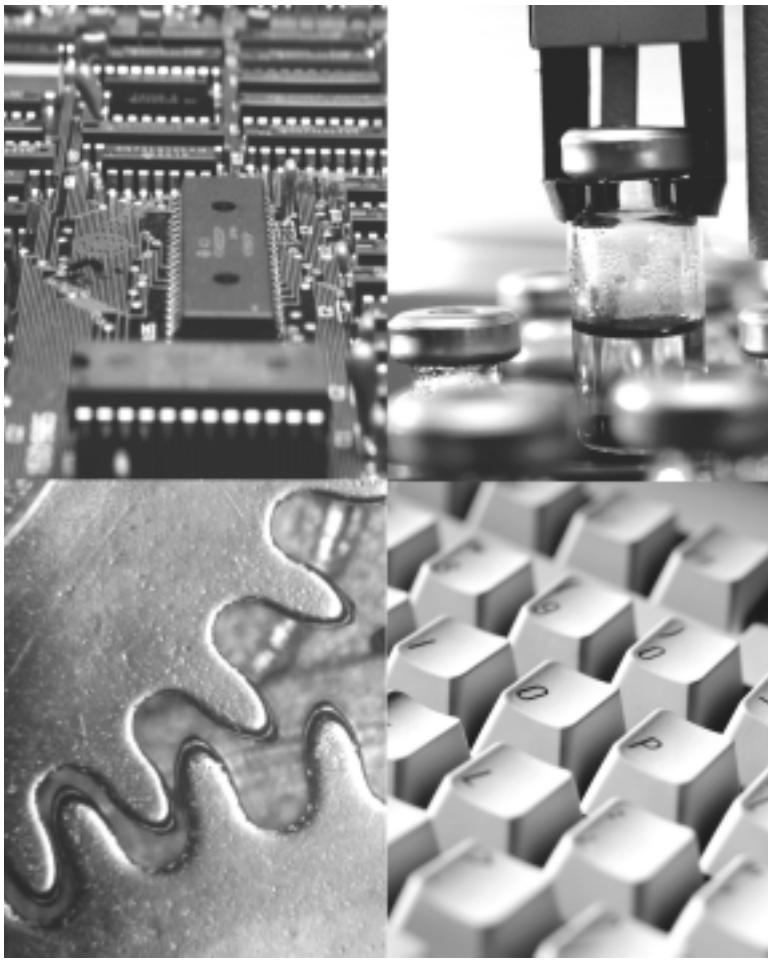
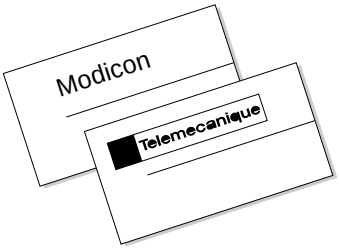
Modicon Modbus Plus
Host Interface Device Driver Manual For Windows NT

890 USE 110 00 Version 2.0





GROUPE SCHNEIDER

■ Modicon ■ Square D ■ Telemecanique

Data, Illustrations, Alterations
Data and illustrations are not binding. We reserve the right to alter products in line with our policy of continuous product development. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us by e-mail at techcomm@modicon.com.

Training
Schneider Automation Inc. offers suitable further training on the system.

Hotline
See addresses for Technical Support Centers at the end of this publication.

Trademarks
All terms used in this publication to denote Schneider Automation Inc. products are trademarks of Schneider Automation Inc.

All other terms used in this publication to denote products may be registered trademarks and/or trademarks of the corresponding corporations. Microsoft and MS-DOS are registered trademarks of Microsoft Corporation. Windows is a brand name of Microsoft Corporation in the USA and other countries. IBM is a registered trademark of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation.

Modicon Modbus Plus

Host Interface Device Driver Manual for Windows NT

890 USE 110 00 Version 2.0

February 1998

GROUPE SCHNEIDER

■ Modicon ■ Square D ■ Telemecanique

Contents

Selecting Modbus Plus Libraries and Drivers

1

1.1 Modbus Plus Communications Overview

The Modbus Plus drivers described in this manual provide communications connectivity between software programs (such as Modsoft) and Schneider Automation's Modbus Plus PC network cards in a 32 bit Windows environment. These network cards are:

Adapter	Part Number
SA85 Single Channel Modbus Plus Adapter Card	AM-SA85-000
SA85 Dual Channel Modbus Plus Adapter Card	AM-SA85-002
PCMCIA Type II Modbus plus Adapter Card	416NHM21200
PCMCIA Type III Modbus Plus Adapter Card	416NHM21203

1.2 Application-Specific Drivers and Libraries

There are three components to the Modbus Plus MBX drivers. The selection of the correct driver components is determined by the requirements of your application.

CAUTION

MIXING DRIVERS WILL RESULT IN DRIVER FAILURE

Make sure you are using the correct driver components for your operating system. Windows 95 and Windows NT drivers are specific to Windows 95 and Windows NT operating systems, respectively. (For dual-boot system exception, see Chart Notes below. Mixing Windows 95 and Windows NT drivers within one operating system will result ina driver failure that may be difficult to correct.

Failure to observe this precaution can result in equipment damage.

The drivers are as follows:

- **The 32-bit MBX Driver** (previously called “Local NETLIB Library”) provides connectivity between Windows 95 or Windows NT based applications and Modbus Plus adapter cards. For Windows 95 use the SW-LNET-I95 driver. For Windows NT use SW-LNET-INT. Refer to the selection chart *Stand-alone or Remote Server Configurations* on page 3. See also *Example of a Stand-Alone Configuration* on page 5 for an illustration of how these libraries are applied.
- **The Virtual MBX Driver** (previously called “NETLIB/NetBIOS Virtual Device Driver” allows existing 16-bit DOS or Windows 3.1x applications to run under Windows 95 or Windows NT in their original binary form, without modification. For Windows 95 use the SW-WVDD-I95 Virtual Device Driver. For Windows NT use SW-WVDD-INT. Refer to the selection charts in *Modbus Plus Driver Selection Charts* on page 3. Also see *Example of a Stand-Alone Configuration* on page 5 for an illustration of the application of these drivers.
- **The 32-bit Remote MBX Driver** (previously called “Remote NETLIB Library”) provides connectivity between 32-bit Windows 95 or Windows NT based applications and Modbus Plus adapter cards running in remote Windows NT servers. For Windows 95 use the SW-RNET-I95 library. For Windows NT use SW-RNET-INT. Refer to the selection chart in *Remote Client Configurations* on page 4. Also see *Example of a Remote Client Configuration* on page 7 for an illustration of the application of these libraries.

1.3 Modbus Plus Driver Selection Charts

Use the following driver selection charts to determine the correct driver(s) for your operating system and application. Then refer to the appropriate chapter of this manual for detailed information.

Stand-alone or Remote Server Configurations

Use this chart to determine the correct driver(s) for your operating system and application in a Stand Alone (Local) configuration. The recommended order of installation is MBX (Local) Driver followed by Virtual Driver. See *Chart Notes* on page 4 for important information.

Adapter Type >>>	**SA85 Modbus Plus Adapter Card Single and Dual Channel**			**PCMCIA Modbus Plus Adapter Card Type II and Type III**		
Operating System	**16 Bit Applications**	**32 Bit Applications**	**Mixed 16 & 32 bit Applications**	**16 Bit Applications**	**32 Bit Applications**	**Mixed 16 & 32 bit Applications**
Win 95	SW-LNET-I95 and SW-WVDD-I95	SW-LNET-I95	SW-LNET-I95 and SW-WVDD-I95	SW-LNET-I95 and SW-WVDD-I95	SW-LNET-I95	SW-LNET-I95 and SW-WVDD-I95
Win NT	SW-LNET-INT and SW-WVDD-INT	SW-LNET-INT	SW-LNET-INT and SW-WVDD-INT	SW-LNET-INT and SW-WVDD-INT	SW-LNET-INT	SW-LNET-INT and SW-WVDD-INT

Remote Client Configurations

Use this chart to determine the correct driver(s) for your operating system and application when your Modbus Plus communications card is installed in a remote computer. The remote computer must be running SW-LNET-INT under Windows NT. The recommended order of installation is MBX (Local) Driver, then Virtual Driver if needed, followed by the Remote Driver. See *Chart Notes* on page 4 for important information.

Adapter Type >>>	**SA85 Modbus Plus Adapter Card Single and Dual Channel**			**PCMCIA Modbus Plus Adapter Card Type II and Type III**		
Operating System	**16 Bit Applications**	**32 Bit Applications**	**Mixed 16 & 32 bit Applications**	**16 Bit Applications**	**32 Bit Applications**	**Mixed 16 & 32 bit Applications**
Win 95	SW-RNET-I95 and SW-WVDD-I95	SW-RNET-I95	SW-RNET-I95 and SW-WVDD-I95	SW-RNET-I95 and SW-WVDD-I95	SW-RNET-I95	SW-RNET-I95 and SW-WVDD-I95
Win NT	SW-RNET-INT and SW-WVDD-INT	SW-RNET-INT	SW-RNET-INT and SW-WVDD-INT	SW-RNET-INT and SW-WVDD-INT	SW-RNET-INT	SW-RNET-INT and SW-WVDD-INT

Chart Notes

Dual-Boot Systems: Both Windows 95 and Windows NT MBX components may be installed on a dual-boot system, i.e., a system that optionally boots either Windows NT or Windows 95 optionally.

