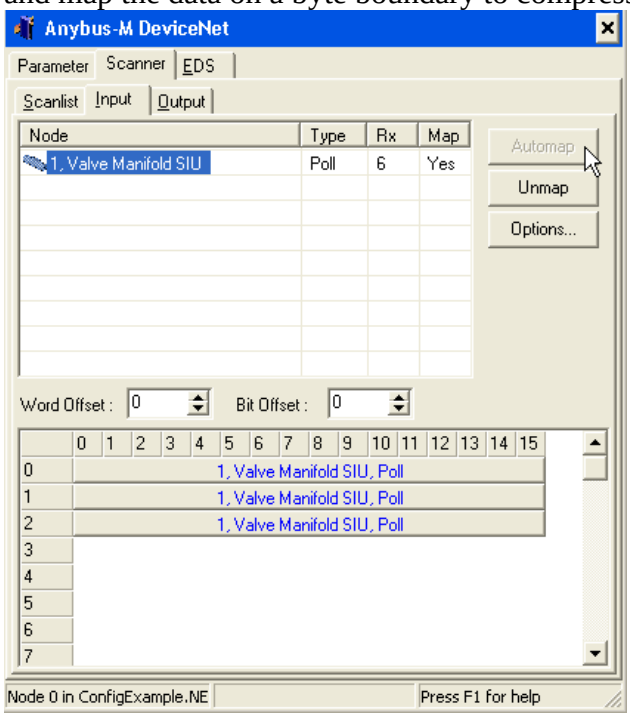



We will now begin to map the I/O of the SMC device. Highlight the device and select the Input tab.

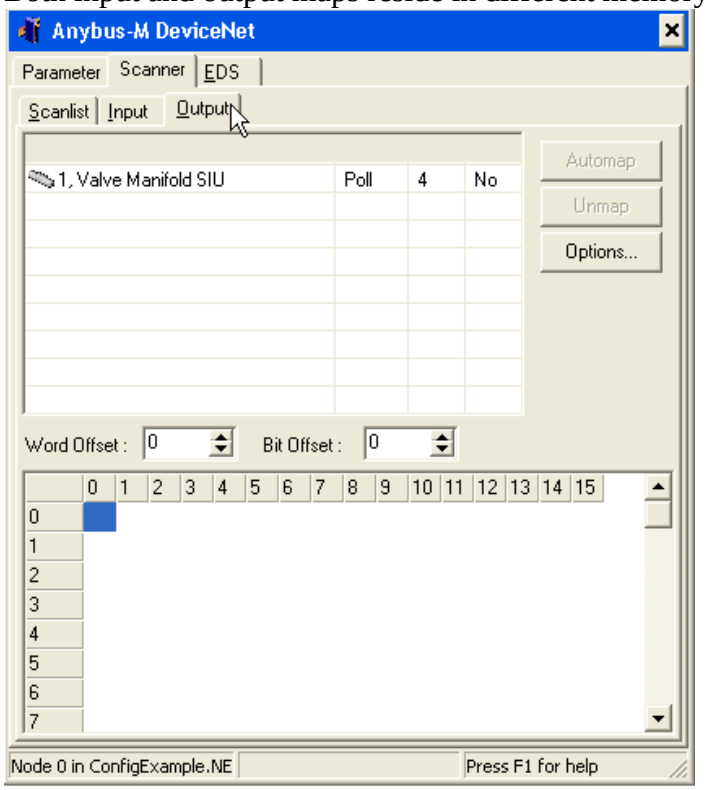


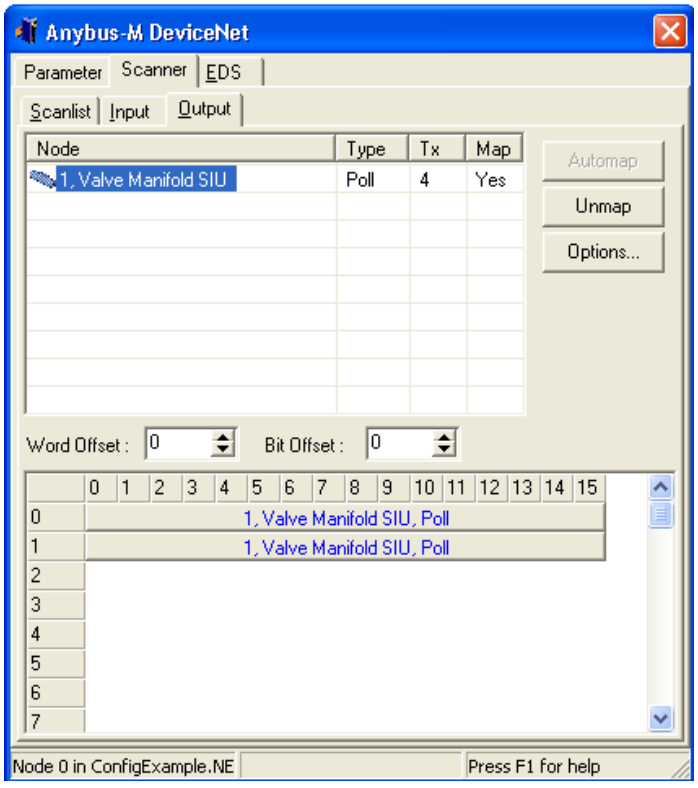



Note that we map all produced (input) data starting with a word offset of 0 and building up to 256 words (512 bytes). A maximum of 1024 inputs can be supported requiring only 128 bytes; the remaining area is available for scanning analog or other storage parameters available on the particular device you are interfacing to. Outputs are treated the same way. Highlight the Valve Manifold SIU and click Automap. 6 bytes of data will be mapped into memory representing added Model 5300 inputs 1 to 48, first bit being 1. If we had an odd number of bytes, say 5, you could click the Options button and map the data on a byte boundary to compress it.

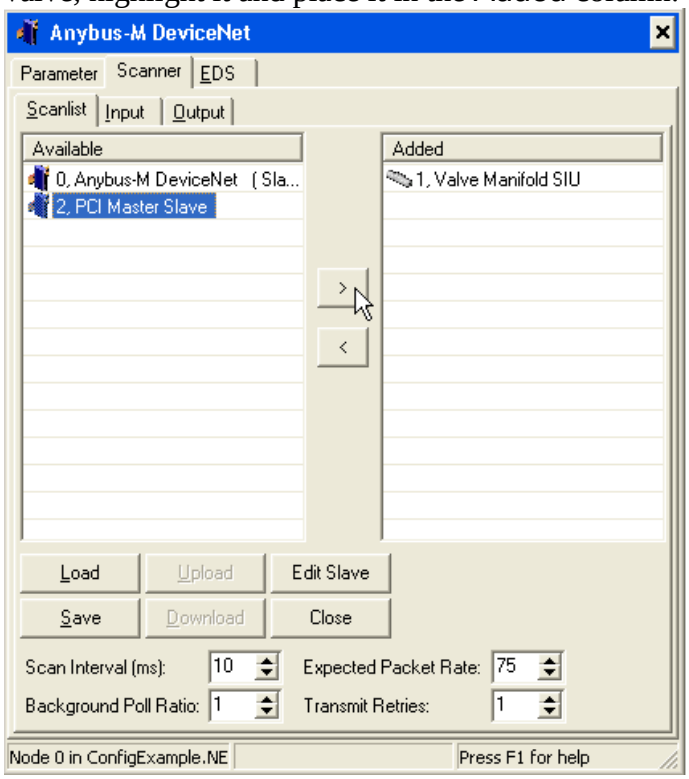


Next we'll configure the consumed (output) data. The valve has 4 bytes or 32 bits of discrete outputs. Mapping is similar to the input data except you select the Output tab. Both input and output maps reside in different memory partitions.

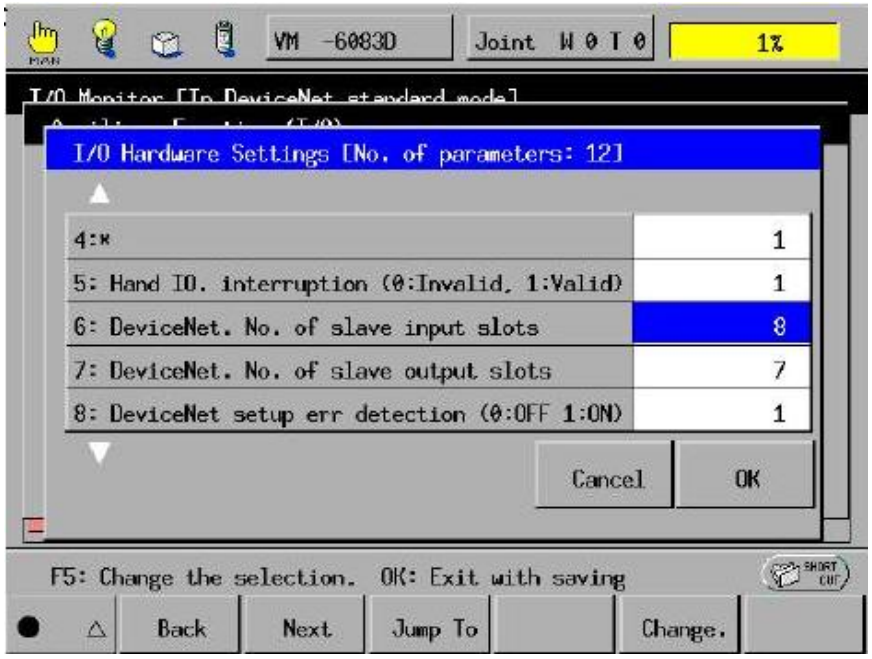


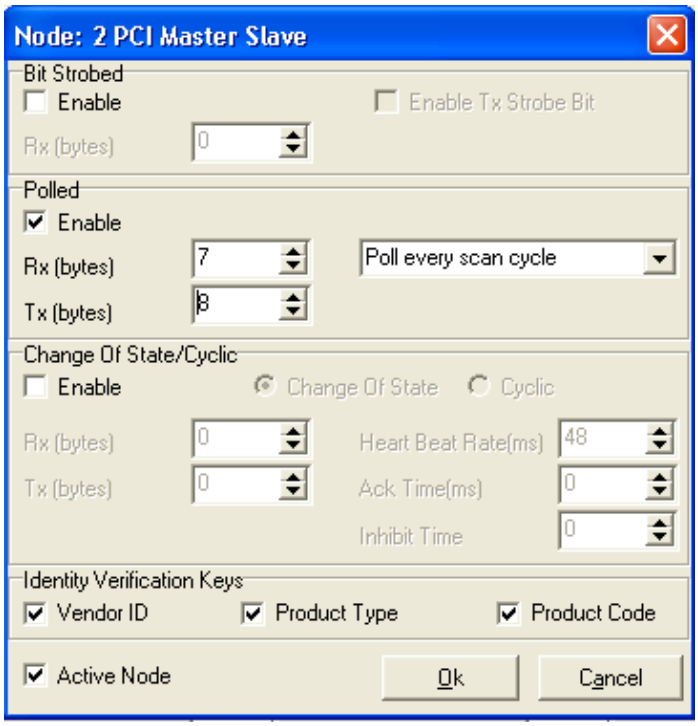



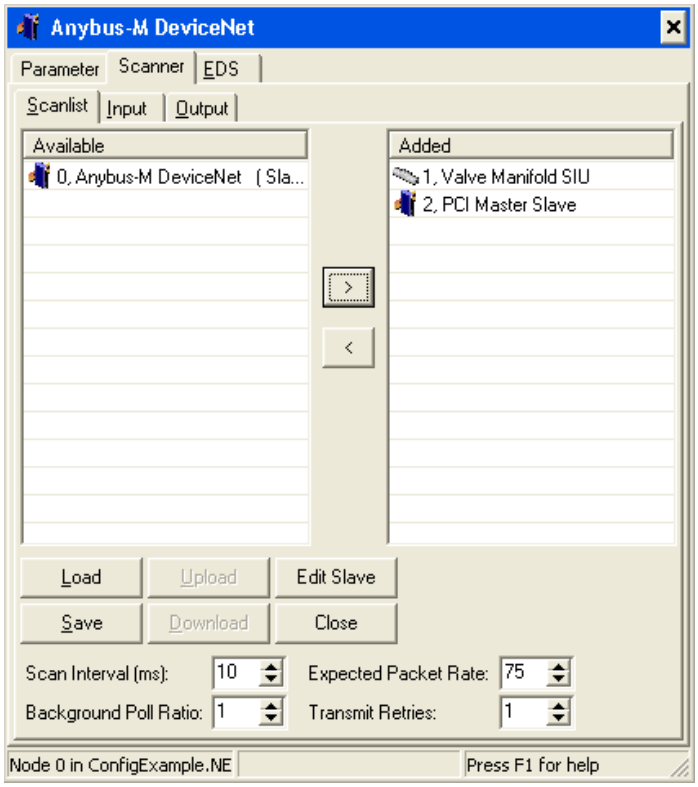
Once the SMC Valve is configured it is time for the Denso PCI Card. As with the SMC valve, highlight it and place it in the Added column:



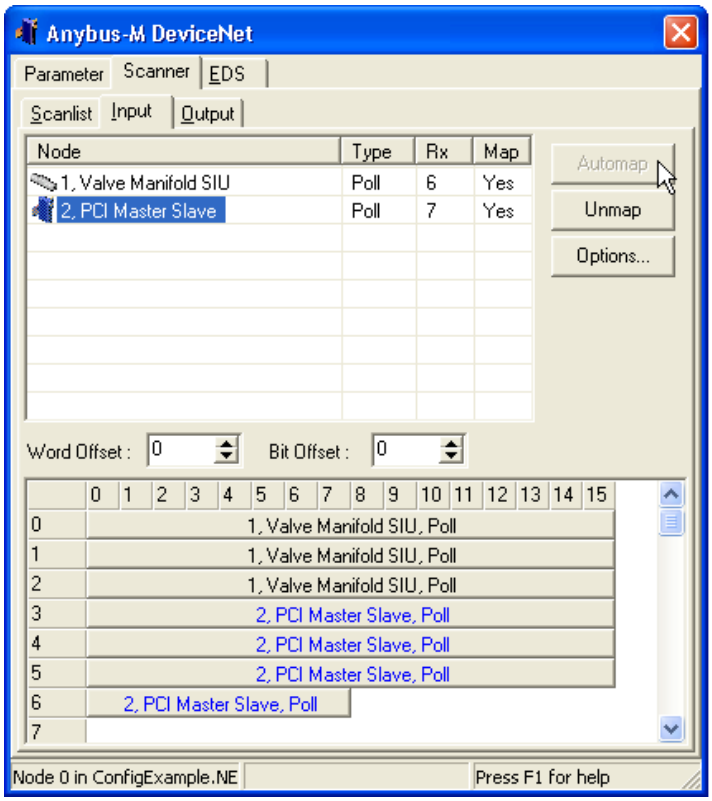

Prior to being added the proposed access methods, bytes received (input) and transmitted (output) are defined. The EDS file defines this as 32 bytes for each or 256 bits of data. In reality you must reference the configuration of the Denso to determine the proper setting or errors will occur. The popup screen below is from page 51 of the *Denso Robot with RC7M Controller **-G Series Options Manual*, Step 6:



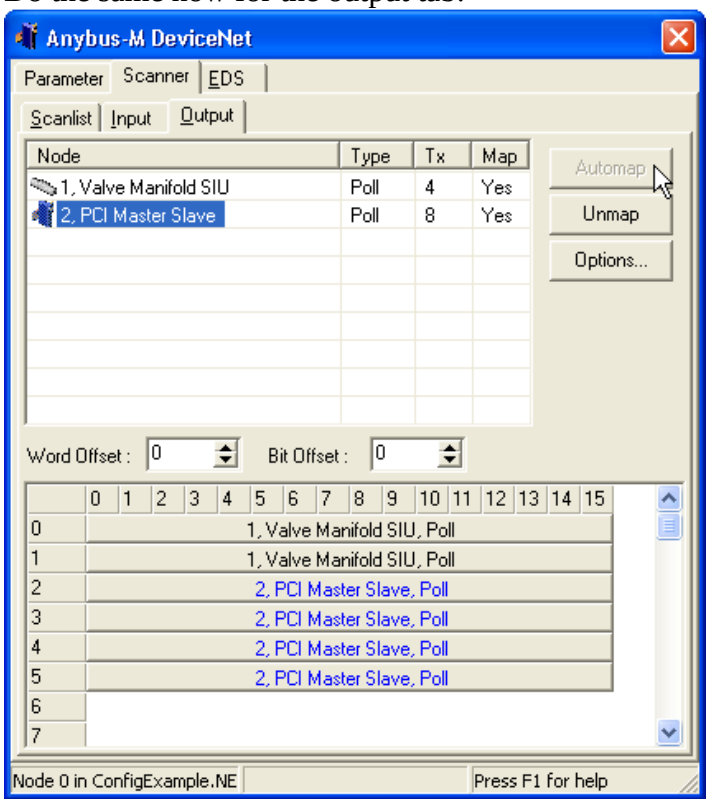



Highlight the PCI Master Slave entry in the Added column and select the Input tab. As before highlight the PCI entry and click Automap. The input bits will appear after the SMC Valve entry:



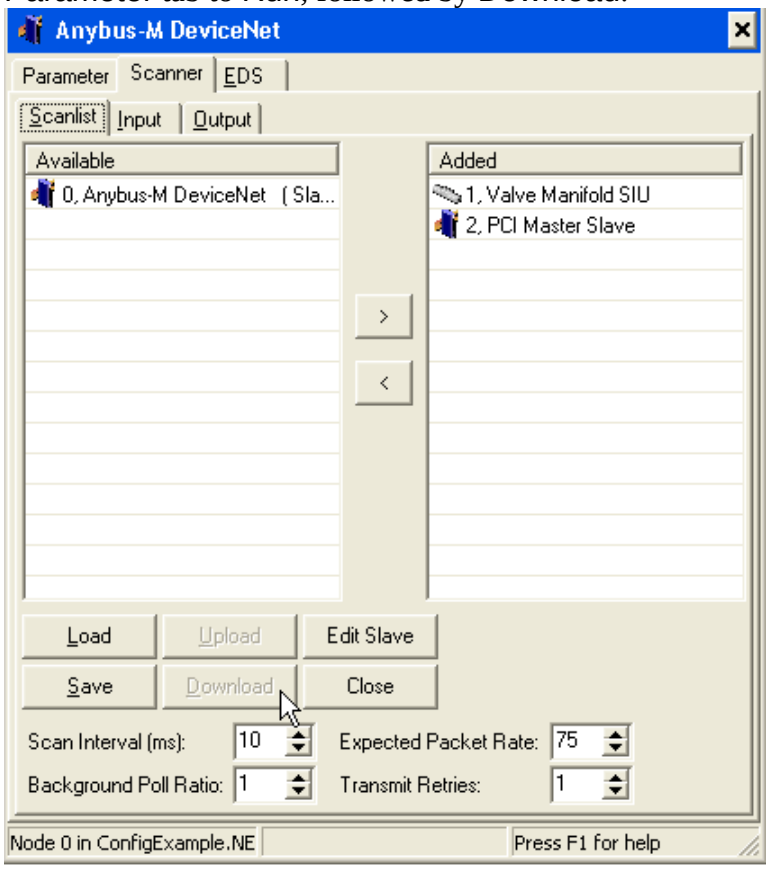

Do the same now for the output tab:



Input/Output data is now fully mapped. From a Model 5300 perspective the DeviceNet input data is treated as normal digital inputs beginning after all existing cards. If no other cards with inputs/outputs then:

SMC Valve = Inputs 1 to 48, Outputs 1 to 32
Denso Robot = Inputs 49 to 112, Outputs 33 to 88 (note that Input 49/Output 33 is first on the robot).