CAUTION

MIXING DRIVERS WILL RESULT IN DRIVER FAILURE

Make sure you are using the correct driver components for your operating system. Windows 95 and Windows NT drivers are specific to Windows 95 and Windows NT operating systems, respectively. (For dual-boot system exception, see Chart Notes below. Mixing Windows 95 and Windows NT drivers within one operating system will result ina driver failure that may be difficult to correct.

Failure to observe this precaution can result in equipment damage.

16 Bit Applications Include: ModLink, Modsoft, FactoryLink Version 4.x, ProWorx, and InTouch for Windows 95.

32 Bit Applications Include: FactoryLink ESC 6.xx and InTouch for Windows NT, plus any new applications that are based on Windows NT or Windows 95.

1.4 Example of a Stand-Alone Configuration

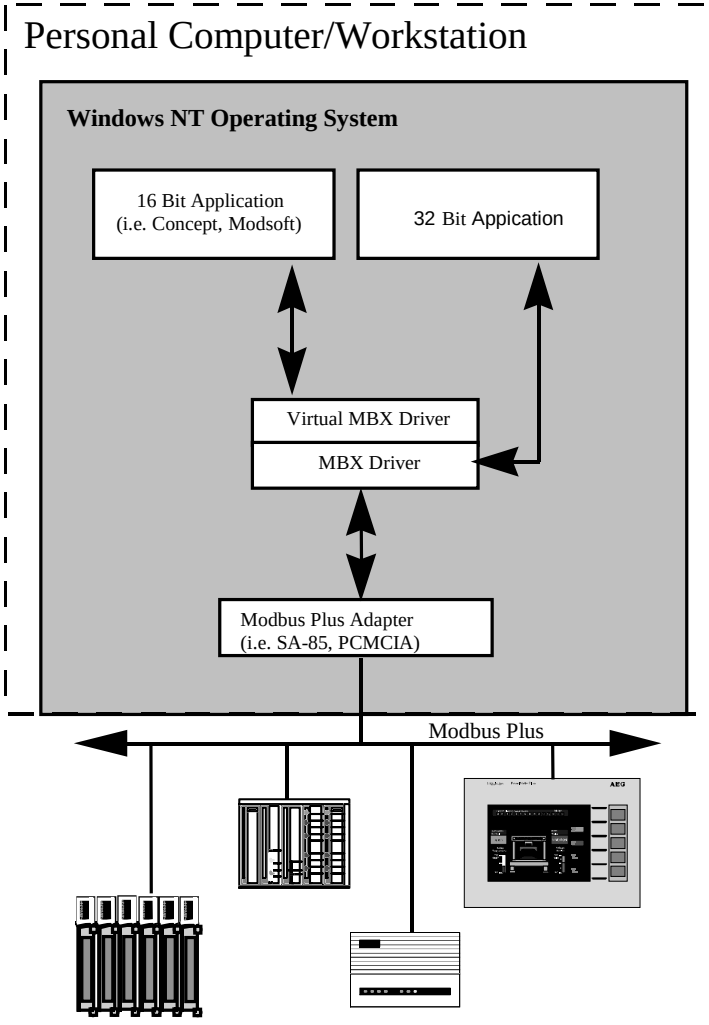


Figure 1 Example of a Stand-Alone Configuration

The following table lists the components of a stand-alone system:

Quantity	Description
1	Concept, Modsoft, Modlink, MBPSTAT, FactoryLink, other 16 or 32 bit application software
1	MBX (formerly called “Local”) Driver (SW-LNET-INT)
1	Virtual MBX Driver (SW-WVDD-INT)
1	Windows NT Operating System Software
1	Modbus Plus Host Interface Adapter (i.e., SA85, PCMCIA)

1.5 Example of a Remote Client Configuration

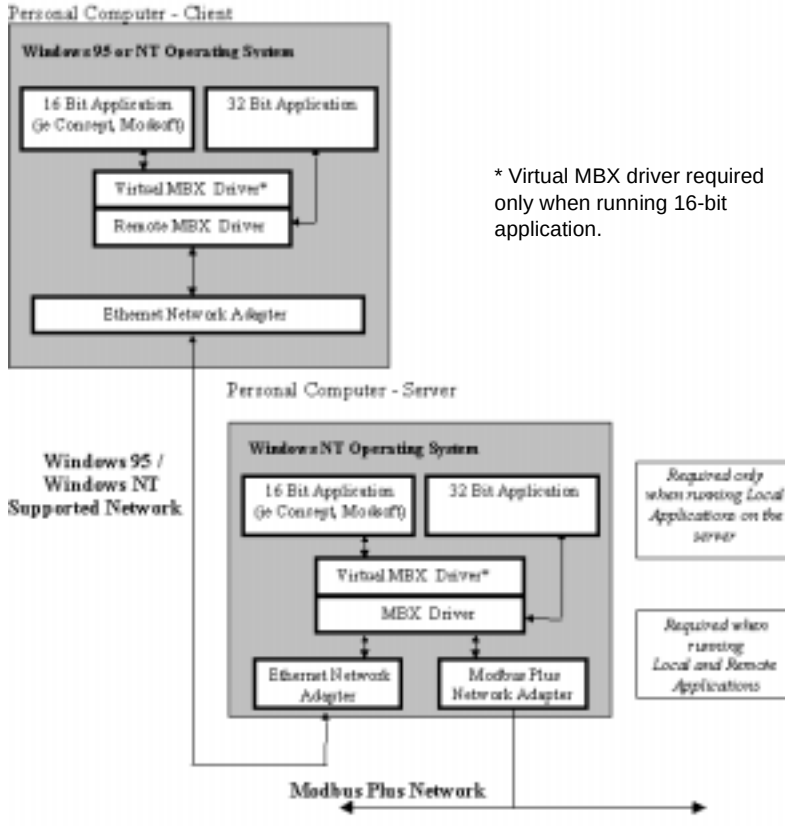


Figure 2 Example of a Remote Client Configuration

The following table lists server components:

Quantity	Description
1	MBX Driver (SW-LNET-INT)
1	Virtual MBX Driver (SW-WVDD-INT) - The virtual MBX driver is only required when running 16-bit applications on the server.

The following table lists client components:

Quantity	Description
1	Remote MBX Driver (SW-RNET-INT or SW-RNET-I95)
1	Virtual MBX Driver (SW-WVDD-INT or SW-WVDD-I95) - The virtual MBX driver is only required when running 16-bit applications on the server.

MBX Driver

2

2.1 Introduction

The 32-bit MBX Driver for Microsoft Windows NT operating system provides connectivity between Schneider Automation ModConnect host interface adapters and 32-bit applications running under Windows NT.

The kernel mode device driver that is part of this product is the highest performance Modbus Plus driver in the industry. The driver can operate in either *interrupt* or *polled mode* and supports all current Schneider Automation ModConnect host interface adapters for ISA, EISA, MCA, and PC Card (PCMCIA) buses. Multiple interface cards can be installed at the same time, limited only by the number of available slots. Full implementation of all MB+ features provides support for *Data Master/Slave*, *Program Master/Slave*, and *Global Data*. The high performance native API of the MBX Driver is designed to take advantage of all features of the event-driven, multitasking, multithreaded Windows NT operating system.

The driver provides compatibility with applications written to utilize it's high performance application programming interface (API) as well as the industry standard NETLIB interface specification from Schneider Automation. The 32-bit NETLIB compatibility provides an excellent bridge for developers who would like to port their 16-bit, NETLIB-compatible applications to Windows NT. Developers of new applications can use either the NETLIB or the high performance MBXAPI programming interface. To obtain the MBXAPI software development kit, including the MBXAPI specification, MBXAPI sample source code, and NETLIB sample source code, contact Schneider Automation. For complete reference of all NETLIB library functions refer to "*Modicon IBM Host Based Devices User's Guide*" from Schneider Automation (890 USE 102 00).

The performance of the MBX Driver, along with other diagnostic information, can be monitored through the *Performance Monitor* utility that is provided by Microsoft with Windows NT. Performance information from multiple interface cards can be monitored simultaneously.

Remote Connectivity

The MBX Driver also includes the MBX Remote Server. The MBX Remote Server provides access to remote client nodes over any Windows NT-compatible computer network, enabling the remote client nodes to access the (configured) MBX Driver adapters residing on the same node with the MBX Remote Server. The remote client can either be a Windows NT or a Windows® 95 node and it must be running the separately purchased Remote MBX Driver. However, a host interface adapter, such as Schneider Automation's SA85 card, is not required on the client node. The Remote MBX Driver provides complete MBX functionality to the client node, including support for *Data Master/Slave*, *Program Master/Slave* and *Global Data*. Any node on the network can be configured as a client to a number of remote servers and at the same time communicate to it's local Modbus Plus networks.

Running 16-Bit Software

Another companion product, the Virtual MBX Driver, will allow all 16-bit NETLIB/NetBIOS-compatible applications (e.g., ModSoft) to run concurrently with all 32-bit applications in the same computer. The Virtual MBX Driver will allow multiple 16-bit applications as well as multiple instances of a single 16-bit application to run under the 32-bit Windows NT operating system. (The Virtual MBX Driver is purchased separately and it requires either the MBX Driver or the Remote MBX Driver to operate).

2.2 Installation

The MBX Driver version 4.00 is compatible with Windows NT versions 3.51 and 4.0. Differences between these versions of Windows NT require different installation procedures as listed below.