Once you have confirmed your entries the M3-61A must be loaded with the scan list information. From the Scanner tab, if you are online, the Download button will be highlighted. Click it and you will see the information downloaded to the M3-61A and stored in its internal memory. Note that you must be in idle mode as set at the beginning of this configuration process (Parameter tab). Once downloaded you should set the Parameter tab to Run, followed by Download.




After full configuration it is necessary to cycle power on the Model 5300 controller to ensure that the DeviceNet I/O and local I/O install correctly. The Model 5300 will not operate until all scanned devices that were defined are properly seen as online, as it is assumed that their appearance is required for proper application program execution. You will know things are running properly when all 3 LED's are green and none are red. A flashing red indicator means there is a network setup error.

## Administrative Screen DeviceNet Window

The Model 5300 can be accessed via telnet or a serial port in order to access the standard Remote Administrative screen. From this screen general node status and version information can be obtained. The command to retrieve this information is `get anybus info`. Upon execution something similar to below will appear for each module installed:

```
BlueFusion/>get anybus info
*** BOARD #1, Slot 4 ***
M3-61A DEVICENET MASTER MODULE CONTROL INFO:
  XML Config Ver = 0x0201
  Avail Heap Mem = 458632
  Number Tags =    0x0003
  Active Exp Msgs= 0
  Boot Ver =       0x2301
  Mod Ver =        0x3101
  Fbus Ver =       0x0000
  Serial No =      0x0DAB0AA0
  VendorID =       0x0100
  Fbus Type =      0x2500
  LED[ ] =         0 1 1 1
  Module Status = 0x0102
M3-61A DEVICENET MASTER MODULE FIELDBUS INFO:
  Module Status = 00 02 04 00 00 4B 01 0C
  Node Active =   00000400 00000000
  Node Faulted =  00000000 00000000
  Total DIN    =  16
  Total DOUT   =  8
  Total AIN    =  0
  Total AOUT   =  0
  Node Status = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*** BOARD #2, Slot 6 ***
M3-61A DEVICENET MASTER MODULE CONTROL INFO:
  XML Config Ver = 0x0202
  Avail Heap Mem = 457716
  Number Tags =    0x0003
  Active Exp Msgs= 0
  Boot Ver =       0x2301
  Mod Ver =        0x3101
  Fbus Ver =       0x0000
  Serial No =      0xE8340BA0
  VendorID =       0x0100
  Fbus Type =      0x2500
  LED[ ] =         0 1 1 1
  Module Status = 0x0102
M3-61A DEVICENET MASTER MODULE FIELDBUS INFO:
  Module Status = 00 02 04 00 00 4B 01 0C
  Node Active =   00000008 00000000
  Node Faulted =  00000000 00000000
  Total DIN    =  8
  Total DOUT   =  8
  Total AIN    =  2
  Total AOUT   =  2
  Node Status = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The first two items on the screen are from a configuration file that is discussed in more detail later in this manual.

***XML Config Ver*** – This is the XML module configuration file version that is currently installed. The shown version is 2.01 and was defined by the user using the XML element <VERSION>0201</VERSION>. *Chapter 7: XML Configuration File & I/O Declarations* provides additional information.

***Avail Heap Mem*** – This is the amount of memory available for DeviceNet operation. It is referenced for diagnostic purposes to ensure explicit messages are returning memory to the system properly.

***Number Tags*** – The number of tags defined in the XML configuration file. Reference *Chapter 7: XML Configuration File & I/O Declarations* for additional information.

***Active Exp Msgs*** – This is the number of explicit messages that have been queued on the Anybus module and are awaiting a response. It must remain below 64 and at 40 all scanning will stop to throttle traffic. It is a good indication of how heavily loaded the module is.

Other information on the screen directly references the status information defined in the Control and Fieldbus areas of the HMI Anybus-M card. Reference their manual:

http://www.anybus.com/upload/86-2545-ABM-DEV-1.02.pdf

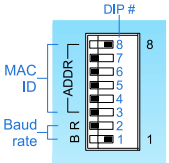
Summary of useful information, which appears in this manual, is provided below:

***Control LED [ ]*** - Byte array of the 4 large LEDs on the front panel. The important ones are the last 3; the first one is not used. In order to operate the last 3 must be 1's (on). The second LED will be a 2 if the configuration is in error.

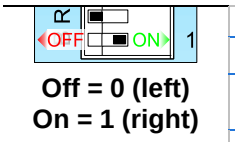
| Name | States | |
|---|---|---|
| **MS (Module status)** | Off | No power or not initialized |
| | Green | Module status is OK. |
| | Flashing red | Minor fault |
| | Red | Major fault |

***Fieldbus Module Status*** –

Byte 0, 1 – (0x0004), Active Connections, master counts as one as well.
Byte 2 – (0x04), Reserved
Byte 3 – (0x00), Reserved
Byte 4, 5 (0x004b), Expected packet rate in mS.
Byte 6 – (0x01), Dip switch settings




| MAC ID settings | |
|---|---|
| Address | DIP 3 – 8 |
| 0 | 000000 |
| 1 | 000001 |
| 2 | 000010 |
| 3 | 000011 |
| … | |
| 62 | 111110 |




**Off = 0 (left)**
**On = 1 (right)**

| | |
|---|---|
| 63 | 111111 |

**Baud rate settings**

| Baud rate (kBit/sec) | DIP 1 - 2 |
|---|---|
| 125 | 0 0 |
| 250 | 0 1 |
| 500 | 1 0 |
| Reserved | 1 1 |

Byte 7 – (0x0C), Scan flags.
The parameter called Scan flags contains information about the operation mode of the scanner. The major purpose of this parameter is to display the idle or run mode of the module. See table below for information about the meaning of the bits in the register. The module uses the reserved bits internally, but they do not give the user any useful information.

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| Reserved | Strobe active | Reserved | Idle mode | Reserved | ISD active | Poll | Reserved |

The bit "Poll" is set when the ISD timer has expired and the master shall produce data on established poll connections.

The bit "ISD active" (Inter Scan Delay) is set while the master is waiting for the Inter Scan Delay timer to expire. When the timer has expired, the master produces and sends data on all I/O connections that are using the ISD timer.

The "Idle mode" bit is set when the master is in idle mode.

The "Strobe active" bit is set when the master has any active bit-strobe connections.

***Fieldbus Node Active*** – Stored high byte to low byte, thus 648h is the 8th byte, 64Fh is the 16th byte. In the example, nodes 3 and 10 are actively configured in the scan list.

The Node Active Area is an 8-byte-long bit-array, which contains information about which nodes are configured in the master. If the bit is set (=1), the node is configured in the scanlist, and the master will try to establish connections to the node. If the bit is cleared (=0), the node is not configured, and the master will not communicate with the node.

**Note:** Node means a module in the network corresponding to a certain MAC ID or Node ID.

| | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|
| 648h | Node 7 | Node 6 | Node 5 | Node 4 | Node 3 | Node 2 | Node 1 | Node 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 649h | Node 15 | Node 14 | Node 13 | Node 12 | Node 11 | Node 10 | Node 9 | Node 8 |
| ↓ | … | … | … | … | … | … | … | … |
| 64Fh | Node 63 | Node 62 | Node 61 | Node 60 | Node 59 | Node 58 | Node 57 | Node 56 |

***Fieldbus Node Faulted*** – Stored high byte to low byte, thus 658h is the 8th byte, 65Fh is the 16th byte. In the example node no nodes are faulted since all are 0x00.

The Node Faulted Area is an 8 byte long bit-array, which tells which nodes in the network are not running correctly. If the bit is set (=1), the corresponding node is faulted. If the bit is cleared (=0), the node is operating correctly. For more information about the fault of the node, see the corresponding information in the Node Status Area.

**Note:** Node means a module in the network corresponding to a certain MAC ID or Node ID.

| | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|
| 658h | Node 7 | Node 6 | Node 5 | Node 4 | Node 3 | Node 2 | Node 1 | Node 0 |
| 659h | Node 15 | Node 14 | Node 13 | Node 12 | Node 11 | Node 10 | Node 9 | Node 8 |
| ↓ | … | … | … | … | … | … | … | … |
| 65Fh | Node 63 | Node 62 | Node 61 | Node 60 | Node 59 | Node 58 | Node 57 | Node 56 |

***Fieldbus Node Status*** – First byte is MACID 0x00, next is 0x01, etc.

The node status field is a 64-byte long array, which tells the status of the nodes in the network. If any node is faulted, an error code will be presented here that describes the fault (if possible). The Status of the ANYBUS M module is also presented here, in the byte that corresponds to the MAC ID of the module.

**Note:** Node means a module in the network corresponding to a certain MAC ID or Node ID.

| Address | Corresponding Node |
|---|---|
| 660h | Status for Node 0 |
| 661h | Status for Node 1 |
| 662h | Status for Node 2 |
| ↓ | … |

| | |
|---|---|
| 69Fh | Status for Node 63 |

Should errors develop, specifically defined codes are as follows:

The following table describes the meaning of the values that can appear in the Node status area. Some of these parameters may only appear in the Node Status byte that corresponds to the masters MAC ID.

| Value Dec | Value Hex | Meaning | Value Dec | Value Hex | Meaning |
|---|---|---|---|---|---|
| 00 | 0x00 | OK or Not in scan list | 84 | 0x54 | Node not yet initialized |
| 70 | 0x46 | Duplicate MAC ID failure | 85 | 0x55 | Receive buffer overflow |
| 71 | 0x47 | Scanner configuration error | 86 | 0x56 | Node changed to IDLE mode |
| 72 | 0x48 | Device communication error | 87 | 0x57 | Shared master error (not used) |
| 73 | 0x49 | Wrong device type | 88 | 0x58 | Shared choice error (not used) |
| 74 | 0x4A | Port over-run error | 89 | 0x59 | Keeper object failure (not used) |
| 75 | 0x4B | Network failure | 90 | 0x5A | CAN port disabled (not used) |
| 76 | 0x4C | No CAN messages detected | 91 | 0x5B | Bus off |
| 77 | 0x4D | Wrong data size | 92 | 0x5C | No bus power detected |
| 78 | 0x4E | No such device found | 95 | 0x5F | Updating flash (not used) |
| 79 | 0x4F | Transmit failure | 96 | 0x60 | In test mode (not used) |
| 80 | 0x50 | Node in IDLE mode | 97 | 0x61 | Halted by user cmd. (not used) |
| 81 | 0x51 | Node in fault mode | 98 | 0x62 | Firmware failure (not used) |
| 82 | 0x52 | Fragmentation error | 99 | 0x63 | System failure |
| 83 | 0x53 | Unable to initialize node | | | |

CHAPTER 7

# [7] XML Configuration File & I/O Declarations

The M3-61A can be used for simple I/O scanning or it can be enhanced to support complex explicit messaging. Configuration of how the module should operate is done by setting up a special ASCII configuration file. This chapter discusses that file and how it is used for simple digital input and analog scanning. Explicit messaging will be handled in the next chapter.

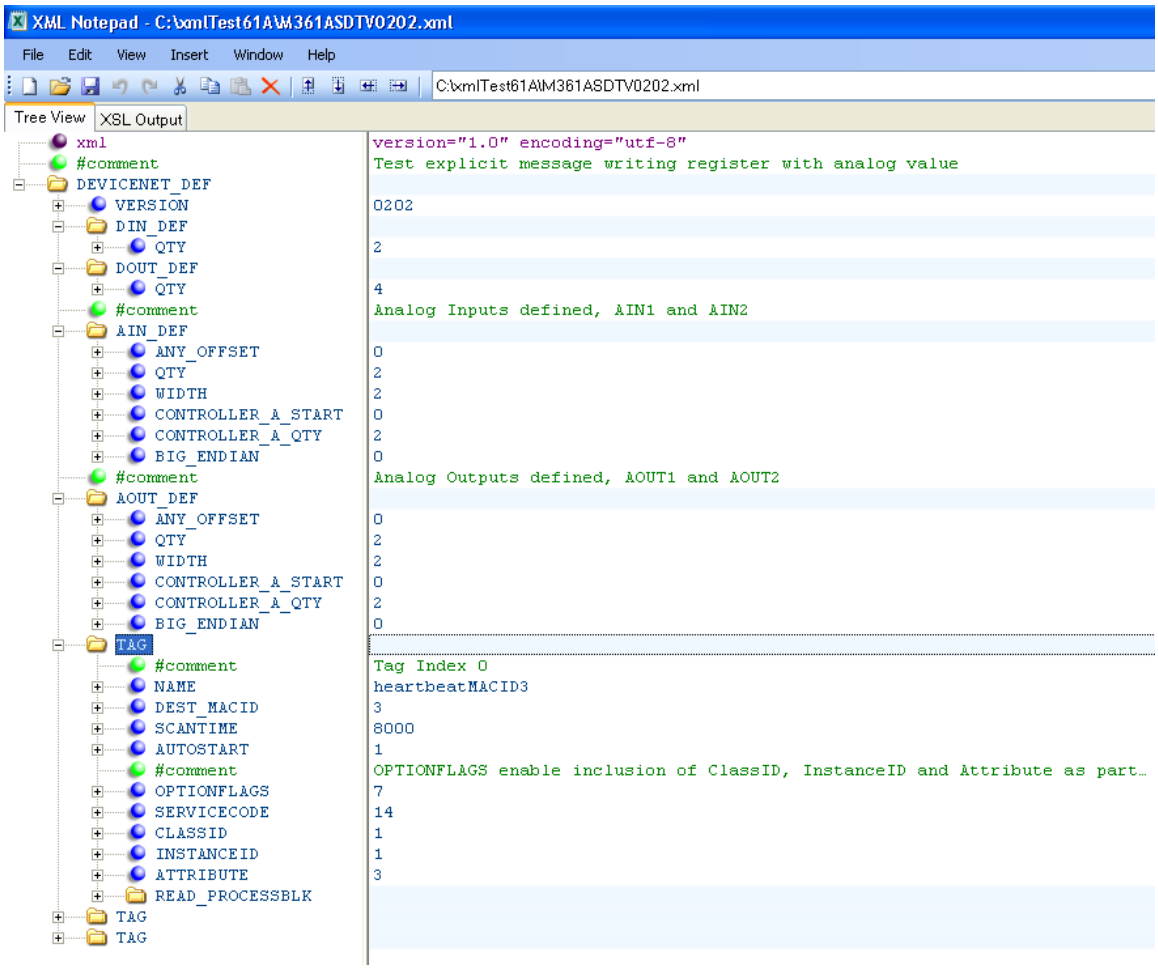
## *Configuration File*

XML is a standard industry format used for exchanging information. The Model 5300 uses this file format as a means to initialize the M3-61A for operation. When using NetTools, digital and analog data is mapped from the DeviceNet network to the Anybus dualport memory. The configuration file then takes this data and maps it so it is available to the Model 5300, thus becoming useful for your application programs to access.

Creation of the XML file can be done in a standard text editor, such as Notepad, and a template is available in the download area of our web site. A better tool to use is available free of charge from Microsoft, called XML Notepad 2007. It will do syntax checking for you as well as provide tree views, greatly simplifying data entry:

http://www.microsoft.com/downloads/details.aspx?familyid=72d6aa49-787d-4118-ba5f-4f30fe913628&displaylang=en

Use of this tool is beyond the scope of this document. A fairly complicated configuration file can be greatly simplified. Reference *Example 3* in *Chapter 9: Special Register Features*. There are multiple pages of XML code for this example, but in the presentation view of the same file in XML Notepad 2007 it is much simpler, especially if starting with a template file:



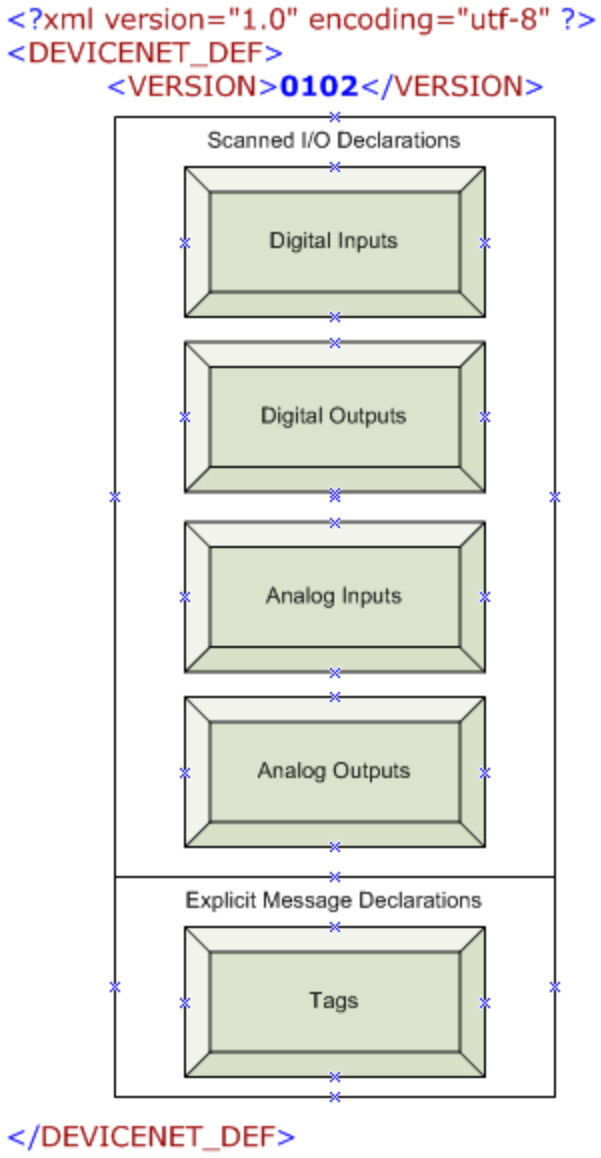

*The XML file is a standard ASCII file with each line terminated by a CR LF combination.*

## Configuration Sections

The XML starts with a couple of unique elements:

1. <?xml version="1.0" encoding="utf-8" ?> - XML declaration header, first line of the file.
2. <DEVICENET_DEF> - Start of DeviceNet definition parameters, ended by </DEVICENET_DEF>. All digital IO, analog, and tags are enclosed by these root elements.
3. <VERSION> - Defines the file version information. This is useful to ensure running the correct xml file. Use `get devicenet info` command to retrieve that set in this field. It is ended by </VERSION>. Child element to <DEVICENET_DEF>.

The configuration file is composed of a number of sections and must follow the format and sequence defined below:




***Digital Inputs*** – Up to 1024 digital inputs are available on a Model 5300. All may reside on a DeviceNet network or be mixed with local cards. These are scanned IO.

***Digital Outputs*** – Up to 1024 digital outputs are available on a Model 5300. All may reside on a DeviceNet network or be mixed with local cards. These are scanned IO.

***Analog Inputs*** – Up to 256 analog inputs are available on a Model 5300. All may reside on a DeviceNet network or be mixed with local cards. These may be scanned IO, or mixed with explicit messages, depending on the device type communicating with. Analog input data that is on a DeviceNet network is loaded into high speed dual port RAM for the Model 5300 to access, after each scan.

***Analog Outputs*** – Up to 256 analog outputs are available on a Model 5300. All may reside on a DeviceNet network or be mixed with local cards. These may be scanned IO or mixed with explicit messages, depending on the device type communicating with. Analog outputs are stored in registers local to the M3-61A after each write, generating either an automatic explicit message or an update on the next scan cycle if IO mapped.

***Tags*** – Tags define explicit messages. Their discussion is limited to the next chapter.

## Digital Input Definitions

The total number of digital inputs that were mapped with NetTools needs to be declared so the PLC knows what to expect and how many inputs to add to the standard I/O in the system. This allows multiple master cards to exist in a single PLC. There can only be 1024 total inputs for both local and networked I/O.

<QTY> item lists the number of inputs that will be made available by the DeviceNet interface. There may only be one DIN_DEF per file and it should be prior to any analog or tag declarations. Any I/O included in the later defined EXCLUDE_DIN should be removed from this total.

Example:

```
<DIN_DEF>
        <QTY>12</QTY>
</DIN_DEF>
```

## Digital Output Definitions

The total number of digital outputs that were mapped with NetTools needs to be declared so the PLC knows what to expect and how many outputs to add to the standard I/O in the system. This allows multiple master cards to exist in a single PLC. There can only be 1024 total outputs for both local and networked I/O.

<QTY> item lists the number of outputs that will be made available by the DeviceNet interface. There may only be one DOUT_DEF per file and it should be prior to any analog or tag declarations. Any I/O included in the later defined EXCLUDE_DOUT should be removed from this total.

Example:

```
<DOUT_DEF>
        <QTY>12</QTY>
</DOUT_DEF>
```

## *Analog Input Definitions*

Analog input definitions are required if you have mapped any analog inputs to the I/O area using NetTools. Some devices present analog information as bit sequences that can be scanned as part of the input data, others require direct explicit messages. Only include analogs that are scanned as I/O, not explicit messages in the total count. Explicit messages will be added separately, as defined later. An entry is required for each of the mappings done using NetTools. Additionally this section must precede any explicit message or tag definitions. Note that there are up to 512 bytes of input area available for mapping data, with both input and output areas being separate. Element definitions are as follows:

<AIN_DEF> - Header element to notify an analog input definition is to follow.

<ANY_OFFSET> - Number of bytes offset to the start of analog input data as defined in the NetTools mapping of dual port memory, starting at 0.

<QTY> - Number of consecutive analog inputs defined in NetTools for this device.

<WIDTH> - Width in bytes of each analog value being scanned from the remote device's perspective. Either 2 or 4.

<CONTROLLER_A_START> - Analog input start offset for the M3-61A card, with 0 being the first analog input, as referenced by the controller. Each input is added to the end of the total local input module. Module placement in rack does not matter.

<CONTROLLER_A_QTY> - Number of Analog inputs to be defined for controller access by this definition. Must be <= <QTY>. Typically it is the same as <QTY> but in some cases you may not want to reference all the analog inputs.

<BIG_ENDIAN> - Defines type of data that is being scanned from the remote DeviceNet device, 0 for little endian (default), 1 for big endian (not currently supported).

</AIN_DEF> - Header element to notify an analog input definition is complete.

Example (Two analog inputs which are of type short (2 bytes each) begin at a byte offset of 2 within the Anybus dualport memory as mapped by NetTools and are mapped as the first 2 analog inputs on the M3-61A card):

```
<AIN_DEF>
      <ANY_OFFSET>2</ANY_OFFSET>
      <QTY>2</QTY>
      <WIDTH>2</WIDTH>
      <CONTROLLER_A_START>0</CONTROLLER_A_START>
      <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
```

```
        <BIG_ENDIAN>0</BIG_ENDIAN>
</AIN_DEF>
```

## Analog Output Definitions

Analog output definitions are required if you have mapped any analog outputs to the I/O area using NetTools. Some devices present analog information as bit sequences that can be scanned, others require direct explicit messages. Only include analogs that are scanned as I/O, not explicit messages in the total count. An entry is required for each mapping done using NetTools. Additionally this section must precede any explicit message tag definitions. Note that there are up to 512 bytes of output area available for mapping data, with both input and output areas being separate. Element definitions are as follows:

<AOUT_DEF> - Header element to notify an analog output definition is to follow.

<ANY_OFFSET> - Number of bytes offset to the start of analog output data as defined in the NetTools mapping of dual port memory, starting at 0.

<QTY> - Number of analog outputs defined in NetTools for this device.

<WIDTH> - Width in bytes of each analog value being written, from the remote device's perspective. Either 2 or 4.

<CONTROLLER_A_START> - Analog output start offset for M3-61A card, with 0 being the first analog output, as referenced by the controller. Each output is added to the end of the total local output modules. Module placement in rack does not matter.

<CONTROLLER_A_QTY> - Number of Analog outputs to be defined for controller access by this definition. Must be <= <QTY>. Typically it is the same as <QTY> but in some cases you may not want to reference all the analog outputs.

<BIG_ENDIAN> - Defines type of data that is being scanned from the remote DeviceNet device, 0 for little endian (default), 1 for big endian (not currently supported).

</AOUT_DEF> - Header element to notify an analog output definition is complete.

Example (Two analog outputs which are of type short (2 bytes each) begin at a byte offset of 1 within the Anybus dualport output memory as mapped by NetTools and are mapped as the first 2 analog outputs on the M3-61A card):

```
<AOUT_DEF>
        <ANY_OFFSET>1</ANY_OFFSET>
        <QTY>2</QTY>
        <WIDTH>2</WIDTH>
```

```
        <CONTROLLER_A_START>0</CONTROLLER_A_START>
        <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
        <BIG_ENDIAN>0</BIG_ENDIAN>
</AOUT_DEF>
```

*Outputs start at offset 0 when using NetTools. 1 means the first byte is 8 bits of output data prior to the analog output definition.*

## Exclude Input Definitions

Many devices provide additional information over and above the standard inputs, such as status information. Typically this information is mapped as part of the digital inputs. In many cases it may be beneficial to exclude this information from the general input map, thus reducing the number of inputs appearing on DeviceNet as well as simplifying the interface. This is done using the <EXCLUDE_DIN> element, which should be the last section defined in the configuration file, after tags and I/O declaration. This allows you to offset into the buffer mapped by NetTools and mark bytes to be ignored.

<EXCLUDE_DIN> - Header element to notify an exclude input definition is to follow.

<ANY_OFFSET> - Number of bytes offset, as defined in the NetTools mapping of dual port memory, to the start of input data to be excluded.

<QTY> - Number of bytes to exclude from the input list.

</EXCLUDE_DIN> - Header element to notify an exclude input definition is complete.

Example: Wago controllers with 2 analog inputs and an 8 input digital card returns 6 bytes, with the first 4 being analog data, 5th byte digital input and the 6th status information. To avoid having the status information as part of the digital inputs, the following would be used, assuming first module defined with NetTools:

```
<EXCLUDE_DIN>
        <ANY_OFFSET>5</ANY_OFFSET >
        <QTY>1</QTY>
</EXCLUDE_DIN>
```

## Exclude Output Definitions

Many devices provide additional outputs over and above those that may want to be used and mapped into the general IO space. As with the digital inputs, there is a way to exclude this byte area from definitions. This is done using the <EXCLUDE_DOUT> element, which should be the last section defined in the configuration file, after tags and I/O declaration. This allows you to offset into the buffer mapped by NetTools and mark bytes to be ignored.

<EXCLUDE_DOUT> - Header element to notify an exclude output definition is to follow.

<ANY_OFFSET> - Number of bytes offset, as defined in the NetTools mapping of dual port memory, to the start of output data to be excluded.

<QTY> - Number of bytes to exclude from the output list.

</EXCLUDE_DOUT> - Header element to notify an exclude output definition is complete.

Example: A device has output data defined for the first 6 bytes, thus causing 48 outputs to be mapped to the Model 5300 controller. The last 8 outputs are not used. Below removes them from the mapping resulting in only 40 outputs.

```
<EXCLUDE_DOUT>
        <ANY_OFFSET>5</ANY_OFFSET >
        <QTY>1</QTY>
</EXCLUDE_DOUT>
```

## Simple Digital I/O Example

Assuming the scanned devices only consist of digital inputs and outputs, a very simple XML file can be loaded. If there is only a single DeviceNet Master card you may create a universal file that basically says map all remaining digital I/O in the system to DeviceNet:

```
<?xml version="1.0" encoding="utf-8" ?>
<DEVICENET_DEF>
    <VERSION>0102</VERSION>
    <DIN_DEF>
        <QTY>1024</QTY>
    </DIN_DEF>
    <DOUT_DEF>
        <QTY>1024</QTY>
    </DOUT_DEF>
</DEVICENET_DEF>
```

If you wish to use more than one card simply set the QTY elements to their actual values. This will allow each of the cards to know what I/O they are responsible for. It is generally good practice to set the I/O to the actual total so the controller will fault if I/O that does not exist is written to.

## Simple Analog I/O Example

Some devices have the ability to provide both digital and analog information in the scanned data. One such controller is the Wago 750-306 DeviceNet Slave. When configuring this device a simple example is:

```
<?xml version="1.0" encoding="utf-8"?>
<DEVICENET_DEF>
    <VERSION>0203</VERSION>
    <DIN_DEF>
```

```
        <QTY>2</QTY>
    </DIN_DEF>
    <DOUT_DEF>
        <QTY>4</QTY>
    </DOUT_DEF>
    <!-- Analog Inputs defined, AIN1 and AIN2 -->
    <AIN_DEF>
        <ANY_OFFSET>0</ANY_OFFSET>
        <QTY>2</QTY>
        <WIDTH>2</WIDTH>
        <CONTROLLER_A_START>0</CONTROLLER_A_START>
        <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
        <BIG_ENDIAN>0</BIG_ENDIAN>
    </AIN_DEF>
    <!-- Analog Outputs defined, AOUT1 and AOUT2 -->
    <AOUT_DEF>
        <ANY_OFFSET>0</ANY_OFFSET>
        <QTY>2</QTY>
        <WIDTH>2</WIDTH>
        <CONTROLLER_A_START>0</CONTROLLER_A_START>
        <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
        <BIG_ENDIAN>0</BIG_ENDIAN>
    </AOUT_DEF>
</DEVICENET_DEF>
```

The above example is for 2 digital inputs, 4 digital outputs, 2 analog inputs and 2 analog outputs. All these I/O would appear as though they were local to the Model 5300 application program and the normal digital input/output and analog input/output register access would apply.

## *XML Configuration File Storage*

The configuration file is stored in resident serial flash memory. There are approximately 8Mbytes available for storage thus little chance of running out of space. The file is loaded into memory during Model 5300 initialization, reporting back to the main controller the I/O configuration for each module. The serial flash is loaded similar to re-flashing modules, using the telnet remote administrative screens. Assuming slot 4 for the module and an xml file called M361ASDTV0102.xml, the following is a command line example:

```
fupdate slot 4 M361ASDTV0102.xml
```

It may take up to a couple of minutes for the command to complete execution and no display feedback is given until the re-flash is completed. The file must reside in the /_system/Firmware sub-directory prior to command execution. The controller must also be reset after the file is transferred to the M3-61A for the new version to become active. Assuming the <VERSION> element was set in the XML file, you may then use the `get anybus info` command to verify proper loading.

CHAPTER 8

# [8] Explicit Messaging & Tags



The M3-61A has extensive support for explicit messaging. Messages may be set up to read and write on a one-shot or polled basis. Model 5300 registers may act as either a source or destination of explicit message data depending upon the type of message being implemented, thus providing extensive data mapping. The configuration of explicit messages is done using a special XML file format to define 'tags'. Multiple 'tags' may exist, each supplying the parameters needed to create an explicit message.

*When defining XML parameters, each XML element begins with < > and ends with </ >. Also, all data must be on the same line for each parameter.*

## Register Summary

Explicit messages can be sent and received by the M3-61A. In doing so information read from a remote device (get attribute) must be placed somewhere for program access, as well as when a write (set attribute) is done data must be sourced. There are a number of locations for data that may be mapped to/from messages. *Chapter 9: Special Register Features* details some of these to a greater degree but in summary:

***5300 Controller Register*** – All registers local to the Model 5300, including Variants, may be accessed to source or store information from an explicit message by the use of data mapping.

***High Speed Dualport Registers*** – 256 general dual port registers are shared by the controller and the M3-61A card. These are accessed directly by the M3-61A via data mapping and by the use of variant 36825 in a Model 5300 application program.

***Global Common Bits*** – The Model 5300 CPU and all 40 series motion cards support global bits that may be set or cleared. These may be accessed directly by explicit messages and used as flags for executing Motion Sequence Blocks (MSBs) resident on other modules or monitored by Quickbuilder programs.

***Global Common Vars*** – The Model 5300 CPU and all 40 series motion cards support 256 common bytes that may be written to or read. . These may be accessed directly by explicit messages and used as flags for executing MSBs resident on other modules or monitored by Quickbuilder programs

## *Tags*

An explicit message 'tag' defines all the parameters of a specific message. It is stored in an XML format and may be created in a simple text editor for download to the M3-61A card. Each 'tag' has an assigned text string name and may be invoked from Quickbuilder applications by that name (TBD) or run from special 5300 registers by referencing its index in the file (*Chapter 9: Special Register Features*). An index is simply which tag comes first, with the first defined being 0. The parameters of a tag, in XML format, are as follows:

<TAG> - Start of new Tag definition is to follow

.

<NAME> - ASCII string for high level name of Tag. Up to 32 characters and should be unique, case sensitive. For future Quickbuilder use.

<DEST_MACID> - DeviceNet MACID that explicit message is to be directed towards.

<SCANTIME> - How often, in milliseconds, that this Tag should run. Zero (0) means invoked on demand only, not polled. [Optional, defaulting to 0.]

<AUTOSTART> - By default a Tag is not run at power up or reset, it must be started by an external source such as a command over a communications channel (register write to 3032, *Chapter 9: Special Register Features*) or an external Quickstep or Quickbuilder program. To automatically start a Tag and run it based upon its SCANTIME, set this parameter to a 1. [Optional, defaulting to 0.]

<OPTIONFLAGS> - This parameter defines which explicit message items (Class ID, Instance ID and Attribute) are to be included in the message, consisting of a bit OR of Class ID (Bit 0, 0x0001), Instance ID (Bit 1, 0x0002), and Attribute (Bit 2, 0x0004). [Typically this parameter is set to decimal 7, all required.]

<SERVICECODE> - The type of explicit message to be performed such as a GET ATTRIBUTE (14, 0x0E) or a SET ATTRIBUTE (16, 0x10).

<CLASSID> - The Class ID required for the explicit message being defined. [Required if specified by OPTIONFLAGS.]

<INSTANCEID> - The Instance ID required for the explicit message being defined. [Required if specified by OPTIONFLAGS.]

<ATTRIBUTE> - The Attribute required for the explicit message being defined. [Required if specified by OPTIONFLAGS.]

<WRITE_PROCESSBLK> - The WRITE PROCESS BLOCK lists where to get each item which is to be written by an explicit message. Since a message may contain multiple parameters, each individual entry is listed as multiple <WRITE_ENTRY> items. Each item maps the written data to some optional source, like a data register or analog output. [Only one per tag.]

<MSGBLOCK> - The MSGBLOCK is only required with a WRITE_PROCESSBLK and only one is defined. It consists of two characters, which represent a hex value, thus 00 is really 0x00 and 0102 is 0x01 and 0x02. It is used to both define the length of data to be written during an explicit message and to pre-fill the message buffer with any desired character. It may then be overlaid, as desired with data sourced from other locations, such as a register. In some cases it may simply remain static to write a constant. In most cases it will simple be filled with 00's as a space filler. Reference the Analog Explicit Message Tag Example section.

<READ_PROCESSBLK> - The Read Process Block lists where to store each item which is to be read by an explicit message. Since a message may contain multiple parameters each individual entry is listed as multiple <READ_ENTRY> items. Each item maps the read data to some optional source, like a data register or analog input. [Only one per tag.]

<WRITE_ENTRY> - Multiple WRITE ENTRY items define the mapping of message items, one for one. Write entries define where to get data that is to be written by an explicit message.

<READ_ENTRY> - Multiple READ ENTRY items define the mapping of message items, one for one. Read entries define where to put data after it is read by an explicit message.

<OFFSET> - Byte offset into explicit message data.

<LENGTH> - Length of data desired at OFFSET into message.

<FLAGS> - Data type in explicit message. 0 for little endian (intel format) or 1 for big endian. [Default is 0 for little endian, big endian is not currently supported.]

<LOCATION> - On READ ENTRY, where to store data once read. On WRITE ENTRY, where to obtain data prior to write. Defined as follows:

- 0 – NOOP: Do nothing, no operation on read. On WRITE_ENTRY the MSGBLOCK will be treated as constant

data and written as it exists. Can be used to initialize one-time data setups with static information.

- 32768 – Register Access: read/write any Model 5300 register, including Variants. REGNUM defines the register to access. INDEXROW is a variant row, INDEXCOL is a variant column. TYPE defines the data type:
  VARIANT_INTEGER BIT0 (1, 0x0001)
  VARIANT_STRING BIT2 (4, 0x0004)
  VARIANT_FLOAT BIT3 (16, 0x0008)
  VARIANT_DOUBLE BIT4 (32, 0x0010)
  VARIANT_LLONGINTEGER (256, 0x0100)
- 65536 – Global Common Bits: Get/set bit in REGNUM based upon INDEXROW mask. On a write operation the global common bit is read and if true the byte referenced by OFFSET into message is OR'ed with INDEXROW, else bits in INDEXROW are cleared. On a read operation the read data is masked with INDEXROW and the global common bit referenced by REGNUM is either set or cleared depending upon mask result. True sets bit. LENGTH may be 1 or 2.
- 131072 – Global Comman Vars: Get/set byte in REGNUM. On a write operation the global common variable byte is read and inserted into the explicit message at OFFSET into the message. On a read operation the global common variable is written with that read by the explicit message using data at OFFSET into the message.
- 524288 – Dualport Register: Read/write local dualport register. Each card has 128 local registers which can store integers and floats. They are referenced via Variant register 36825 where the row is the M3-61A card (0 for first) and column is the desired register, 0 to n. REGNUM references the register of interest, 1 being the first. OFFSET, offsets into the explicit message block. LENGTH defines number of bytes to move from message on read or to it on write. TYPE defines the data type:
  VARIANT_INTEGER BIT0 (1, 0x0001)
  VARIANT_FLOAT BIT3 (16, 0x0008)
- 1048578 – Analog Input where REGNUM is the local analog input, first being 0. LENGTH is number of bytes to move in or out of explicit message buffer. INDEXROW is number of consecutive analog inputs. INDEXCOL is width of analog data, 2 or 4 bytes, in explicit message. OFFSET is number of bytes to start in explicit message, incremented by INDEXCOL for each analog input. This reference adds an analog input to those available and reported by the card.
- 2097152 – Analog Output where REGNUM is the local analog output, first being 0. LENGTH is number of bytes to

move in or out of explicit message buffer. INDEXROW is number of consecutive analog outpus. INDEXCOL is width of analog data, 2 or 4 bytes, in explicit message. OFFSET is number of bytes to start in explicit message, incremented by INDEXCOL for each analog output. This reference adds an analog outputs to those available and reported by the card.

<TYPE> - Data Type referencing:
VARIANT_INTEGER BIT0 (1, 0x0001)
VARIANT_STRING BIT2 (4, 0x0004)
VARIANT_FLOAT BIT3 (16, 0x0008)
VARIANT_DOUBLE BIT4 (32, 0x0010)
VARIANT_LLONGINTEGER (256, 0x0100)

<REGNUM> - Item referencing, refer to <LOCATION> parameter for specifics.

<INDEXROW> - Index based upon <LOCATION> selection.

<INDEXCOL> - Index based upon <LOCATION> selection.

</TAG> - Tag terminator, definition complete.

 *No more than 64 explicit messages may be outstanding (awaiting response) on the network at one time. To help prevent this, a request for a scanned message when > 40 messages are currently outstanding will be delayed.*

## Explicit Message Heartbeat

When the first explicit message is initiated to a remote node, it can take up to 2 seconds for the connection to take place. This is due to the fact that when the master tries to send the explicit message, it first needs to allocate some kind of connection for this, so it first tries with UCMM (Unconnected Message Manager) and not all devices support this. If UCMM fails it falls back to using the commands associated with 'group 2' devices (Wago 750-306 is such a device). The sequence is something like:

- Receive the command from the application.
- Make one attempt with UCMM.
- 1s timeout since this will not work with a 'group 2' device.
- Make a second attempt with UCMM.
- 1s timeout again since this will not work with a 'group 2' device.
- Fall back to using the command set for a 'group 2' device which now will reply quickly.
- Relay the answer to the application.

Once the connection is obtained it will be maintained for approximately 10 seconds. During this period additional messages will occur very fast, typically less than 20 milliseconds depending on the response of the remote node. For explicit messages that occur randomly, directed towards devices that do not support UCMM, you must ensure that they repeat in less than 10 seconds, or a message delay of 2 seconds could possibly occur. If this cannot be tolerated a simple solution is to create a heart beat tag. This is simply an explicit message that is sent periodically and reads the device product code (Get Attribute), doing nothing with the data. An example of a heartbeat tag would be:

```
<TAG>
    <NAME>heartbeatMACID3</NAME>
    <DEST_MACID>3</DEST_MACID>
    <!—Run tag every 8 seconds, timeout is around 10 on connection -->
    <SCANTIME>8000</SCANTIME>
    <!—Automatically start at boot  -->
    <AUTOSTART>1</AUTOSTART>
    <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
        of the MSGBLOCK.  They will precede the block.
         #define EXPLICIT_CLASSID 0x0001
         #define EXPLICIT_INSTANCEID 0x0002
         #define EXPLICIT_ATTRIBUTE 0x0004
    -->
    <OPTIONFLAGS>7</OPTIONFLAGS>
    <!-- Explicit message for keeping connection open -->
    <SERVICECODE>14</SERVICECODE>
    <CLASSID>1</CLASSID>
    <INSTANCEID>1</INSTANCEID>
    <ATTRIBUTE>3</ATTRIBUTE>
    <READ_PROCESSBLK>
        <!—Data mapping for keeping connection open, ignore data -->
        <READ_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>0</OFFSET>
            <!-- Length of data interested in, returned by remote device in data block -->
            <LENGTH>1</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to store the data once read, in this case none -->
            <LOCATION>0</LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Desired register to write to, none in this case -->
            <REGNUM>0</REGNUM>
            <!-- Variant row -->
            <INDEXROW>0</INDEXROW>
            <!-- Variant column -->
            <INDEXCOL>0</INDEXCOL>
        </READ_ENTRY>
    </READ_PROCESSBLK>
</TAG>
```

## *Analog Explicit Message Tag Example*

Some remote devices, such as Wago, allow analog inputs and outputs to be defined as scanned I/O as well as using explicit messages. The example below first defines the analog inputs of a Wago 750-467 card as the first two scanned but then also defines the same analog inputs as explicit messages. Thus when referenced from a Quickstep program the analogs would appear as follows:

First Controller Analog input – First Wago Analog Input via scanned I/O.
Second Control Analog input – Second Wago Analog Input via scanned I/O.
Third Controller Analog input – First Wago Analog Input via explicit msg.
Fourth Controller Analog input – Second Wago Analog Input via explicit msg.

The controller would also report 4 analog inputs are handled by the DeviceNet Master module. The explicit message data returned on the network would appear as follows:

| Byte | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| 0 | low byte channel 1 | | | | | | | |
| 1 | high byte channel 1 | | | | | | | |
| 2 | low byte channel 2 | | | | | | | |
| 3 | high byte channel 2 | | | | | | | |

A class 4 (assembly), instance 9, attribute 3, Get Attribute explicit message is used:

| Attribute ID | Used in buscoupler | Access rule | Name | Data type | Description | Value |
|---|---|---|---|---|---|---|
| 3 | dep. on kind of connected modules | get | Process image | Array of Byte | process image, collection of all analog modules process input data | |

The tag definitions are as follows:

```
<!-- Define the analog inputs available via scanned IO and make them the first 2 available to the controller -->
<AIN_DEF>
    <ANY_OFFSET>0</ANY_OFFSET>
    <QTY>2</QTY>
    <WIDTH>2</WIDTH>
    <CONTROLLER_A_START>0</CONTROLLER_A_START>
    <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
    <BIG_ENDIAN>0</BIG_ENDIAN>
</AIN_DEF>
<TAG>
    <!-- Define the first analog input as an explicit message and scan it every 100 milliseconds -->
    <NAME>analogInput3_4</NAME>
    <DEST_MACID>3</DEST_MACID>
    <SCANTIME>100</SCANTIME>
    <AUTOSTART>1</AUTOSTART>
    <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
        of the MSGBLOCK.  They will precede the block.
         #define EXPLICIT_CLASSID 0x0001
         #define EXPLICIT_INSTANCEID 0x0002
         #define EXPLICIT_ATTRIBUTE 0x0004
    -->
    <OPTIONFLAGS>7</OPTIONFLAGS>
    <SERVICECODE>14</SERVICECODE>
    <CLASSID>4</CLASSID>
    <INSTANCEID>9</INSTANCEID>
    <ATTRIBUTE>3</ATTRIBUTE>
    <READ_PROCESSBLK>
        <!—Data mapping for Analog inputs, AIN1 -->
```

```
        <READ_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>0</OFFSET>
            <!-- Length of data interested in, not used for analog -->
            <LENGTH>2</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to store the data once read, in this case Analog -->
            <LOCATION>1048576</LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Local analog start offset, 0 is first -->
            <REGNUM>0</REGNUM>
            <!-- Number of consecutive Analogs -->
            <INDEXROW>1</INDEXROW>
            <!-- Data width in returned packet, 2 or 4 -->
            <INDEXCOL>2</INDEXCOL>
        </READ_ENTRY>
        <!—Data mapping for Analog inputs, AIN2 -->
        <READ_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>2</OFFSET>
            <!-- Length of data interested in, not used for analog -->
            <LENGTH>2</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to store the data once read, in this case Analog -->
            <LOCATION>1048576</LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Local analog start offset, 0 is first -->
            <REGNUM>1</REGNUM>
            <!-- Number of consecutive Analogs -->
            <INDEXROW>1</INDEXROW>
            <!-- Data width in returned packet, 2 or 4 -->
            <INDEXCOL>2</INDEXCOL>
        </READ_ENTRY>
    </READ_PROCESSBLK>
    </TAG>
```

*An alternative to the above example would be to eliminate the second READ_ENTRY and simply increase the INDEX_ROW of the first one to 2. This only applies when items are sequential.*

The above is an example of a Get Attribute and its mapping. Below is an example of defining an analog output and a Set Attribute command. In this case the Wago has a 2 channel output module and the explicit message requires that when a write occurs both analog outputs be written in the same message.

| Byte | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| 0 | low byte channel 1 | | | | | | | |
| 1 | high byte channel 1 | | | | | | | |
| 2 | low byte channel 2 | | | | | | | |
| 3 | high byte channel 2 | | | | | | | |

A class 4 (assembly), instance 3, attribute 3, explicit message is used:

| Attribute ID | Used in buscoupler | Access rule | Name | Data type | Description | Value |
|---|---|---|---|---|---|---|
| 3 | dep. on kind of connected modules | get/set | Process image | Array of Byte | process image, collection of all analog modules process output data. | |

This tag maps data from the first two analog output registers to the explicit message sent to the Wago and executes whenever the Model 5300 controller Analog Output 1 or 2 are written to (assuming no local analog output modules):

```
<TAG>
    <NAME>analogOutput1_2</NAME>
    <DEST_MACID>3</DEST_MACID>
    <SCANTIME>0</SCANTIME>
    <AUTOSTART>0</AUTOSTART>
    <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
        of the MSGBLOCK.  They will precede the block.
         #define EXPLICIT_CLASSID 0x0001
         #define EXPLICIT_INSTANCEID 0x0002
         #define EXPLICIT_ATTRIBUTE 0x0004
    -->
    <OPTIONFLAGS>7</OPTIONFLAGS>
    <SERVICECODE>16</SERVICECODE>
    <CLASSID>4</CLASSID>
    <INSTANCEID>3</INSTANCEID>
    <ATTRIBUTE>3</ATTRIBUTE>
    <WRITE_PROCESSBLK>
        <!-- Explicit message data filler required for Analog output, AOUT1/2, nulls, overlaid by actual -->
        <MSGBLOCK>00000000</MSGBLOCK>
        <WRITE_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>0</OFFSET>
            <!-- Length of data interested in, not used for analog, but set to size of integer for consistency -->
            <LENGTH>4</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to get  the data for the write, in this case local Analog -->
            <LOCATION>2097152</LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Local analog start offset, 0 is first -->
            <REGNUM>0</REGNUM>
            <!-- Number of consecutive Analogs -->
            <INDEXROW>2</INDEXROW>
            <!-- Data width required by explicit message for each analog output -->
            <INDEXCOL>2</INDEXCOL>
        </WRITE_ENTRY>
    </WRITE_PROCESSBLK>
</TAG>
```

CHAPTER 9

# [9] Special Register Features



A number of special registers are available to support the M3-61A. These consist of tag execution control as well as general storage for shared DeviceNet remote mapped data.

## *Tag Execution Registers*

Explicit message tags may be run automatically from the configuration file in a scanned mode but at times there is a need to run a message on demand. There may also be the need to begin a scan and to stop it. Thus the following registers are made available:

***3031*** – Anybus Module Selection Register, with the first module being 0, default.
***3032*** – Tag Execution Register, writing a tag index value to this register will cause its defined explicit message to be queued for transmission. The first tag in the configuration file is considered index 0. Upon writing a tag index the result register, 3033, will display a –1, followed by either a 0 for success or an error code as defined below:

0 – Message successfully queued for transmission.
66 – Tag does not exist.
45 – Out of memory.

The above allows only access to the queue results of a message being sent. For diagnostic purposes it may be desirable to access the results returned by the DeviceNet network for the last explicit message sent by a tag. This may be obtained for any tag by setting bit 31 (0x80000000) of the tag desired. Thus to inspect the status of the tag at index 1 you would write a 0x80000001 (2147483649) to register 3032.

The status that will appear in register 3033 will be 0 if the tag is not executing or is successful. Alternatively the ODVA result code returned by the network will appear. It is represented by 2 bytes and referenced in Appendix B, Status Codes of The CIP

Networks Library, Volume 1, Common Industrial Protocol (CIP). A few of the most common codes are listed below:

(First byte is general status, second byte is extended status and always 0xff, thus appearing as 0xff?? in the register.)

0x01 – Connection failure.
0x02 – Resource unavailable.
0x03 – Invalid parameter value, also 0x20.
0x07 – Connection lost.
0x08 – Service not supported.
0x09 – Invalid attribute value.
0x0a – Attribute list error.
0x0e – Attribute not settable.
0x10 – Device state conflict.
0x11 – Reply data too large.
0x13 – Not enough data.
0x14 – Attribute not supported
0x15 – Too much data.
0x1f – Vendor specific error, the high order byte, 0xff will reflect the error.
0x20 – Invalid parameter.

***3033*** – Explicit Message Result Register. This register is used by registers 3032 and 3034 for command results display.

***3034*** – Tag Stop Execution Register. To stop a tag from executing that is currently in scan mode write the index number of that tag to this register. The results will appear in register 3033.

0 – Message successfully queued for transmission.
66 – Tag does not exist.

## *High Speed Dualport Registers*

In addition to the main Model 5300 registers, special high speed registers that are accessible to both the main Model 5300 CPU and the DeviceNet card are available to map explicit message data being read or written. Mapping to a controller register involves RPC (remote procedure calls). In other words, the DeviceNet program interrupts the main Model 5300 processor and makes a call to software on the main CPU, passing data packets across the backplane. For large amounts of data flow or high update rates this can slow the system down. A better method is to use the 256 shared register area, Variant array 36825. This is a two dimensional array with the row reflecting the module number (0 = first), and the column being the register number (0 = first). Thus 36825[0][0] would be the first special register on the first DeviceNet module, 36825[0][1] would be the second, etc.

The declared data type used in the configuration file, VARIANT_INTEGER or VARIANT_FLOAT must match any declaration in Quickbuilder. Quickstep is limited to VARIANT_INTEGER. In most DeviceNet applications only integer will be used.

***Example 1:*** Example of an xml tag definition to read the first analog input from a Wago controller (750-306) and place it in local register 6. This uses a class 4, instance 9, and attribute 3, get attribute explicit message:

**Instance 3:**

| Attribute ID | Used in buscoupler | Access rule | Name | Data type | Description | Value |
|---|---|---|---|---|---|---|
| 3 | dep. on kind of connected modules | get/set | Process image | Array of Byte | process image, collection of all analog modules process output data. | |