Procedure **Windows NT 3.51 Installation Procedure**

1. Exit all applications that are using MBX products like the *MBX Driver*, the *Remote MBX Driver*, or the *Virtual MBX Driver* before installing the new version. If the Virtual MBX Driver is loaded globally from the CONFIG.NT and AUTOEXEC.NT located in the System32 directory, remove references to MBP16.SYS and MBP16VEC in these files and reboot the system before proceeding with the installation.
2. Place the installation disk in the floppy drive (either Drive A: or B:).
3. Select ***Run...*** from the *Program Manager*'s ***File*** menu. Enter A:\SETUP (or B:\SETUP) and follow the on-screen instructions.
4. Configure the MBX Driver to match the installed adapter card(s) using the *MBX Device Configuration* icon in the *Modicon WinConX - MBX Series* program group.
5. Shut down and restart the system.

Procedure **Windows NT 4.0 Installation Procedure**

1. Exit all applications that are using MBX products like the *MBX Driver*, the *Remote MBX Driver*, or the *Virtual MBX Driver* before installing the new version. If the Virtual MBX Driver is loaded globally from the CONFIG.NT and AUTOEXEC.NT located in the System32 directory, remove references to MBP16.SYS and MBP16VEC in these files and reboot the system before proceeding with the installation.
2. Place the installation disk in the floppy drive (either Drive A: or B:).
3. Go to the *Control Panel*, select the ***Add/Remove Programs*** icon, and click the ***Install*** button. Click ***Next>*** and then ***Finish*** to install the software onto the hard drive.
4. Configure the MBX Driver to match the installed adapter card(s) using the *MBX Device Configuration* shortcut icon in the *MBX Driver* sub group located in the *WinConX* program group (i.e. ***Start/Programs/WinConX/MBX Driver/MBX Device Configuration***).
5. Shut down and restart the system.

2.3 MBX Driver Configuration

The MBX Driver includes a kernel mode device driver that requires a number of parameters to be specifically set to match the configuration of the devices used in a system. A utility program, called *MBX Device Configuration,* is provided to create and edit devices and their configuration.

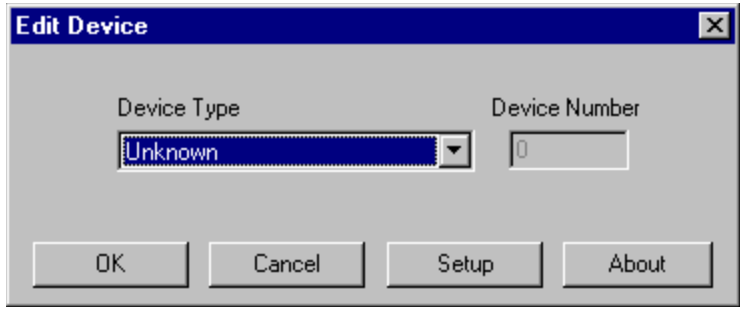
In order to start this utility program, double click the *MBX Device Configuration* icon in the *MBX Driver* subgroup located in the *WinConX* program group (i.e. ***Start/Programs/WinConX/MBX Driver/MBX Device Configuration***). On systems running Windows NT version 3.51, the *MBX Device Configuration* icon is located in the *Modicon WinConX - MBX Series* program group).

During startup the driver may detect a number of configuration problems. When a problem is detected, the driver sends an appropriate message to the Windows NT *Event Logger*. Refer to the *Troubleshooting the MBX Driver* on page 32 for more information.

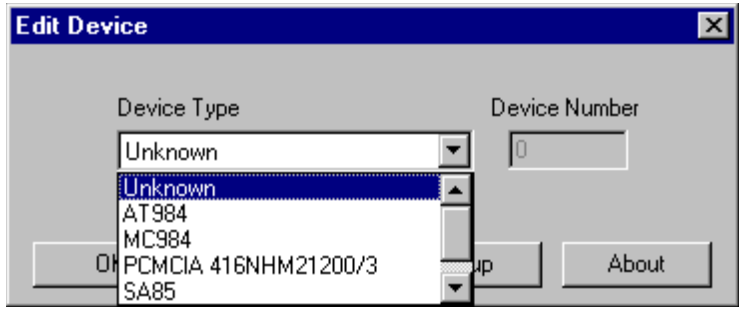
Procedure **First Time Configuration**

1. Run the device configuration utility (*MBX Device Configuration*). The following dialog will appear:

2. Click the ***New*** button. A dialog box will be displayed showing the *Device Number* and *Device Type*.



3. Select the *Device Type* using the pull-down window. The *Device Number* refers to the number that you assign to every device installed in your system. **This is not the Modbus Plus node address!** By default, the configuration utility will try to use consecutive numbers for the devices starting from zero. However, this is not a requirement and the user can assign numbers that are not consecutive. However, when running Modsoft or Concept, adapter numbers 0 or 1 must be used.

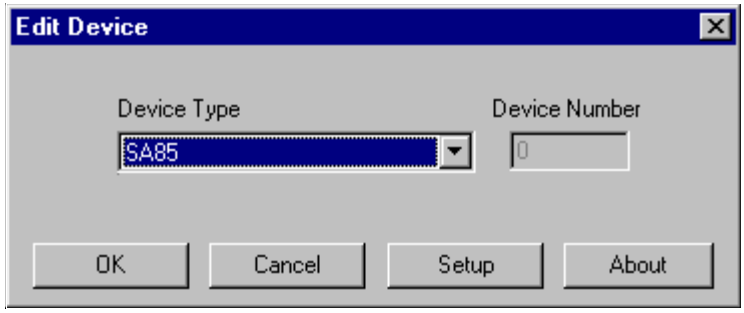


4. Click the ***Setup*** button to configure the device or ***OK*** button when finished.

Procedure **Editing Current Configuration**

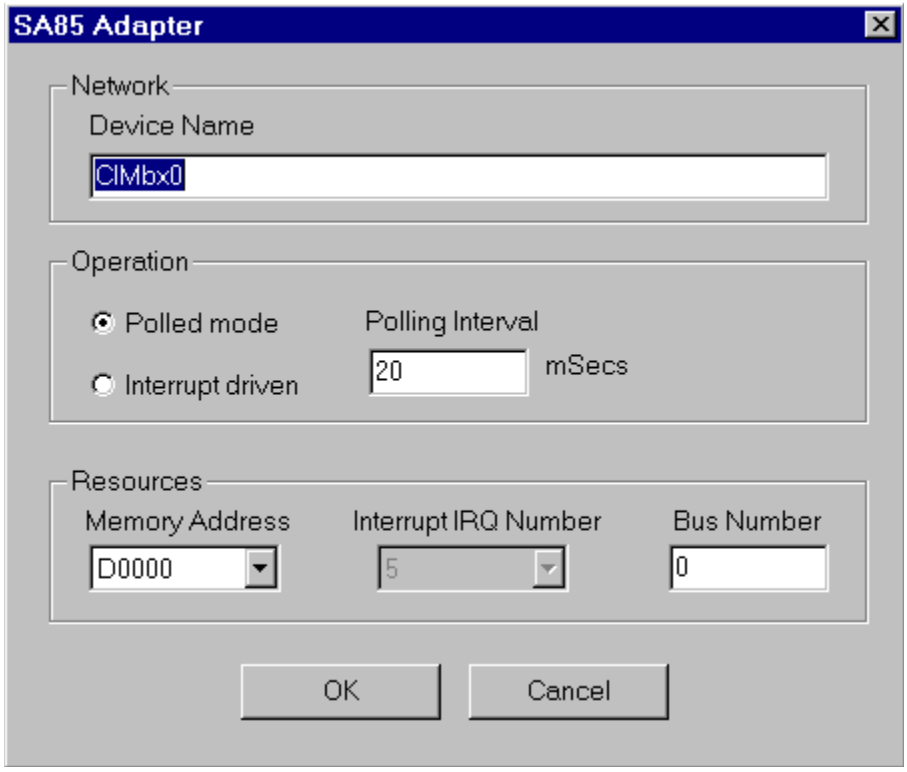
1. Run the device configuration utility (*MBX Device Configuration*). The following dialog will appear:

2. Select the device to be configured from the list and click the ***Edit*** button. Verify the *Device Type* and click the ***Setup*** button.



3. A dialog box will be displayed showing the parameters that need to be configured for that *Device Type*. Each parameter must be set appropriately to match the installed adapter. All parameters are listed below by *Device Type*.

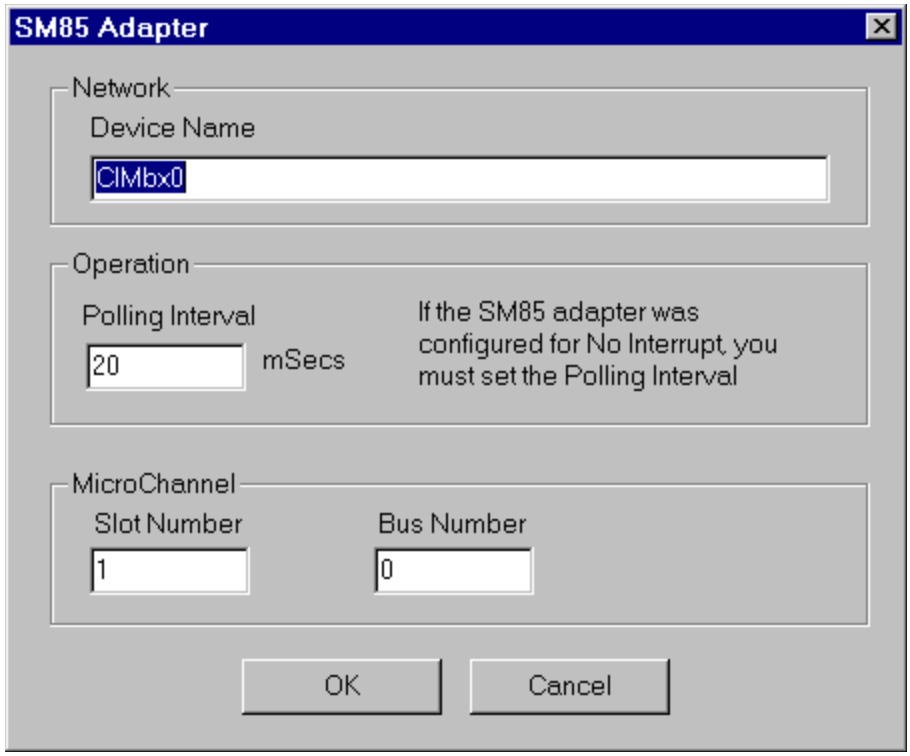
2.3.1 SA85 Adapter



- **Device Name**: This parameter allows the user to assign a name to identify the device. The default for this parameter is “CLMBX#”, where ‘#’ is the selected device number.
- **Polled mode/Interrupt driven**: The kernel mode device driver can operate in either polled mode or interrupt mode. Select the proper mode of operation depending on the adapter card configuration. (The interrupt mode provides better performance than the polled mode. However, interrupt mode requires more processor overhead). It is legal to mix interrupt and polled modes of operation for different cards in the same system. The default for this parameter is polled mode.
- **Polling Interval**: This parameter specifies the polling interval in milliseconds that the driver will use when running in the polled mode. The valid range for the polling interval is 20-1000 milliseconds. The default value is 20 ms.

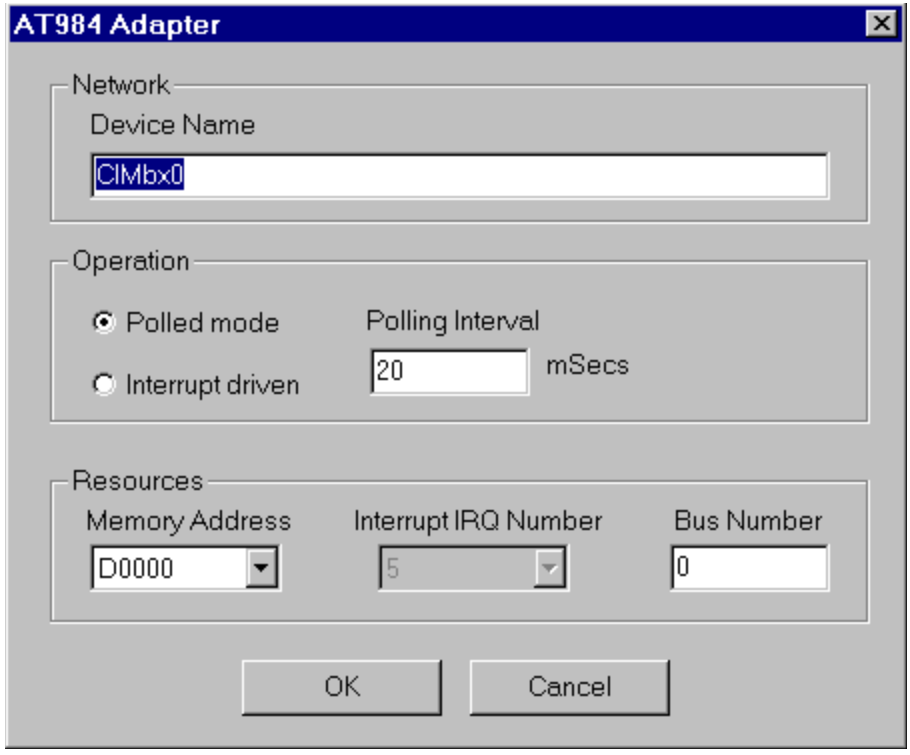
- **Memory Address**: This parameter specifies the base address of the adapter card (e.g. D0000). The default for this parameter is D0000. This address must match the switch settings on the card and has to be unique for each adapter card.
- **Interrupt IRQ Number**: When interrupt mode is selected, this parameter specifies the IRQ number for the interrupt line used (e.g. 5). The default for this parameter is 5. This IRQ number must match the IRQ setting on the adapter card and has to be a unique value for each card in the system.
- **Bus Number**: Windows NT architecture allows multiple buses of the same type in the same system. This parameter specifies the bus number for the adapter card. The default for this parameter is 0 and in most cases should not be changed.

2.3.2 SM85 Adapter



- **Device Name**: This parameter allows the user to assign a name to identify the device. The default for this parameter is “CLMBX#”, where ‘#’ is the selected device number.
- **Polling Interval**: This parameter specifies the polling interval in milliseconds that the driver will use when running in the polled mode. The valid range for the polling interval is 20-1000 milliseconds. The default value is 20 ms.
- **Slot Number**: For all Micro Channel cards the slot number for the adapter card has to be specified. The valid slot numbers start from 1. The default for this parameter is 1.
- **Bus Number**: Windows NT architecture allows multiple buses of the same type in the same system. This parameter specifies the bus number for the adapter card. The default for this parameter is 0 and in most cases should not be changed.

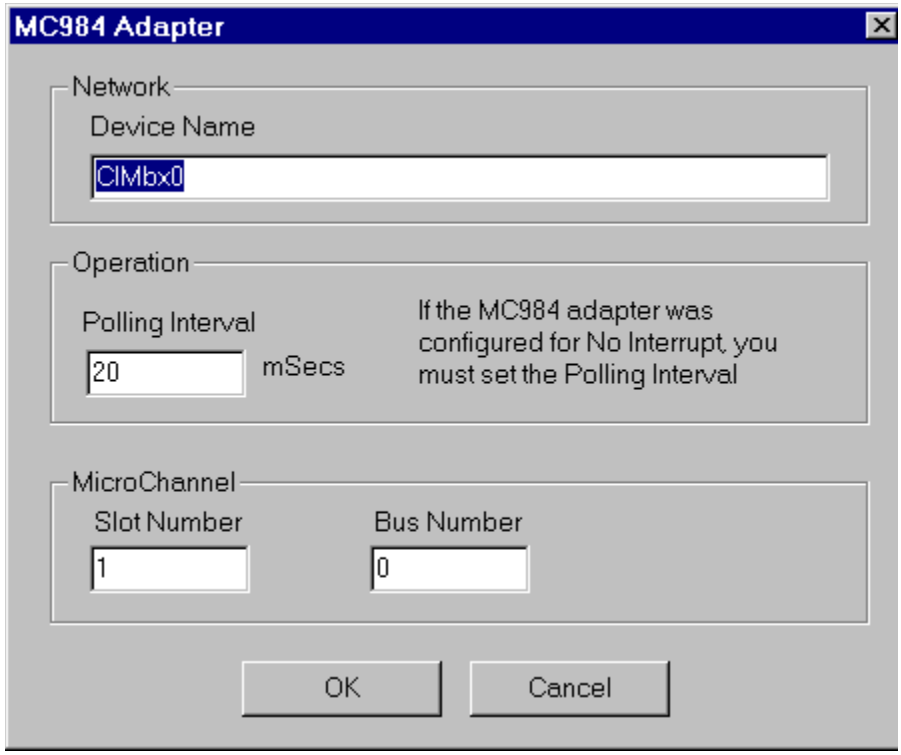
2.3.3 AT984 Adapter



- **Device Name**: This parameter allows the user to assign a name to identify the device. The default for this parameter is “CLMBX#”, where ‘#’ is the selected device number.
- **Polled mode/Interrupt driven**: The kernel mode device driver can operate in either polled mode or interrupt mode. Select the proper mode of operation depending on the adapter card configuration. (The interrupt mode provides better performance than the polled mode. However, interrupt mode requires more processor overhead.) It is legal to mix interrupt and polled modes of operation for different cards in the same system. The default for this parameter is polled mode.
- **Polling Interval**: This parameter specifies the polling interval in milliseconds that the driver will use when running in the polled mode. The valid range for the polling interval is 20-1000 milliseconds. The default value is 20 ms.

- **Memory Address**: This parameter specifies the base address of the adapter card (e.g. D0000). The default for this parameter is D0000. This address must match the switch settings on the card and has to be unique for each adapter card.
- **Interrupt IRQ Number**: When interrupt mode is selected, this parameter specifies the IRQ number for the interrupt line used (e.g. 5). The default for this parameter is 5. This IRQ number must match the IRQ setting on the card and has to be a unique value for each card in the system.
- **Bus Number**: Windows NT architecture allows multiple buses of the same type in the same system. This parameter specifies the bus number for the adapter card. The default for this parameter is 0 and in most cases should not be changed.