```
<TAG>
    <NAME>registerLocal6</NAME>
    <DEST_MACID>3</DEST_MACID>
    <SCANTIME>0</SCANTIME>
    <AUTOSTART>0</AUTOSTART>
    <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
        of the MSGBLOCK.  They will precede the block.
         #define EXPLICIT_CLASSID 0x0001
         #define EXPLICIT_INSTANCEID 0x0002
         #define EXPLICIT_ATTRIBUTE 0x0004
    -->
    <OPTIONFLAGS>7</OPTIONFLAGS>
    <SERVICECODE>14</SERVICECODE>
    <CLASSID>4</CLASSID>
    <INSTANCEID>9</INSTANCEID>
    <ATTRIBUTE>3</ATTRIBUTE>
    <READ_PROCESSBLK>
        <!—Data mapping for collecting data for local dualport register 6, where 0 is the first -->
        <READ_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>0</OFFSET>
            <!-- Length of data interested in, returned by remote device in data block -->
            <LENGTH>2</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to store the data once read, in this case Dualport local register -->
            <LOCATION>524288 </LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Desired register to write to -->
            <REGNUM>6</REGNUM>
            <!-- Variant row, not used -->
            <INDEXROW>0</INDEXROW>
            <!-- Variant column, not used -->
            <INDEXCOL>0</INDEXCOL>
        </READ_ENTRY>
    </READ_PROCESSBLK>
</TAG>
```

***Example 2:*** Example of an xml tag definition to read the first and second analog input from a Wago controller (750-306) and place it in local register 6 and 7. The Wago

returns an array of all analogs in the single explicit message, in this case a 2 channel 750-467 analog input card is installed, register 6 will receive analog input 1, register 7 analog input 2, after successful execution:

```
<TAG>
    <NAME>registerLocal6_7</NAME>
    <DEST_MACID>3</DEST_MACID>
    <SCANTIME>0</SCANTIME>
    <AUTOSTART>0</AUTOSTART>
    <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
        of the MSGBLOCK.  They will precede the block.
         #define EXPLICIT_CLASSID 0x0001
         #define EXPLICIT_INSTANCEID 0x0002
         #define EXPLICIT_ATTRIBUTE 0x0004
    -->
    <OPTIONFLAGS>7</OPTIONFLAGS>
    <SERVICECODE>14</SERVICECODE>
    <CLASSID>4</CLASSID>
    <INSTANCEID>9</INSTANCEID>
    <ATTRIBUTE>3</ATTRIBUTE>
    <READ_PROCESSBLK>
        <!—Data mapping for collecting data for local dualport register 6, where 0 is the first -->
        <READ_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>0</OFFSET>
            <!-- Length of data interested in, returned by remote device in data block -->
            <LENGTH>2</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to store the data once read, in this case Dualport local register -->
            <LOCATION>524288 </LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Desired register to write to -->
            <REGNUM>6</REGNUM>
            <!-- Variant row, not used -->
            <INDEXROW>0</INDEXROW>
            <!-- Variant column, not used -->
            <INDEXCOL>0</INDEXCOL>
        </READ_ENTRY>
        <!—Data mapping for collecting data for local dualport register 7, where 0 is the first -->
        <READ_ENTRY>
            <!-- Byte offset into message -->
            <OFFSET>2</OFFSET>
            <!-- Length of data interested in, returned by remote device in data block -->
            <LENGTH>2</LENGTH>
            <!-- Little Endian format returned from device -->
            <FLAGS>0</FLAGS>
            <!-- Define where to store the data once read, in this case Dualport local register -->
            <LOCATION>524288 </LOCATION>
            <!-- VARIANT_INTEGER, not used but is type -->
            <TYPE>1</TYPE>
            <!-- Desired register to write to -->
            <REGNUM>7</REGNUM>
            <!-- Variant row, not used -->
            <INDEXROW>0</INDEXROW>
            <!-- Variant column, not used -->
            <INDEXCOL>0</INDEXCOL>
        </READ_ENTRY>
    </READ_PROCESSBLK>
</TAG>
```

**Example 3:** Example of an xml tag definition to read Model 5300 register 125 and write it to a storage location in the Wago controller (750-306), tag 1. Additionally tag 2 will read that same location and place it in the Model 5300 register 126, for verification. Each is invoked by writing the tag number to register 3032. This uses a class 100, instance 1, and attribute 2, get attribute explicit message. Below is the full file listing, which assumes the Wago was defined with NetTools with 2 digital inputs, 4 digital outputs, 2 analog inputs, and 2 analog outputs, all treated as scanned I/O.

## Instance 1:

| Attribute ID | Used in buscoupler | Access rule | Name | Data type | Description |
|---|---|---|---|---|---|
| 1 | specific | get/set | Bk_Module No | USINT | module number: 0-Coupler, 1- first module, 2-2.module |
| 2 | specific | get/set | Bk_TableNo | USINT | table number: 0 ... 256; not all existing |
| 3 | specific | get/set | Bk_Register No | USINT | Register number: 0...255 for the Coupler (0...63 for modules) |
| 4 | specific | get/set | Bk_Data | UINT | Register data , Status |
| 5 | specific | get | ProcessState | USINT | Coupler status: 0x01 module communication error, 0x02 internal bus error , 0x08: module diagnostic , 0x80 fieldbus error |
| 6 | specific | get | DNS_i_Trmnl dia (**) | UINT | Module diagnostic, 0x8000 to decode a message, High Byte (Bit14...8): channel number, Low Byte (Bit7..0) Module number |

```
<?xml version="1.0" encoding="utf-8"?>
<DEVICENET_DEF>
    <VERSION>0202</VERSION>
    <DIN_DEF>
        <QTY>2</QTY>
    </DIN_DEF>
    <DOUT_DEF>
        <QTY>4</QTY>
    </DOUT_DEF>
    <!-- Analog Inputs defined, AIN1 and AIN2 -->
    <AIN_DEF>
        <ANY_OFFSET>0</ANY_OFFSET>
        <QTY>2</QTY>
        <WIDTH>2</WIDTH>
        <CONTROLLER_A_START>0</CONTROLLER_A_START>
        <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
        <BIG_ENDIAN>0</BIG_ENDIAN>
    </AIN_DEF>
    <!-- Analog Outputs defined, AOUT1 and AOUT2 -->
    <AOUT_DEF>
        <ANY_OFFSET>0</ANY_OFFSET>
        <QTY>2</QTY>
        <WIDTH>2</WIDTH>
        <CONTROLLER_A_START>0</CONTROLLER_A_START>
        <CONTROLLER_A_QTY>2</CONTROLLER_A_QTY>
        <BIG_ENDIAN>0</BIG_ENDIAN>
    </AOUT_DEF>
    <TAG>
        <!-- Tag Index 0 -->
        <NAME>heartbeatMACID3</NAME>
        <DEST_MACID>3</DEST_MACID>
        <SCANTIME>8000</SCANTIME>
        <AUTOSTART>1</AUTOSTART>
        <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
```

```
            of the MSGBLOCK.  They will precede the block.
             #define EXPLICIT_CLASSID 0x0001
             #define EXPLICIT_INSTANCEID 0x0002
             #define EXPLICIT_ATTRIBUTE 0x0004
        -->
        <OPTIONFLAGS>7</OPTIONFLAGS>
        <SERVICECODE>14</SERVICECODE>
        <CLASSID>1</CLASSID>
        <INSTANCEID>1</INSTANCEID>
        <ATTRIBUTE>3</ATTRIBUTE>
        <READ_PROCESSBLK>
            <!-- Explicit message for keeping connection open -->
            <READ_ENTRY>
                <!-- Byte offset into message -->
                <OFFSET>0</OFFSET>
                <!-- Length of data interested in, returned by remote device in data block -->
                <LENGTH>1</LENGTH>
                <!-- Little Endian format returned from device -->
                <FLAGS>0</FLAGS>
                <!-- Define where to store the data once read, in this case none -->
                <LOCATION>0</LOCATION>
                <!-- VARIANT_INTEGER, not used but is type -->
                <TYPE>0</TYPE>
                <!-- Desired register to write to -->
                <REGNUM>0</REGNUM>
                <!-- Variant row -->
                <INDEXROW>0</INDEXROW>
                <!-- Variant column -->
                <INDEXCOL>0</INDEXCOL>
            </READ_ENTRY>
        </READ_PROCESSBLK>
    </TAG>
    <TAG>
        <!-- Tag Index 1 -->
        <NAME>Table_Num_to_registerRemote126</NAME>
        <DEST_MACID>3</DEST_MACID>
        <SCANTIME>0</SCANTIME>
        <AUTOSTART>0</AUTOSTART>
        <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
            of the MSGBLOCK.  They will precede the block.
             #define EXPLICIT_CLASSID 0x0001
             #define EXPLICIT_INSTANCEID 0x0002
             #define EXPLICIT_ATTRIBUTE 0x0004
        -->
        <OPTIONFLAGS>7</OPTIONFLAGS>
        <SERVICECODE>14</SERVICECODE>
        <CLASSID>100</CLASSID>
        <INSTANCEID>1</INSTANCEID>
        <ATTRIBUTE>2</ATTRIBUTE>
        <READ_PROCESSBLK>
            <!-- Read Coupler Configuration Object Table Num from Wago and storing it to PLC Register 126 -->
            <READ_ENTRY>
                <!-- Byte offset into message -->
                <OFFSET>0</OFFSET>
                <!-- Length of data interested in, returned by remote device in data block -->
                <LENGTH>1</LENGTH>
                <!-- Little Endian format returned from device -->
                <FLAGS>0</FLAGS>
                <!-- Define where to store the data once read, in this case register -->
                <LOCATION>32768</LOCATION>
                <!-- VARIANT_INTEGER, not used but is type -->
                <TYPE>1</TYPE>
                <!-- Desired register to write to -->
                <REGNUM>126</REGNUM>
                <!-- Variant row -->
                <INDEXROW>0</INDEXROW>
```