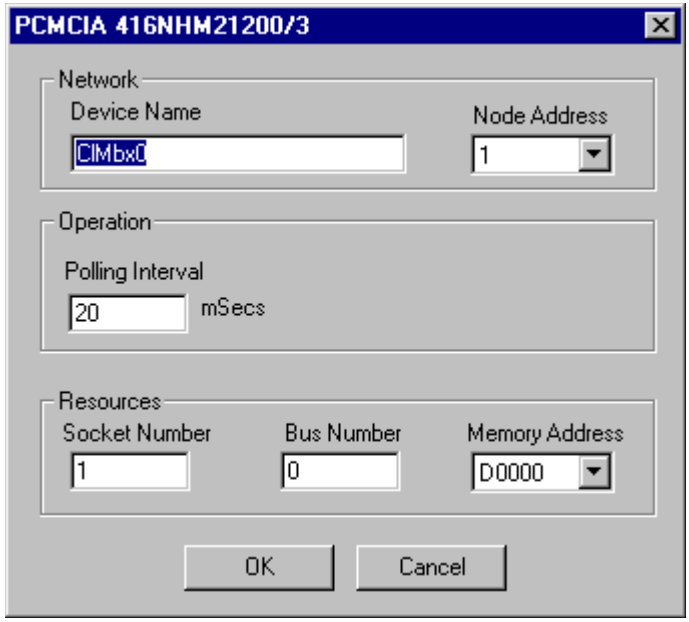
2.3.4 MC984 Adapter



- **Device Name**: This parameter allows the user to assign a name to identify the device. The default for this parameter is “CLMBX#”, where ‘#’ is the selected device number.
- **Polling Interval**: This parameter specifies the polling interval in milliseconds that the driver will use when running in the polled mode. The valid range for the polling interval is 20-1000 milliseconds. The default value is 20 ms.
- **Slot Number**: For all Micro Channel cards the slot number for the adapter card has to be specified. The valid slot numbers start from 1. The default for this parameter is 1.
- **Bus Number**: Windows NT architecture allows multiple buses of the same type in the same system. This parameter specifies the bus number for the adapter card. The default for this parameter is 0 and in most cases should not be changed.

2.3.5 PCMCIA 416NHM21200/3 Adapter

Note: The PCMCIA card should be plugged in before Windows NT is booted and it should not be removed while the system is running.



- **Device Name**: This parameter allows the user to assign a name to identify the device. The default for this parameter is “CLMBX#”, where ‘#’ is the selected device number.
- **Node Address**: This is the Modbus Plus node address for the adapter. Valid node addresses range from 1 to 64. The default for this parameter is 1.
- **Polling Interval**: This parameter specifies the polling interval in milliseconds that the driver will use when running in the polled mode. The valid range for the polling interval is 20-1000 milliseconds. The default value is 20 ms.
- **Socket Number**: The socket into which the PC Card (PCMCIA) adapter is plugged. The valid socket numbers start from 1. The default for this parameter is 1.
- **Bus Number**: Windows NT architecture allows multiple buses of the same type in the same system. This parameter specifies the bus number for the adapter card. The default for this parameter is 0 and in most cases should not be changed.
- **Memory Address**: This parameter specifies the base address of the adapter card (e.g. D0000). The default for this parameter is D0000. This address has to be unique for each adapter card.

2.4 Remote Server Configuration

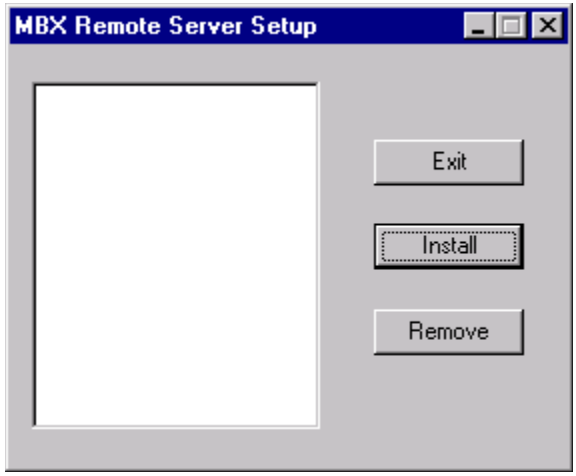
The MBX Remote Server requires a server service to be installed before it can be used. A utility program, called *MBX Remote Server Configuration,* is provided to achieve this task.

In order to start this utility, double click the *MBX Remote Server Configuration* icon in the *MBX Driver* subgroup located in the *WinConX* program group (i.e. ***Start/ Programs/WinConX/MBX Driver/MBX Remote Server Configuration***). On systems running Windows NT version 3.51, the *MBX Remote server Configuration* icon is located in the *Modicon WinConX - MBX Series* program group).

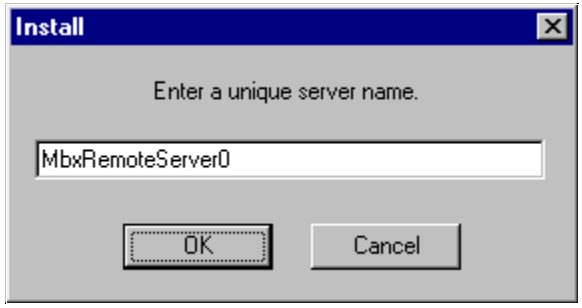
During system startup the server may detect a number of configuration problems. When a problem is detected, the server sends an appropriate message to the Windows NT *Event Logger*. Refer to *Troubleshooting the MBX Remote Server* on page 37 for more information.

Procedure **Installing an MBX Remote Server Service**

1. Run the server configuration utility (*MBX Remote Server Configuration*). The following dialog will appear:

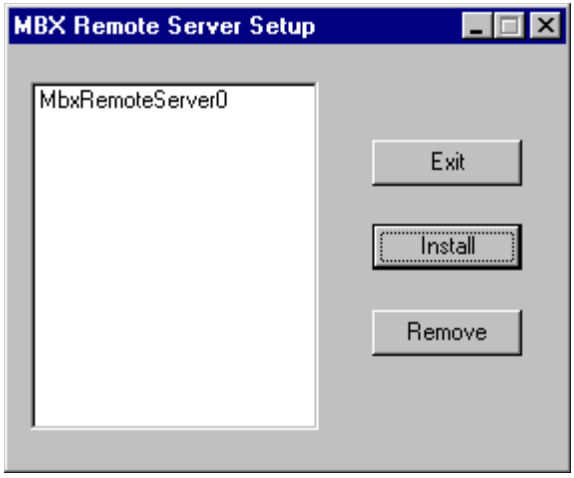


2. Click the ***Install*** button. A dialog box will be displayed showing the default server name.



3. Edit the server name as desired.

4. Click the ***OK*** and then the ***Exit*** button when finished.



The MBX Remote Server service is now installed. However, the server service is always installed in manual start-up mode. Please refer to *Starting the MBX Remote Server* on page 25 for details on starting the service.

2.5 Starting the MBX Remote Server

When a MBX Remote Server service is installed, by default, it is configured for manual startup. To use the server, it must be started manually or configured to start on system boot. Instruction on manually starting and configuring the services are listed below.

2.5.1 Manual Startup

Method 1

1. Run the ***Services*** applet from the *Control Panel*.
2. Select the service to start (WinConX MBX Server *server_name*).
3. Click the ***Start*** button to start the service.

Method 2

At a DOS command prompt, type the following: **net start *server_name*** (Note: Replace ***server_name*** with the name of the server. For example, if the server is named *MbxRemoteServer0*, the command would be: *net start MbxRemoteServer0*)

2.5.2 Automatic Startup

Method 1

1. Run the ***Services*** applet from the *Control Panel*.
2. Select the service to start (WinConX MBX Server *server_name*).
3. Click the ***Startup*** button and select the ***Automatic*** setting.
4. Reboot the system.

Method 2

1. Create a batch file and include the following command: **net start *server_name*** (*Note*: Replace ***server_name*** with the name of the server. If the server is named MbxRemoteServer0, the command would be: net start MbxRemoteServer0).
2. Include the batch file in the *Startup* program group.
3. Reboot the system.

Note: When configured for ***Automatic*** startup mode (Method 1), the server may fail to start due to a known Windows NT bug. This bug is fixed in Windows NT 4.0 Service Pack 3 and later. If you experience this problem and are running Windows NT 3.51, add the following entry in the registry using the RegEdt32 utility. (Refer to your operating system manual for further information on the registry editor).