```
                <!-- Variant column -->
                <INDEXCOL>0</INDEXCOL>
            </READ_ENTRY>
        </READ_PROCESSBLK>
    </TAG>
    <TAG>
        <!-- Tag Index 2 -->
        <NAME>registerRemote125_to_Table_Num</NAME>
        <DEST_MACID>3</DEST_MACID>
        <SCANTIME>0</SCANTIME>
        <AUTOSTART>0</AUTOSTART>
        <!-- OPTIONFLAGS enable inclusion of ClassID, InstanceID and Attribute as part
            of the MSGBLOCK.  They will precede the block.
             #define EXPLICIT_CLASSID 0x0001
             #define EXPLICIT_INSTANCEID 0x0002
             #define EXPLICIT_ATTRIBUTE 0x0004
        -->
        <OPTIONFLAGS>7</OPTIONFLAGS>
        <SERVICECODE>16</SERVICECODE>
        <CLASSID>100</CLASSID>
        <INSTANCEID>1</INSTANCEID>
        <ATTRIBUTE>2</ATTRIBUTE>
        <WRITE_PROCESSBLK>
            <MSGBLOCK>00</MSGBLOCK>
            <!-- Writing 5300 register 125 to Wago Table Number of Coupler Configuration Object -->
            <WRITE_ENTRY>
                <!-- Byte offset into message -->
                <OFFSET>0</OFFSET>
                <!-- Length of data to move into the message from register 125 -->
                <LENGTH>1</LENGTH>
                <!-- Little Endian format returned from device -->
                <FLAGS>0</FLAGS>
                <!-- Define where to get the data from to fill in message, in this case 5300 register, 125 -->
                <LOCATION>32768 </LOCATION>
                <!-- VARIANT_INTEGER -->
                <TYPE>1</TYPE>
                <!-- Desired register to source data from -->
                <REGNUM>125</REGNUM>
                <!-- Variant row, not used since not a variant register -->
                <INDEXROW>0</INDEXROW>
                <!-- Variant column, not used since not a variant register -->
                <INDEXCOL>0</INDEXCOL>
            </WRITE_ENTRY>
        </WRITE_PROCESSBLK>
    </TAG>
</DEVICENET_DEF>
```

## *Status Registers*

Status and fault registers are available for quick program reference.

**12333** – Network Module Selection Register (read/write) – 0 if no DeviceNet or Ethercat/IP Master cards available, 1 (default) to select first card's node status to appear in 13400 – 13463 registers. Will accept entries up to the maximum number of Anybus Master cards installed. Note that Ethernet/IP Master will also count as a card if intermixed with DeviceNet Master cards.

**13464** – Online Status Register (read only) – 0 if offline or faulted, 1 if online and scanning configured nodes. Register 12333 selects which Master card to monitor. Valid for Ethernet/IP and DeviceNet Modules.
**13400 – 13463** – Node Status Registers (read only) – Represents status of MAC ID 0 to 63. 0 if offline, 1 if online and operations, else fault definition as below. Register 12333 selects which Master card to monitor. Only valid for DeviceNet Master Modules.

| Value Dec | Value Hex | Meaning | Value Dec | Value Hex | Meaning |
|---|---|---|---|---|---|
| 00 | 0x00 | Offline | 84 | 0x54 | Node not yet initialized |
| 70 | 0x46 | Duplicate MAC ID failure | 85 | 0x55 | Receive buffer overflow |
| 71 | 0x47 | Scanner configuration error | 86 | 0x56 | Node changed to IDLE mode |
| 72 | 0x48 | Device communication error | 87 | 0x57 | Shared master error (not used) |
| 73 | 0x49 | Wrong device type | 88 | 0x58 | Shared choice error (not used) |
| 74 | 0x4A | Port over-run error | 89 | 0x59 | Keeper object failure (not used) |
| 75 | 0x4B | Network failure | 90 | 0x5A | CAN port disabled (not used) |
| 76 | 0x4C | No CAN messages detected | 91 | 0x5B | Bus off |
| 77 | 0x4D | Wrong data size | 92 | 0x5C | No bus power detected |
| 78 | 0x4E | No such device found | 95 | 0x5F | Updating flash (not used) |
| 79 | 0x4F | Transmit failure | 96 | 0x60 | In test mode (not used) |
| 80 | 0x50 | Node in IDLE mode | 97 | 0x61 | Halted by user cmd. (not used) |
| 81 | 0x51 | Node in fault mode | 98 | 0x62 | Firmware failure (not used) |
| 82 | 0x52 | Fragmentation error | 99 | 0x63 | System failure |