HKEY_LOCAL_MACHINE

System

CurrentControlSet

Services

RPCLOCATOR

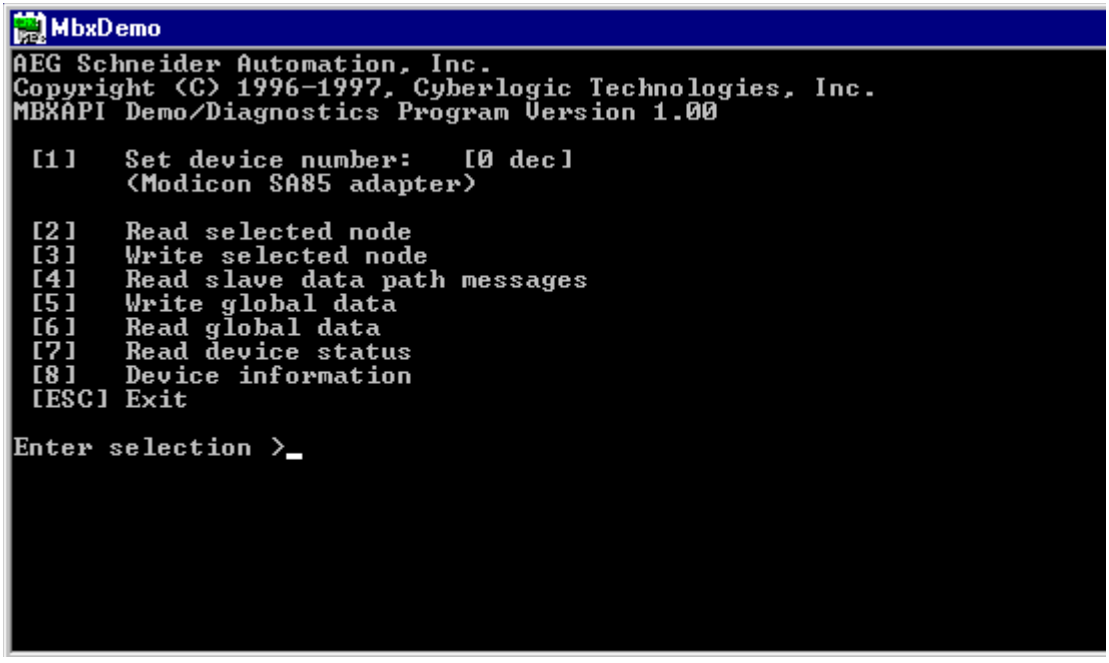
DependOnService = REG_MULTI_SZ "LanmanWorkstation" "Rdr"

2.6 Verification of the Driver Operation

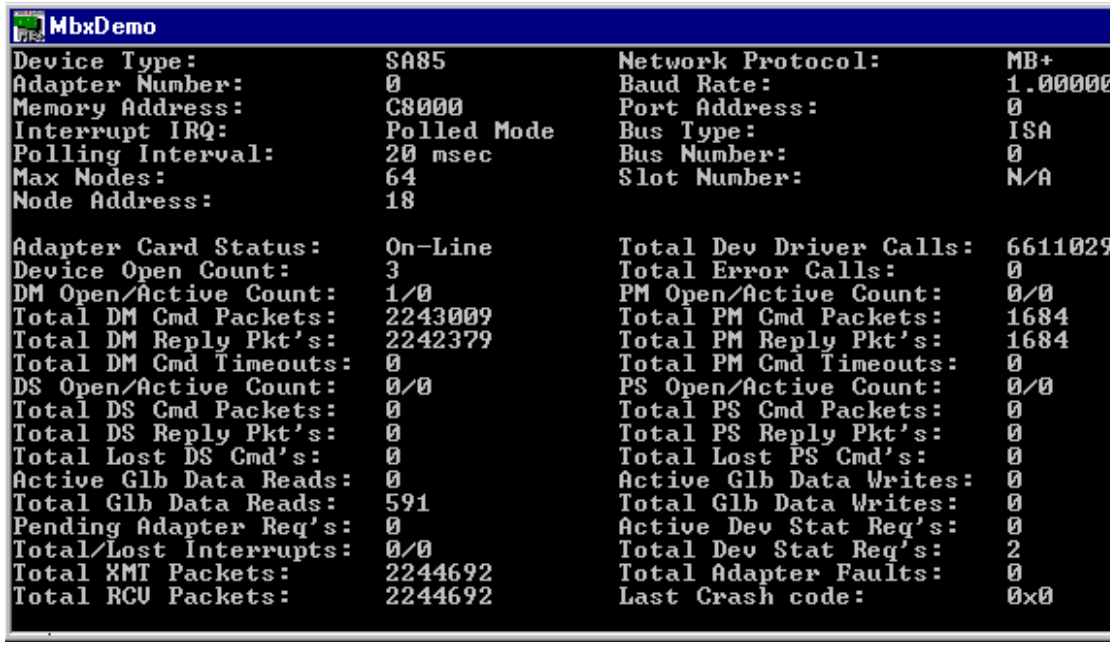
After successful installation and device configuration the user can run the 32-bit *MbxDemo* program to verify the operation of the MBX Driver.

Procedure **Running MBX Demo**

1. Click on the following: Start ‡ Programs ‡ WinCon ‡ MBX Driver ‡ MbxDemo. The initial screen of the *MbxDemo* program provides a number of selections:

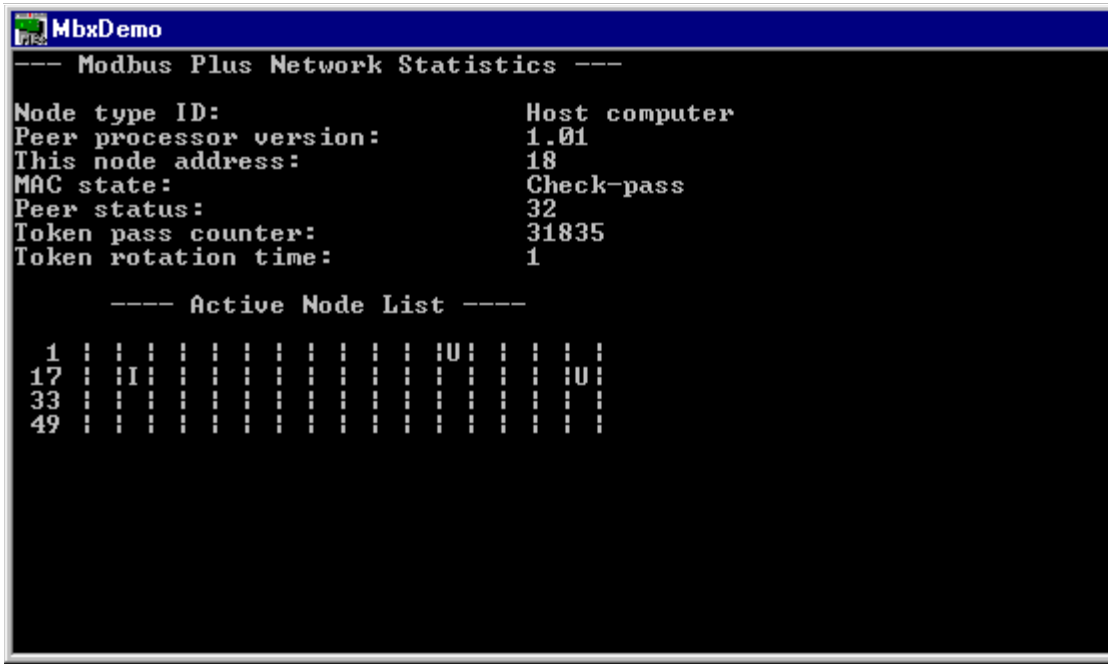


2. Select number ***[8] Device Information***. This screen shows configuration, statistical, and diagnostic information about the driver, the device, and the network.



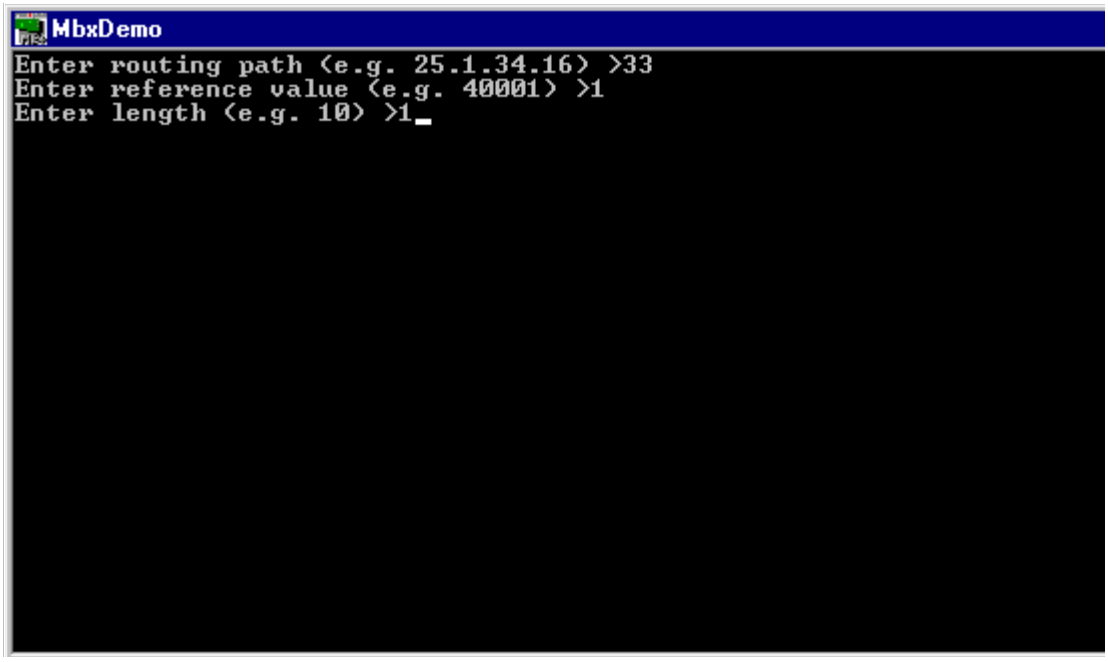
3. Hit ***<Esc>*** to get back to the main menu.

4. Select number ***[7] Read device status***. The following screen will appear. This screen should show all active nodes on the network.



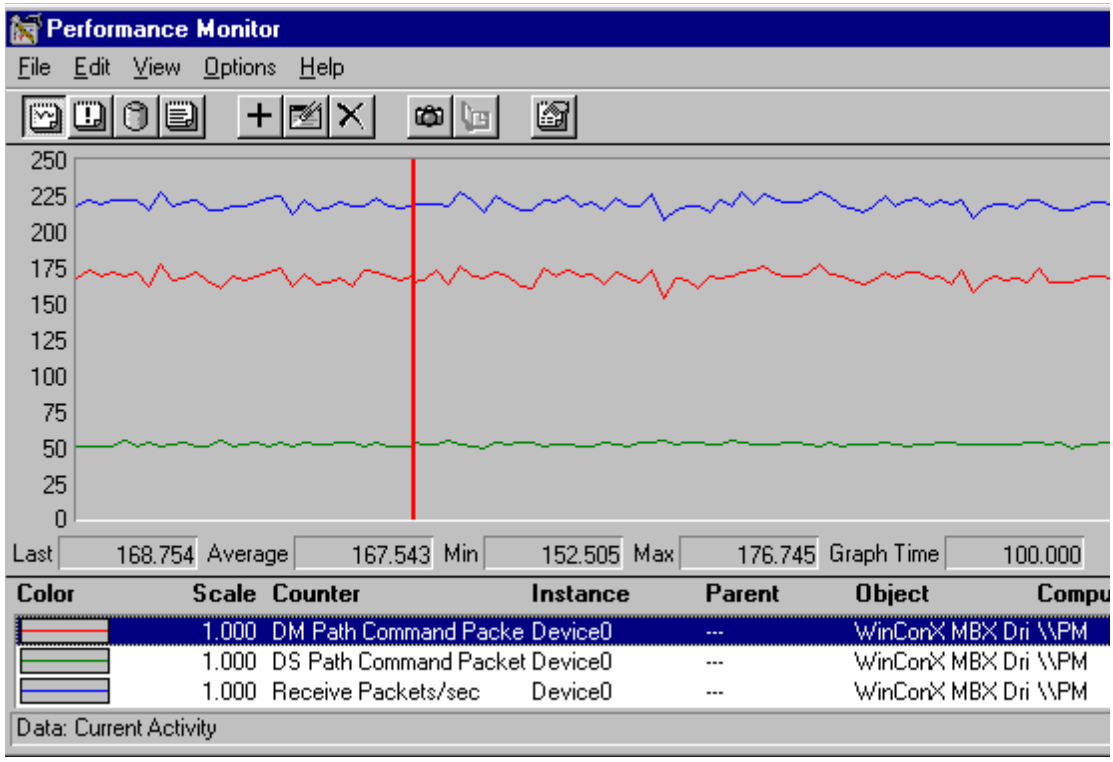
5. Hit ***<Esc>*** to get back to the main menu.

6. Select number ***[2] Read selected node***. You should be able to read data from any active node on the network.



2.7 The Performance Monitor

The *Performance Monitor* is supplied by Microsoft as part of the Windows NT operating system as a diagnostic tool. Applications that support the *Performance Monitor* allow the users to monitor relevant performance information. The MBX Driver supports the *Performance Monitor* and it should be used as a fine-tuning and diagnostic tool.



Select ***Add to Chart*** option from the ***Edit*** menu and then select ***WinConX MBX Driver*** from the ***Object*** list to display performance information.

Multiple adapter cards can be monitored simultaneously.

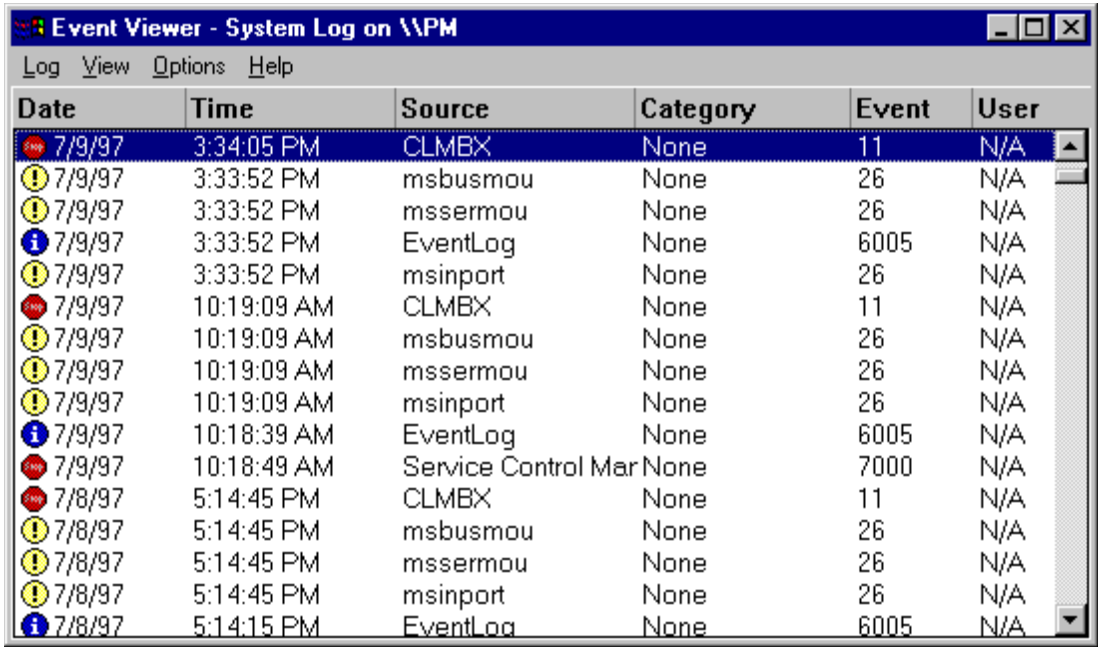
2.8 Troubleshooting the MBX Driver

2.8.1 The Event Logger

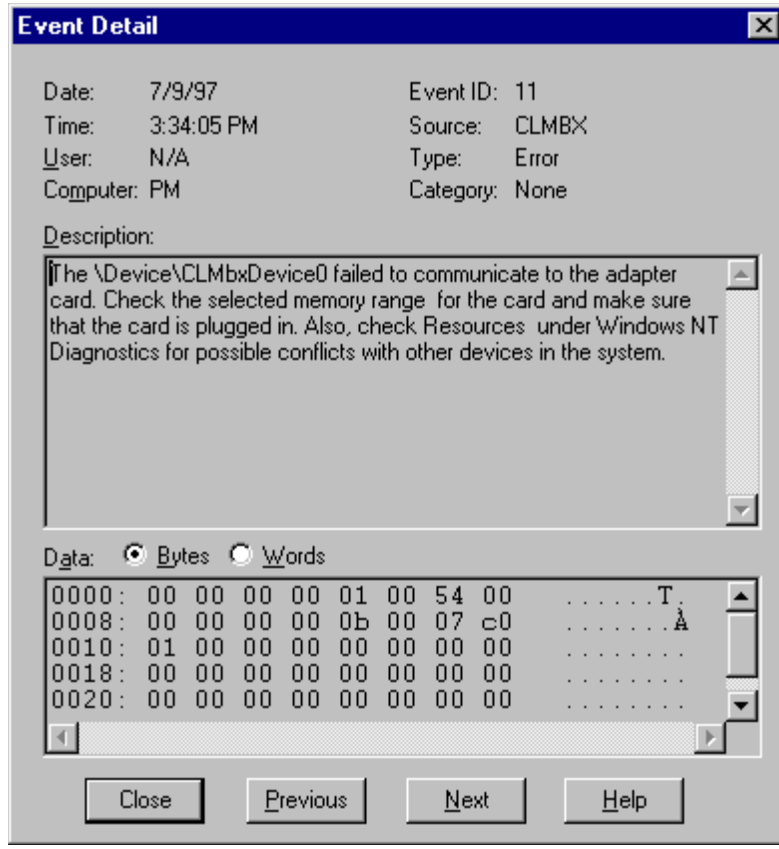
During system startup, the MBX Driver may detect a number of configuration problems. When a problem is detected, the driver sends an appropriate message to the Windows NT *Event Logger*.

Procedure **Viewing the Error Log Messages**

1. From the ***Administrative Tools*** group run *Event Viewer*.
2. Select ***System*** from the ***Log*** menu.
3. Look for entries with the CLMBX in the ***Source*** column.



4. Double click on the selected entry to display complete event message as seen below.



If the *Event Viewer* shows no error messages, you can run the *MbxDemo* program provided in the installation directory to verify the driver operation. Refer to *Verification of the Driver Operation* on page 27 for more details.

2.8.2 MBX Driver Messages

Errors

- Adapter card ID for <device name> in selected slot does not match the expected card ID.

  Invalid configuration parameter. MCA adapter cards, like SM85 and MC984, have unique ID codes used to identify them. The card inserted in the specified MCA slot did not match the expected card ID.

- Adapter card's dual-port memory diagnostics for <device name> at selected memory address failed (Pattern 0x<hex value>). Check the selected memory range for the card and make sure that the card is plugged in. Also, check Resources under Windows NT Diagnostics for possible conflicts with other devices in the system. Try another adapter card.

  May indicate a faulty card.

- Adapter card's interface diagnostics for <device name> failed (Pattern 0x<hex value>). Check the selected memory range for the card and make sure that the card is plugged in. Also, check Resources under Windows NT Diagnostics for possible conflicts with other devices in the system. Try another adapter card.

  May indicate a faulty card.

- Connecting ISR routine to selected interrupt line for <device name> failed. Some device driver in the system did not report it's resource usage. Try to remove some questionable drivers from the system and restart this driver. You may also select another interrupt line.