| 83 | 0x53 | Unable to initialize node | 01 | 0x01 | Online and operational |

*Blank*

Appendix A

# [A] Additional NetTools Examples

This section contains a few configuration examples of different devices that may be connected to the M3-61A, in addition to the examples in Chapter 6.

## *Control Technology Corporation Model 5300 DeviceNet Slave*

The CTC Model 5300 can be a Master and/or a Slave. In a Slave configuration all local digital and analog IO are available to the DeviceNet network. The example to follow consists of the following configuration:

**M3-20A** – Slot 1, 16 input, 16 output digital IO module. 2 bytes produced, 2 bytes consumed.
**M3-20A** – Slot 2, 16 input, 16 output digital IO module. 2 bytes produced, 2 bytes consumed.
**M3-31A** – Slot 3, 16 input, +/-10V analog input module. 64 bytes produced.
**M3-32B** – Slot 4, 16 output, +/-10V analog output module. 64 bytes consumed.
**M3-61B** – Slot 5/6, DeviceNet Slave module.

The produced/consumed data presented to NetTools consists of digital first, followed by analog data:

RX bytes, produced:
struct
{
    unsigned char digitalIn[Number of digital inputs / 8];
    int analogIn[Number of analog inputs];
} IOInputs;

TX bytes, consumed:
struct
{
    unsigned char digitalOut[Number of digital outputs / 8];
    int analogOut[Number of analog outputs];
} IOOutputs;

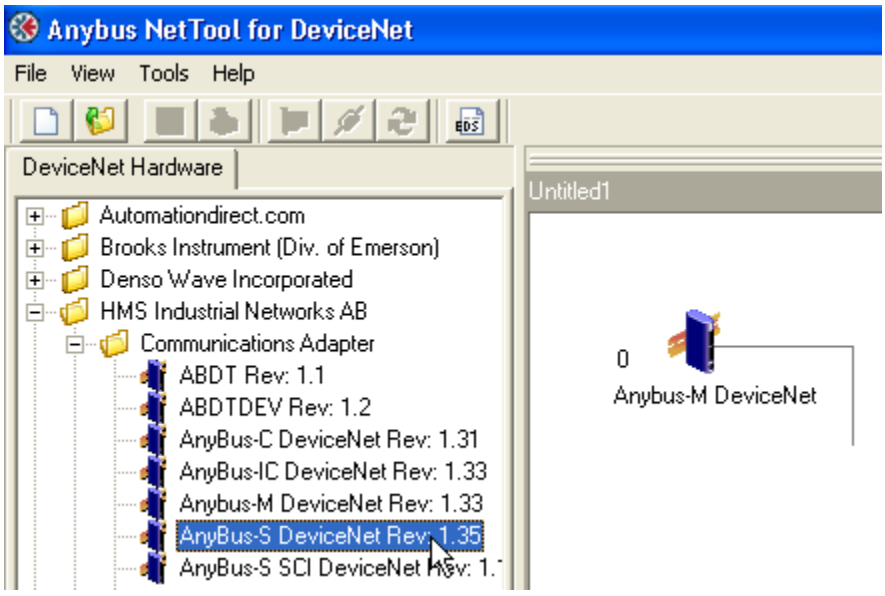
In summary the proper RX/TX polled bytes can be derived adding the bytes available for each module and/or confirmed using NetTools and referencing the Model 5300 controller Anybus-S parameter area, as shown in the example.

The standard Anybus-S DeviceNet Slave EDS file (72-7255-EDS files.zip, EDS_ABS_DEV_V_2_3.eds) is used for configuration and is available for download at both the CTC and anybus websites:

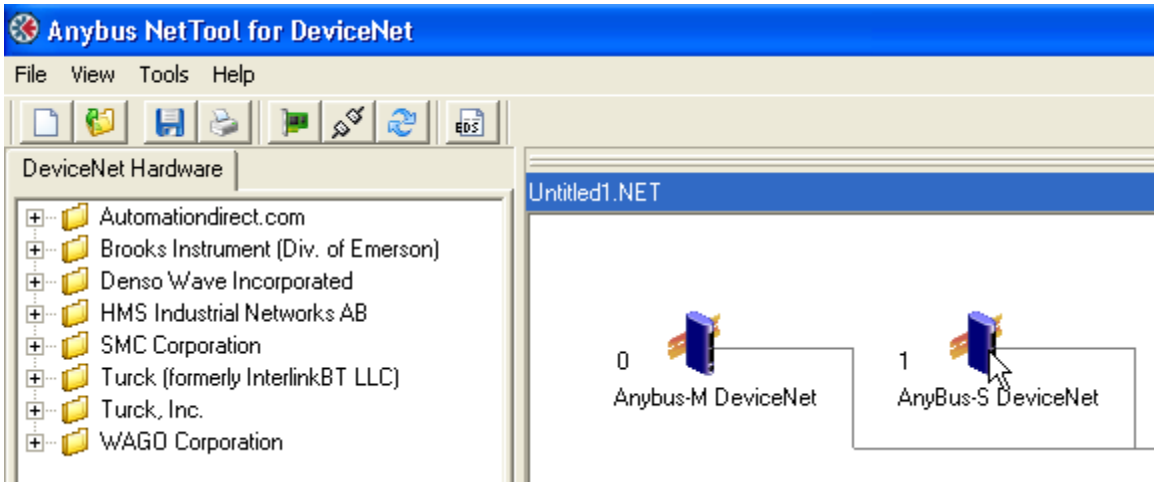
http://www.anybus.com/support/support.asp?PID=72&ProductType=Anybus-S

## Example of a CTC Model 5300 Slave Configuration

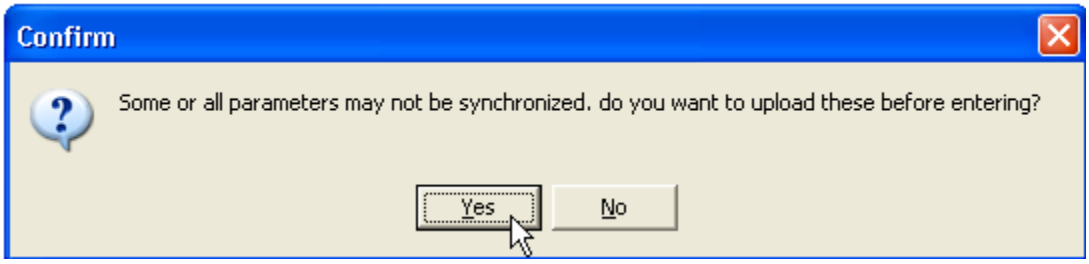
With the EDS file installed either go online and scan the network or add the AnyBus-S DeviceNet module manually by dragging and dropping the selection and setting the MACID desired:



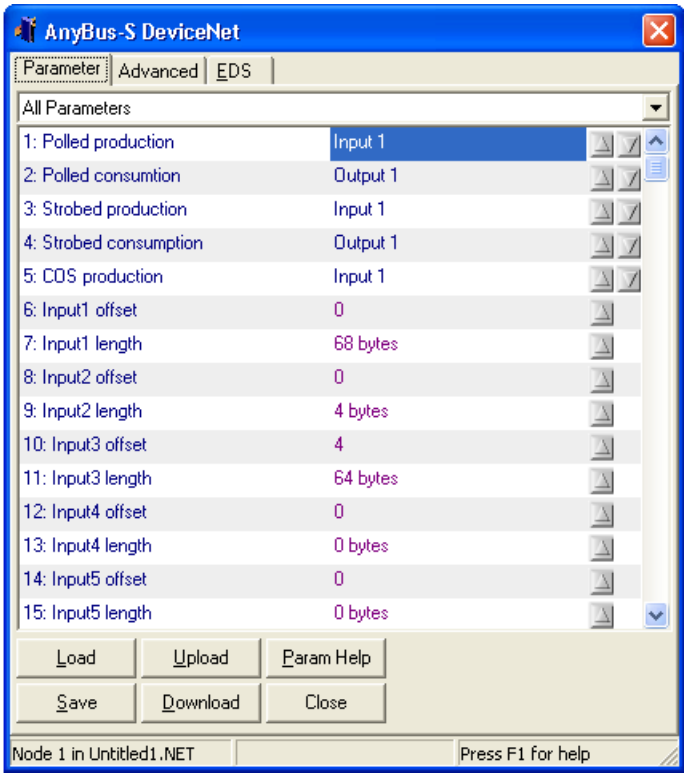

Double clicking from within NetTools the AnyBus-S DeviceNet Icon will allow you to verify the proper produced and consumed byte count (assuming online):



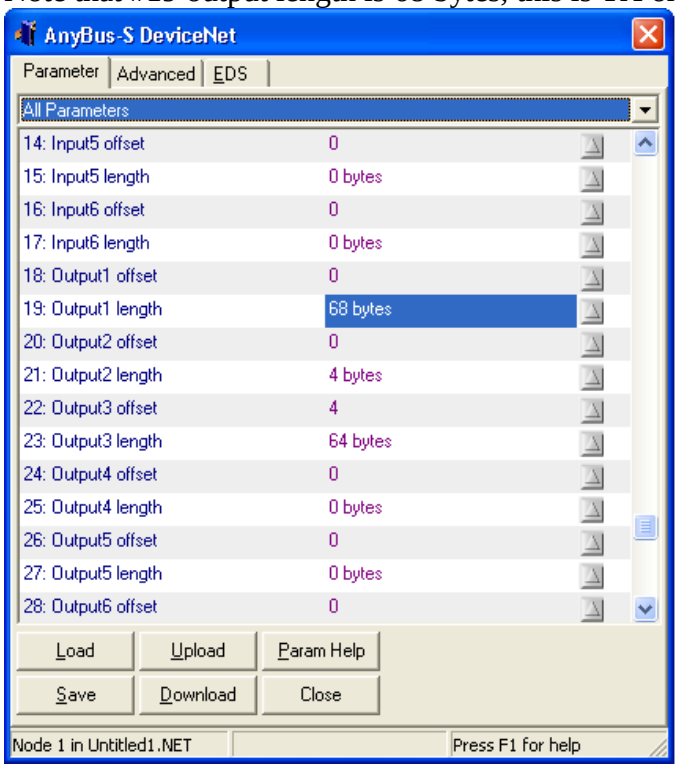

Synchronize with the device:



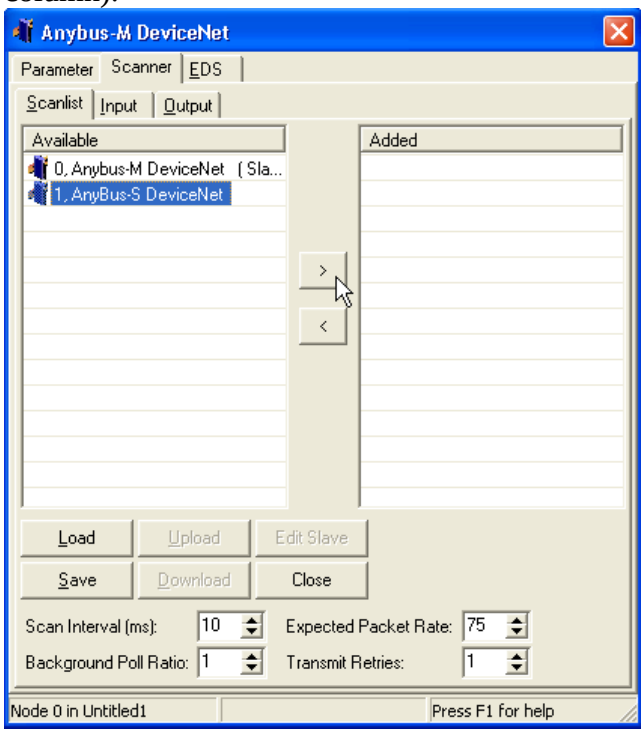

Reference 1 (polled production) and 2 (polled consumption). Note that #7 input 1 length is 68 bytes, this is RX or produced data:



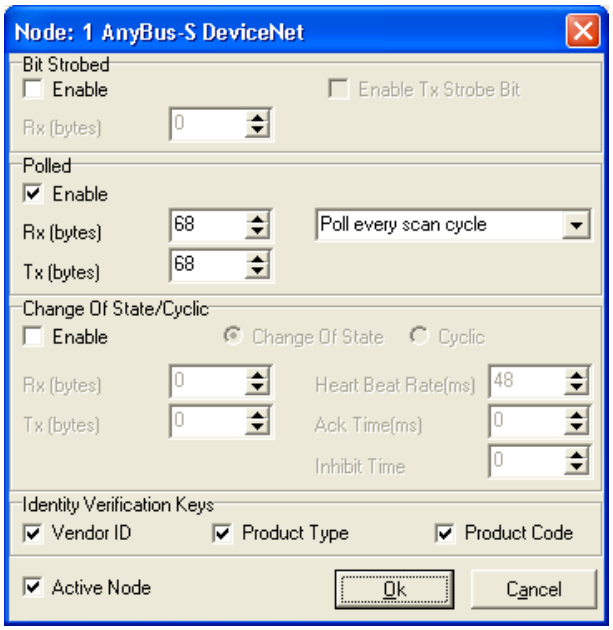
Note that #19 output length is 68 bytes, this is TX or consumed data:



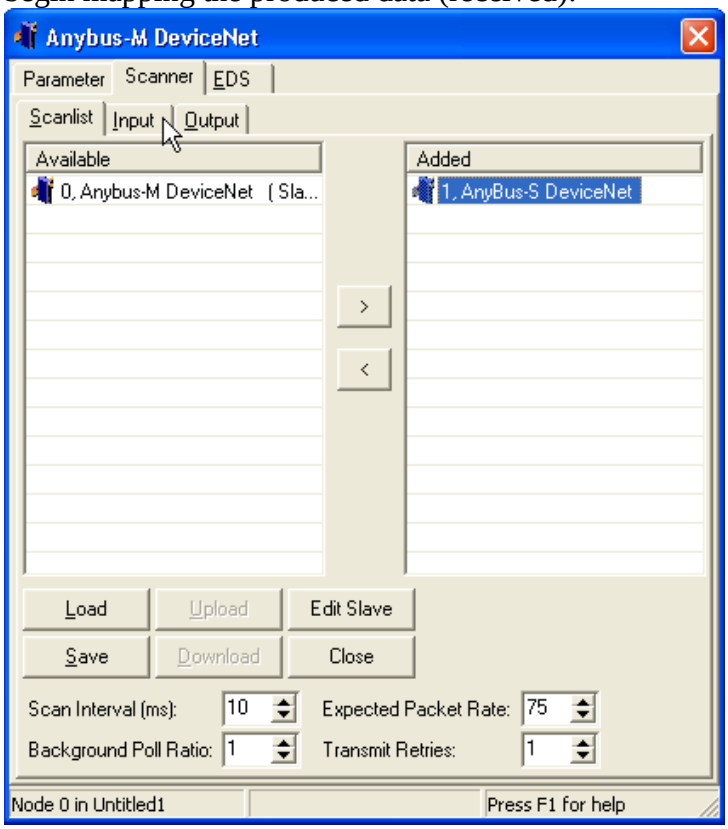

Close the Slave dialog box and double click the Anybus-M icon to begin configuration, selecting the 'Scanlist' tab and then move the Anybus-S to the active scan list (Added column):



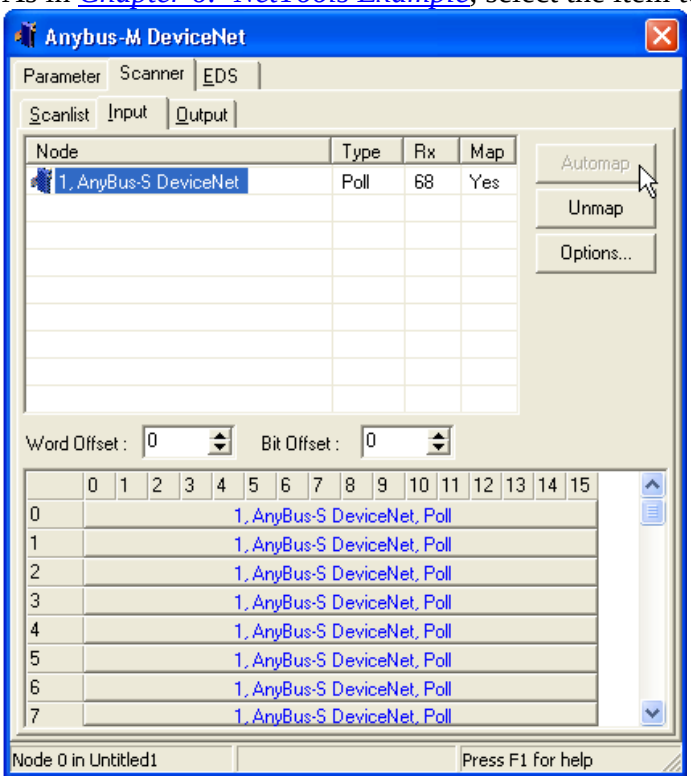
Configure the polling as desired, entering 68 as the RX/TX produced/consumed bytes. Note the 68 was from the Anybus-S parameters or that calculated based on installed modules.



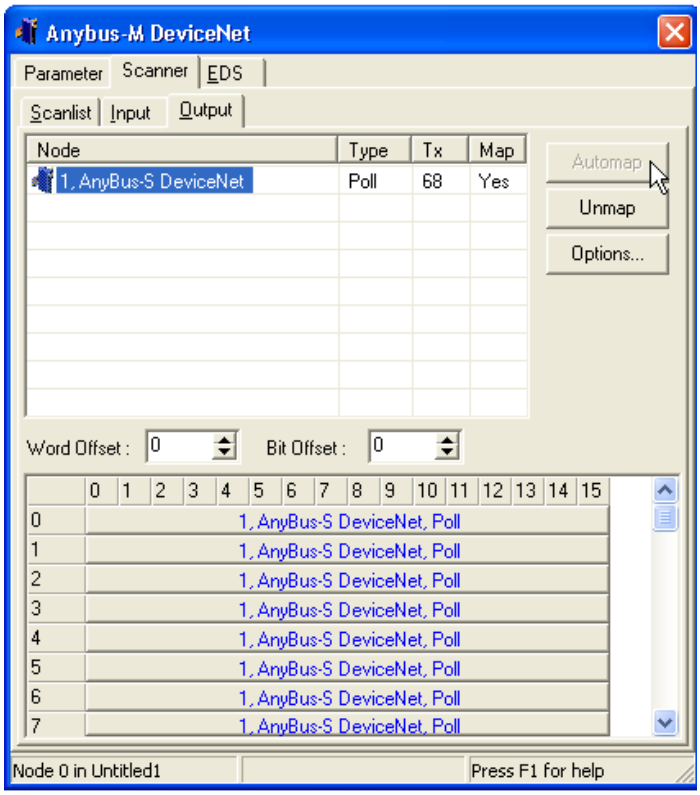

Click OK after any changes and the scan list will appear as below, select the Input tab to begin mapping the produced data (received):



As in *Chapter 6: NetTools Example*, select the item to be mapped followed by Automap:

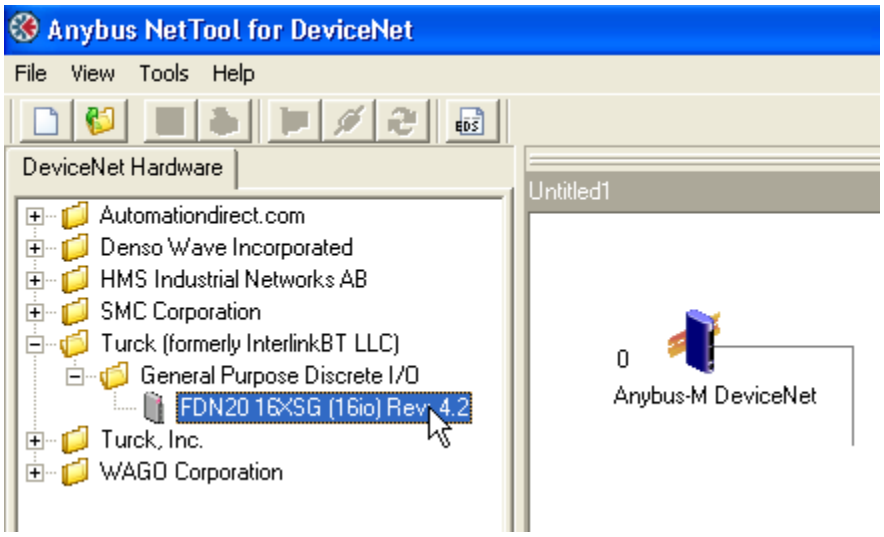


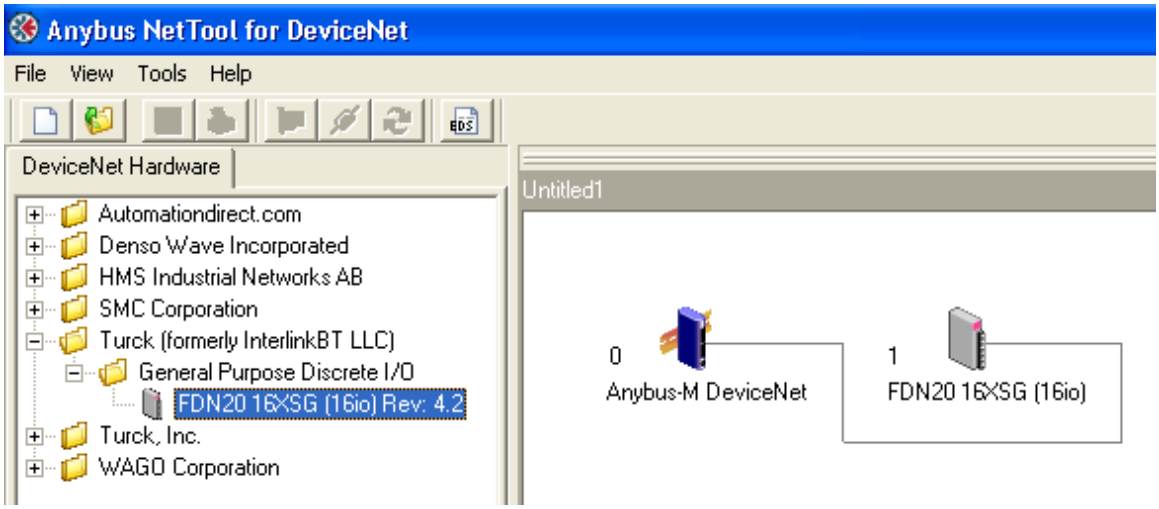

Do the same for the output data:



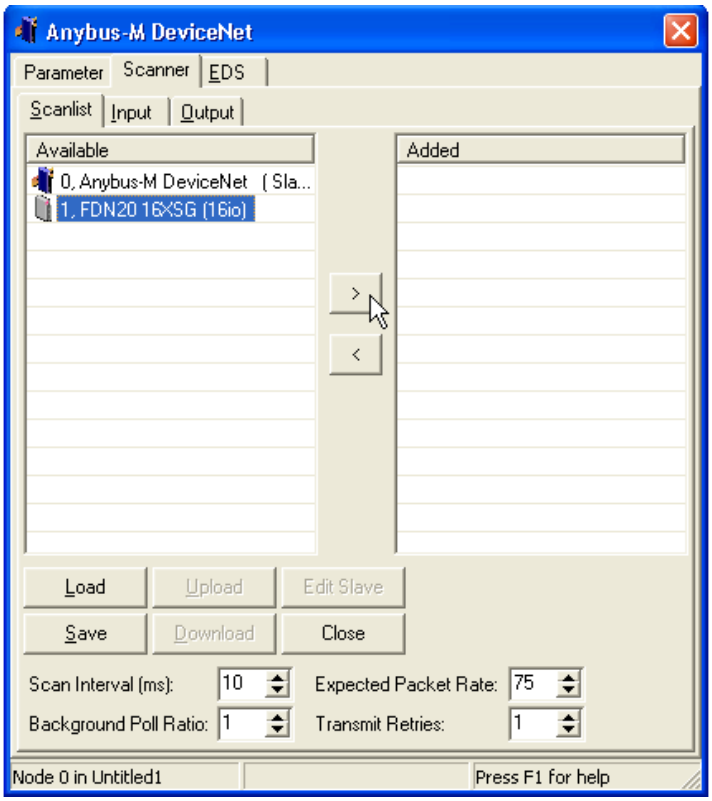
## *Turck FDN20*

The Turck FDN20 is a digital input/output slave node available in a number of different configurations. The example here discusses the 16 in, 16 out unit. The EDS file FDN20-16XSG_R4 was installed into NetTools. As before, drag and drop it into the network area, setting its MACID as desired:

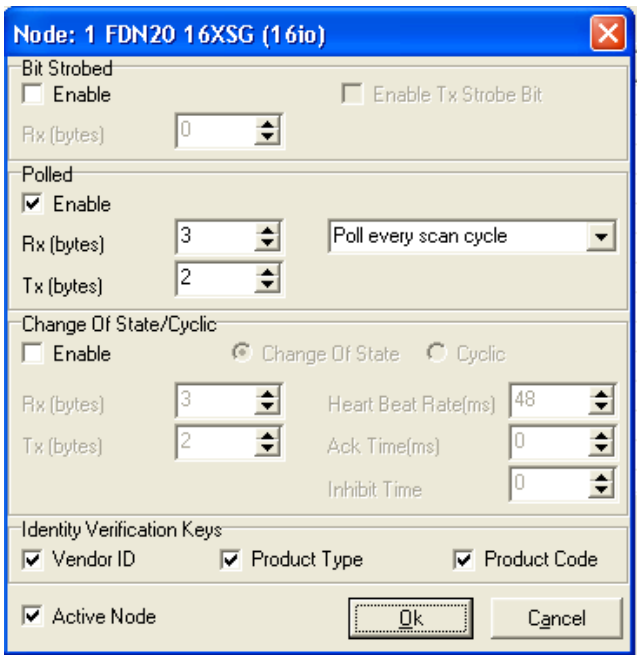


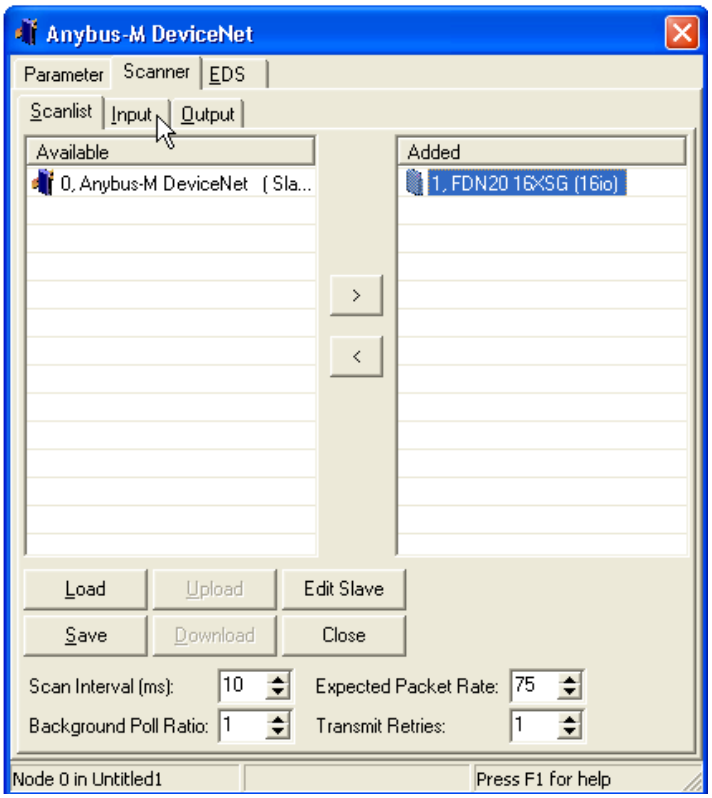

Double click the Anybus-M icon to begin configuration, selecting the 'Scanlist' tab and then move the FDN20 to the active scan list (Added column):



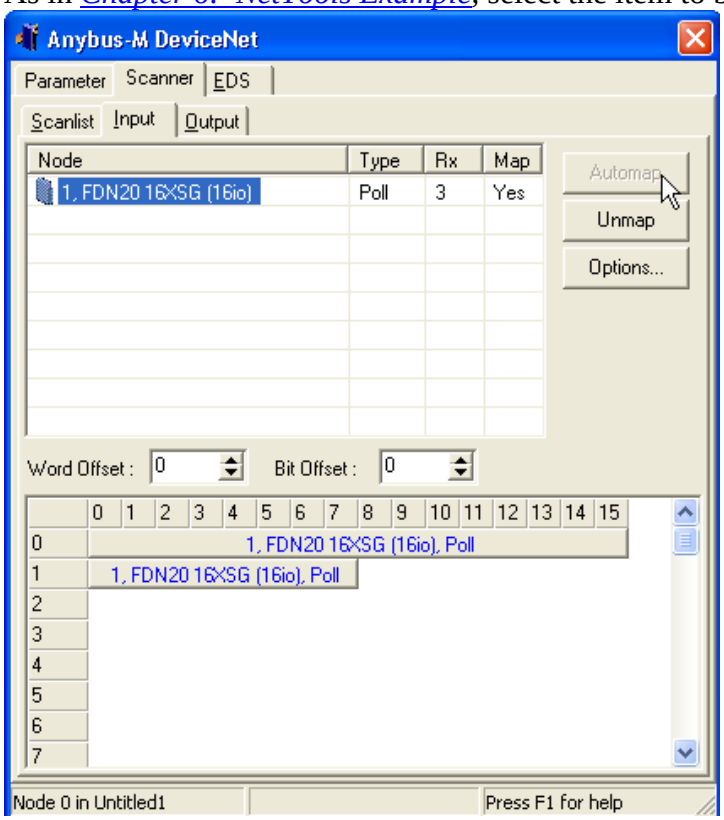

Configure the polling as desired, note that the EDS file states that there will be 3 received bytes and 2 transmitted bytes:



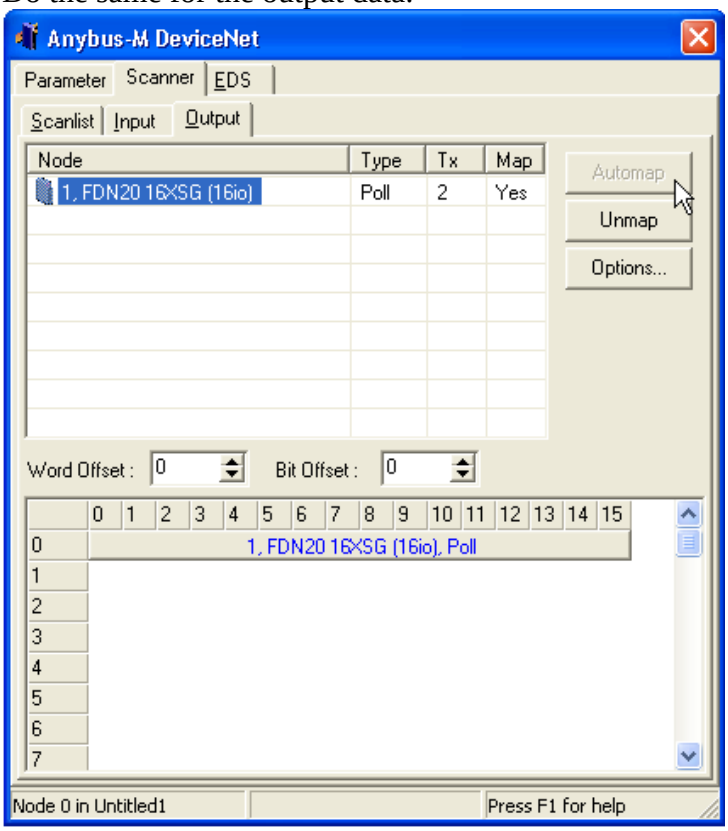

Click OK after any changes and the scan list will appear as below; select the Input tab to begin mapping the produced data (received):




As in *Chapter 6: NetTools Example*, select the item to be mapped followed by Automap:



Do the same for the output data:

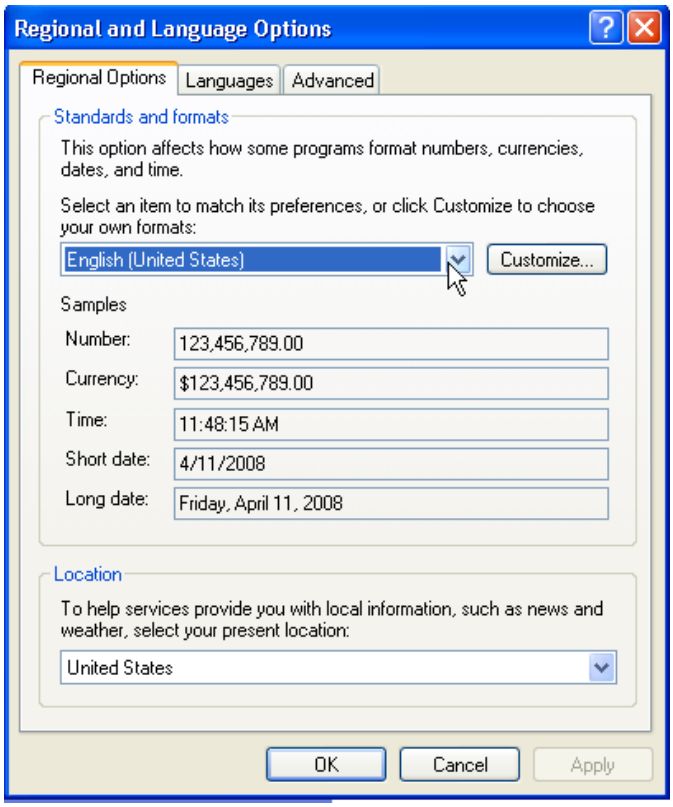


## *Brooks DeviceNet MFCs*

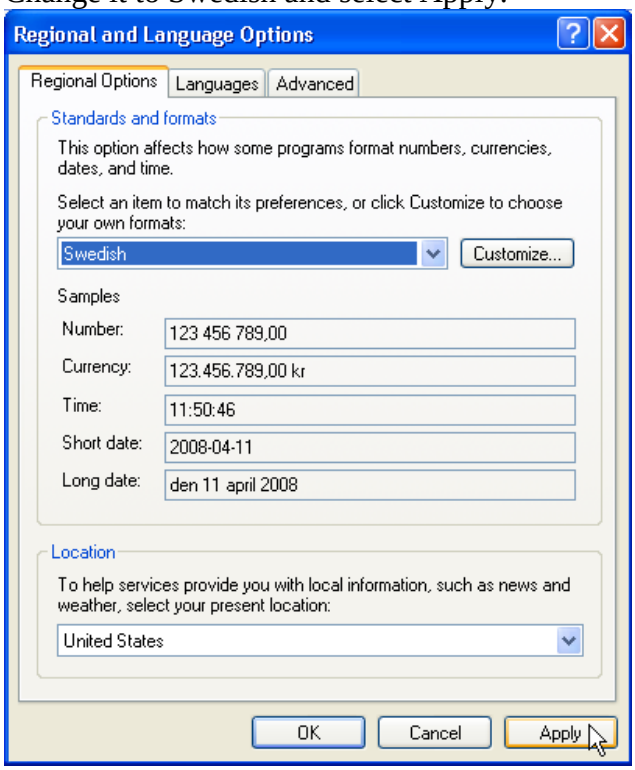
The Brooks Mass Flow Controller has a number of different models. This example should be fairly generic as the EDS file addresses the SLA 58xx/68xx/69xx. The EDS file Delta_MFC-Rel2.1 must be installed into NetTools. Due to a problem in NetTools you must set your computer's Region to that of Sweden prior to EDS installation, and restore it to English/US afterwards. This is caused by a differing way in presenting floating point information using commas instead of periods. This problem will be fixed in a later release of NetTools but has been observed in V3.1.1.1.

Invoke NetTools as you normally would and just prior to importing the Brooks EDS file do the following:

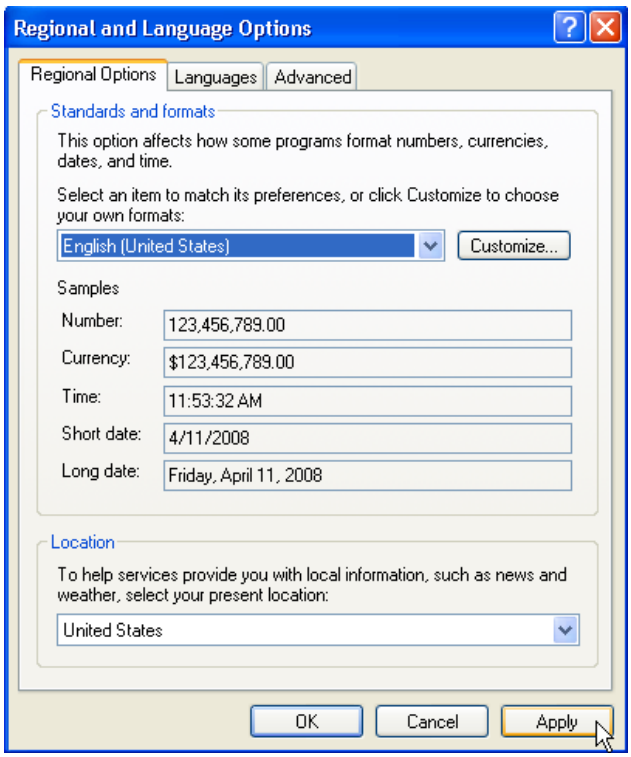
1. Set the Region to Sweden, you will need local administrator rights. Select Control Panel->Regional and Language Options. The following will appear for English:



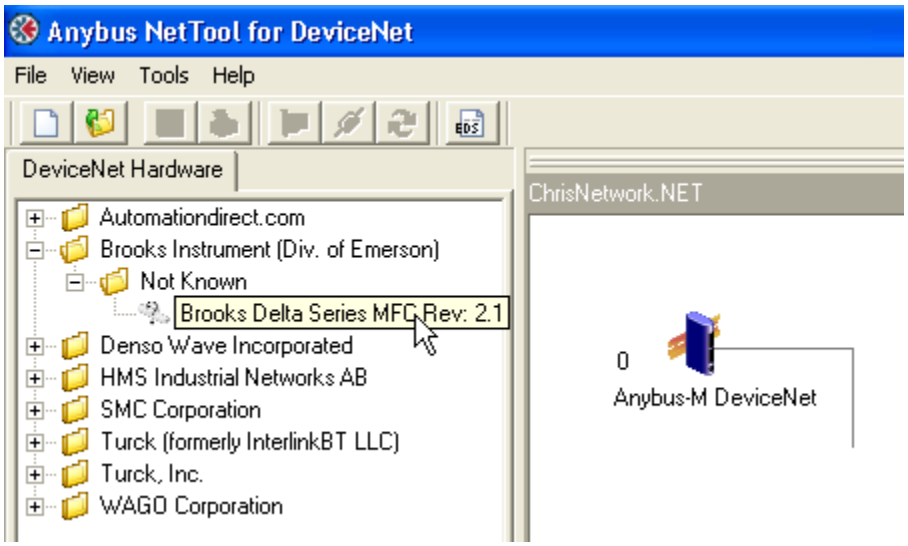
Change it to Swedish and select Apply:

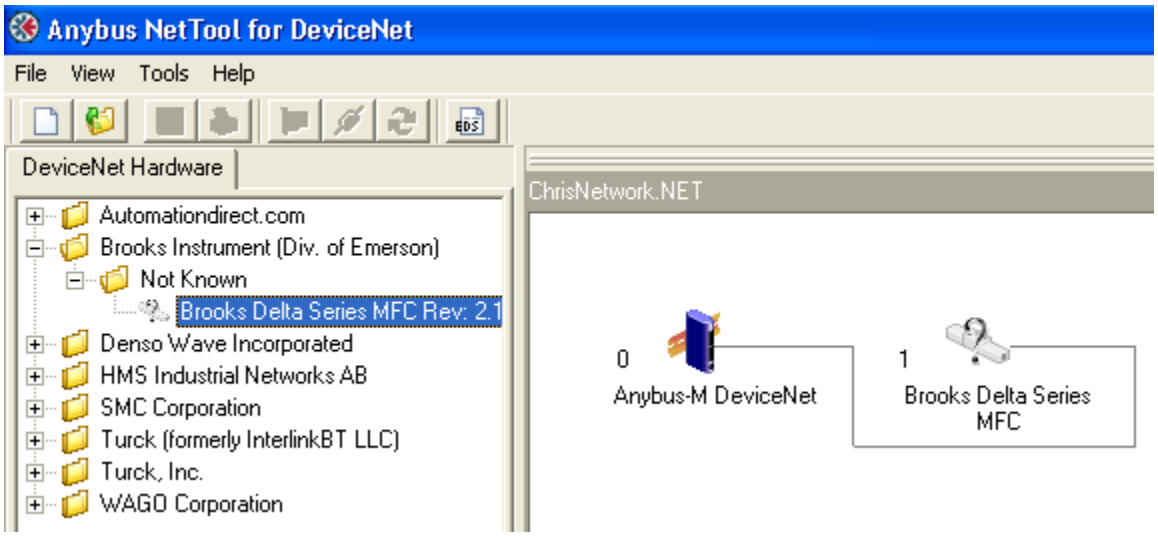



2. In NetTools import the Brooks EDS file.
3. Set the region back to English, exit NetTools and re-load, proceed normally:

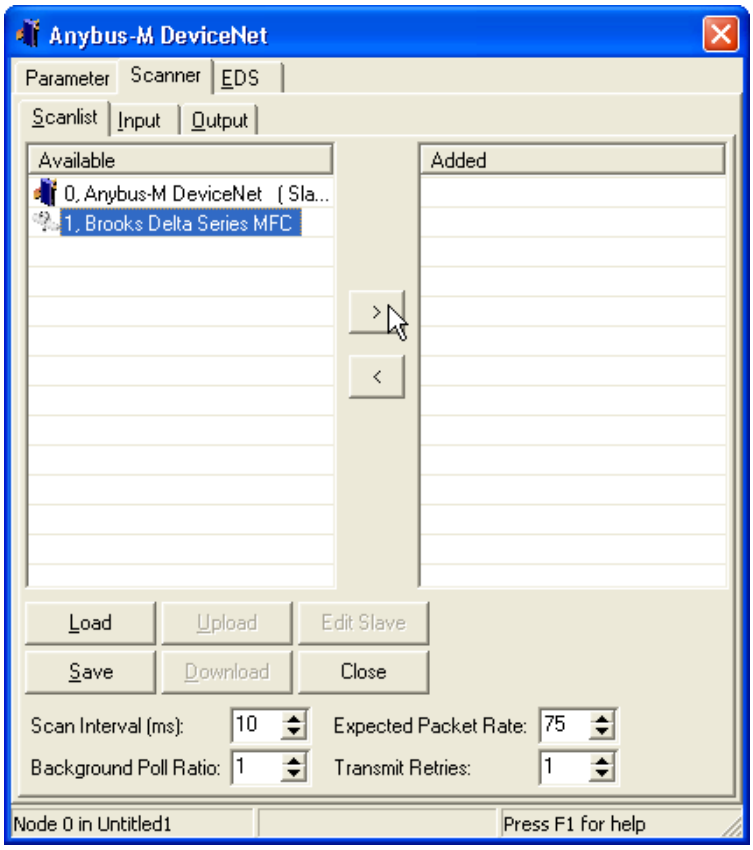


With the EDS file installed you may now proceed to map in the Brooks MFC. As before drag and drop it into the network area setting its MACID as desired:

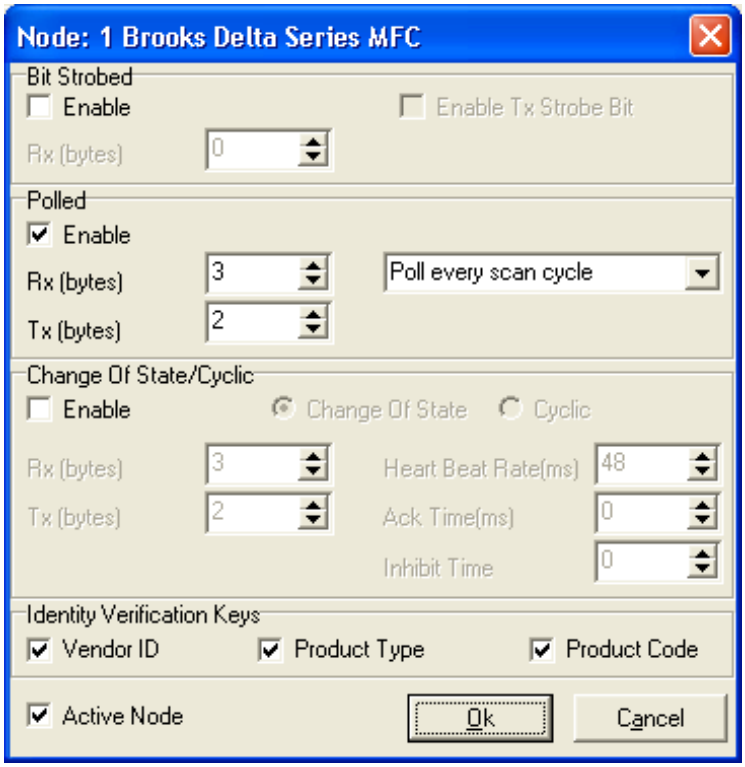


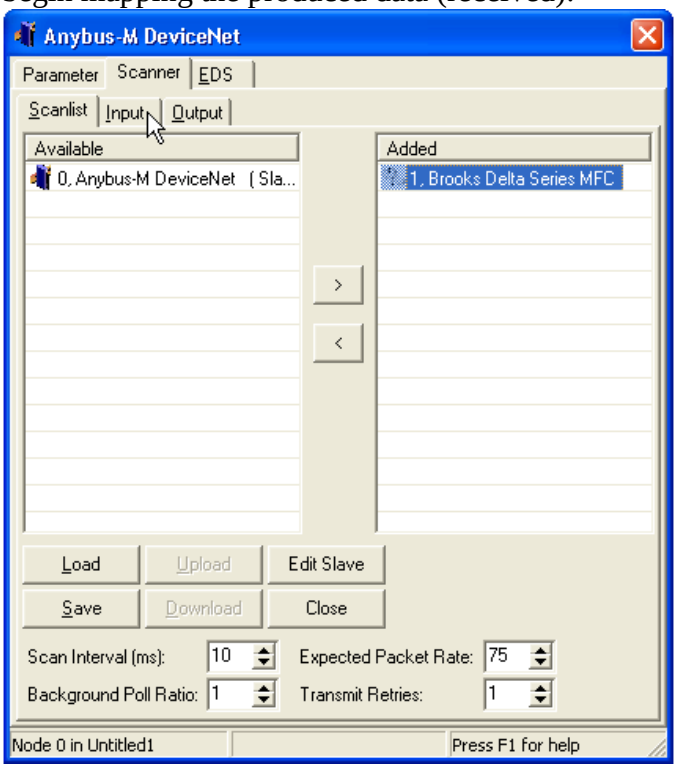

Double click the Anybus-M icon to begin configuration, selecting the 'Scanlist' tab and then move Brooks to the active scan list (Added column):



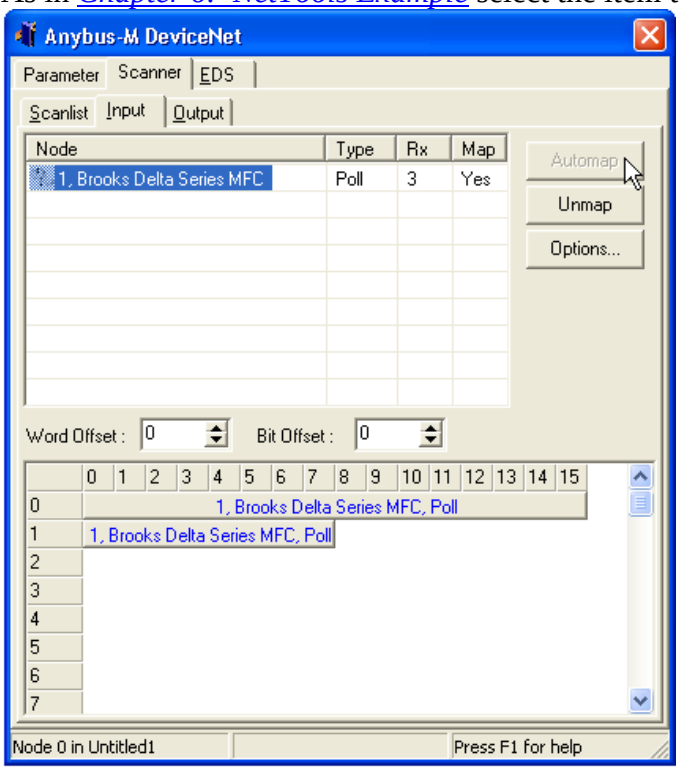

Configure the polling as desired, note that the EDS file states that there will be 3 received bytes and 2 transmitted bytes:



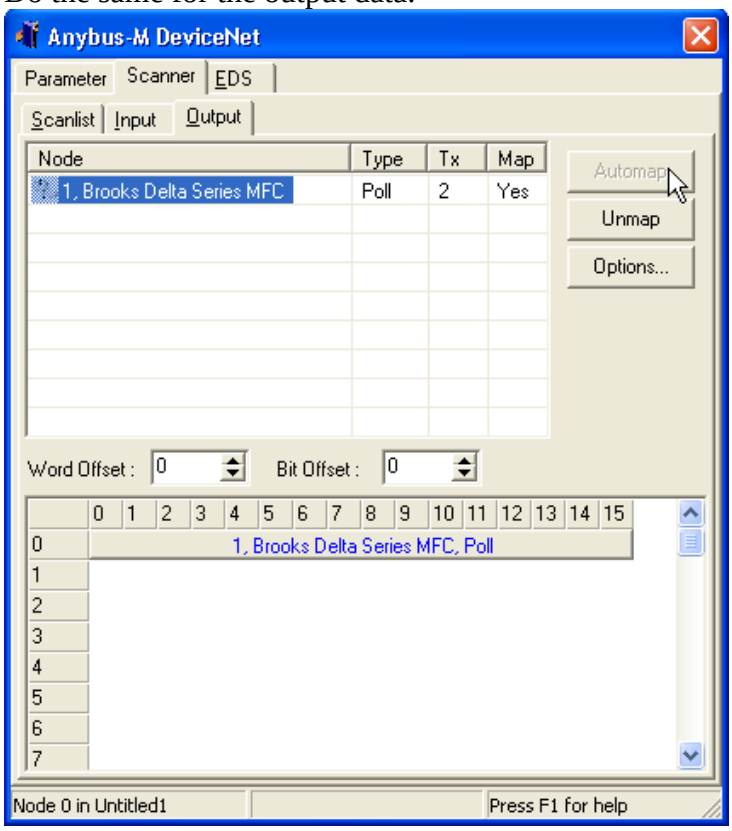
Click OK after any changes and the scan list will appear as below, select the Input tab to begin mapping the produced data (received):




As in *Chapter 6: NetTools Example* select the item to be mapped followed by Automap:



Do the same for the output data:



*Blank*

Appendix B

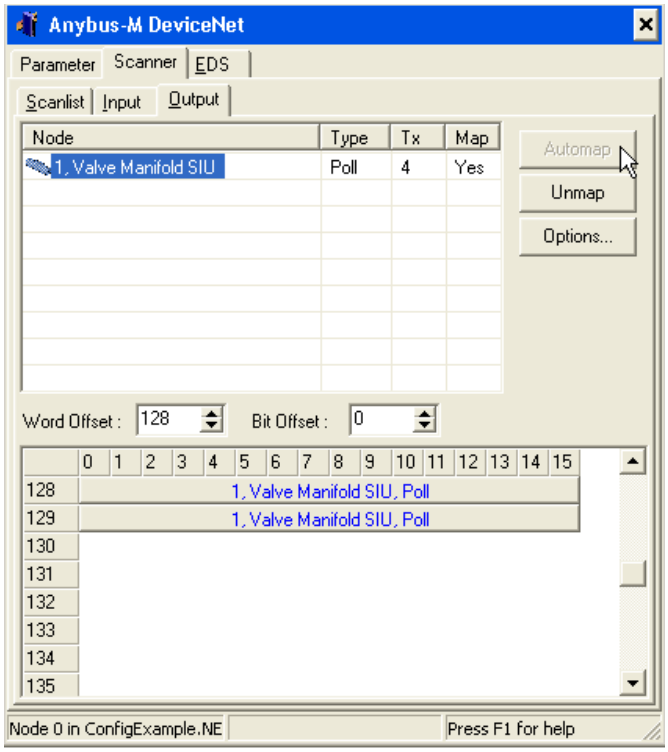
# [B] Backward Compatibility



The prior M3-61A release, firmware M361AV0101 is not compatible with the new functionality and requires a couple of minor changes prior to upgrade.

## *NetTools Output Offset*

The prior revision required an offset of 128 words in the output section of the dual port configuration screen. This is now a 0 offset. All network devices must have this configuration changed and the master reloaded. Previous example:




## XML Configuration File

The previous release required no configuration file and mapped all remaining I/O to the DeviceNet network, supporting only one Master per Model 5300. Also, only Digital I/O was supported and not analog or explicit messaging. To be backward compatible with this configuration, a simple XML file can be loaded using the telnet fupdate command. Assuming slot 4 for the module and an xml file called M361ASDTV0102.xml:

fupdate slot 4 M361ASDTV0102.xml

The XML contents would be as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<DEVICENET_DEF>
        <VERSION>0102</VERSION>
        <DIN_DEF>
                <QTY>1024</QTY>
        </DIN_DEF>
        <DOUT_DEF>
                <QTY>1024</QTY>
        </DOUT_DEF>
</DEVICENET_DEF>
```