  Unreported interrupt already used by another device driver.

- Hardware resources allocation for device <device name> failed. Check Resources under Windows NT Diagnostics for possible conflicts with other devices in the system.

  Invalid configuration parameter. One of the requested system resources, like memory address, interrupt IRQ, has already been allocated to a different device.

- Mapping selected interrupt into system interrupt vector for <device name> failed. Check device configuration and restart the driver.

  Unreported interrupt already used by another device driver.

- Mapping selected physical memory address to logical address space for <device name> failed. Check device configuration and restart the driver.

  Unreported memory range already used by another device driver.

- Not enough memory in <paged/nonpaged> pool was available to allocate internal storage needed for <Device Name>. Close some applications. Add more memory to your system.

Memory allocation from the specified memory pool failed. This is a fatal error. The driver will not load.

- Parameter <parameter name> for device <device name> has invalid value (<parameter value>). Check device configuration and restart the driver.

  Invalid configuration parameter. This is a fatal error.

- Parameter <parameter name> for device <device name> has invalid value (End of dump data has parameter value). Check device configuration and restart the driver.

  Invalid configuration parameter. This is a fatal error.

- The bus number selected for device <device name> is not supported by this computer system.

  Invalid configuration parameter.

- The <device name> failed to communicate to the adapter card. Check the selected memory range for the card and make sure that the card is plugged in. Also, check Resources under Windows NT Diagnostics for possible conflicts with other devices in the system.

  The device driver is unable to communicate to the selected adapter card.

- The interface to the adapter card for <device name> has crashed (Crash code: 0x<hex value>). Check for possible conflicts with other devices in the system. Try another adapter card.

  May indicate a faulty card.

- The slot number in selected bus for device <device name> is not supported by this computer system.

  Invalid configuration parameter.

- Unexpected error in <function name> for <device name>. Please contact technical support of manufacturer.

  Indicates a programming bug in the device driver.

Warnings

- Device <device name> has no value for parameter <parameter name>.

  Invalid configuration parameter.

- Not enough memory in <paged/nonpaged> pool was available to allocate internal storage needed for <Device Name>. The driver may not operate correctly. Close some applications. Add more memory to your system.

  Memory allocation from the specified memory pool failed. This is only a warning. The requested operation will fail but the driver will continue to operate.

- Parameter <parameter name> for device <Device name> is out of range. Defaults to <parameter value>. Check device configuration and restart the driver.

  Invalid configuration parameter.

- This is a promotional copy of the CLMbx.sys device driver. The driver will operate for 4 hrs.

Informational

- CLMbx.sys driver version <version number>

  This is a promotional copy of the CLMbx.sys device driver. The allowed operation time has expired. The driver has been disabled.

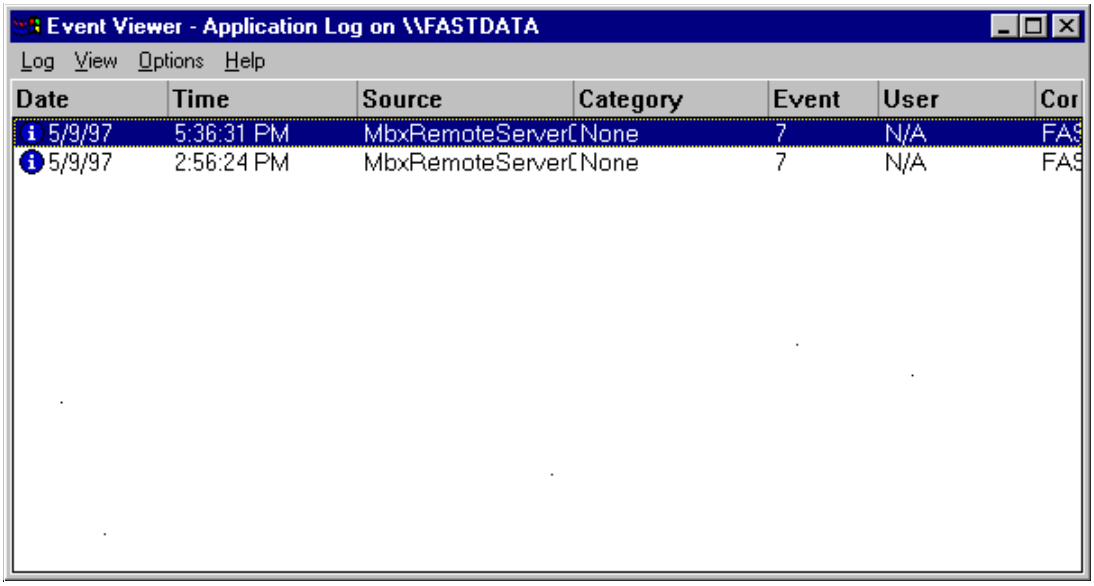
2.9 Troubleshooting the MBX Remote Server

2.9.1 How to View Error Log Messages

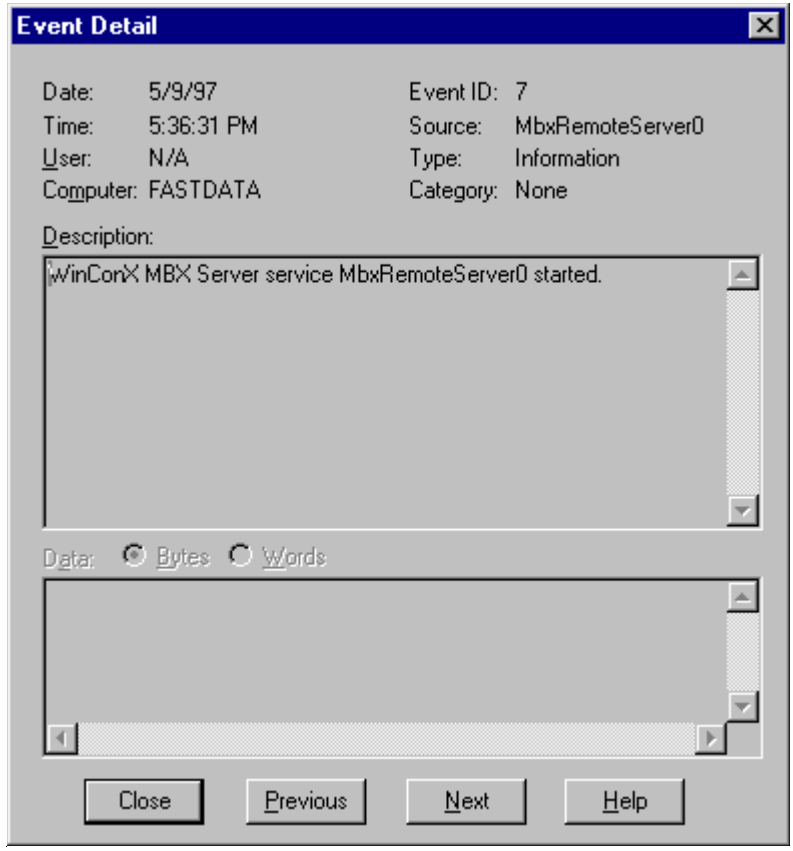
During the system startup, the MBX Remote Server may detect a number of start-up problems. When a problem is detected, the server sends an appropriate message to the Windows NT *Event Logger*.

Procedure **Viewing Error Log Messages**

1. From the ***Administrative Tools*** group run *Event Viewer*.
2. Select ***Application*** from the ***Log*** menu.
3. Look for entries with the server_name (e.g., MbxRemoteServer0) in the ***Source*** column.



4. Double click on the selected entry to display complete event message as seen below.



2.9.2 MBX Remote Server Messages

Errors

- <function name> failed - <system error message>. Data buffer contains the <data description> data.
- CYBREGEDT.DLL failed to load. Reinstall the product.
- Registration verification failed!. Reinstall the product.
- MBXRAPIM.DLL failed to load. Reinstall the product.
- Remote server of <server name> name already exists!!. Change server name.
- <function name> failed - <system error message>.

Informational

- WinConX MBX Server service <server name> started.

Crash Codes

Occasionally, due to adapter card malfunctions, the MBX Driver may detect an adapter card fault. In most cases this would be due to electrical interference (both internal and external to computer system) but it can also be an indication of genuine card failure. The MBX Driver tries to automatically recover from these failures. Every time a fault condition is detected an internal adapter fault counter is incremented as well as the last *crash code* is internally recorded. Both of these numbers can be viewed through the ***Device Information*** screen of the *MbxDemo* program. The following is a complete list of all crash codes that can aid in diagnosing these types of problems.

Crash	Symbolic Name	Error Type	Description
0x00	IFCINTACT	None	Interface operational
0x01	IFCTIMOUT	Interface	2.0-sec interface timeout
0x02	BADIFCOPC	Interface	Bad interface op-code
0x03	IFCDATERR	Interface	Interface data error
0x04	IFCTSTERR	Interface	Interface test error
0x05	IFCDONERR	Interface	Interface x-fer done error
0x06	BADIFCPTH	Interface	Bad interface path
0x07	BADXFRSVR	Interface	Bad transfer state
0x08	BADXFRLEN	Interface	Bad transfer length
0x09	GLBDATLEN	Interface	Global-data length error
0x0A	GLBDATADR	Interface	Global-data address error
0x0B	GLBDATPRS	Interface	Global-data not present
0x81	CKSUMERR	Fatal	PROM check-sum error
0x82	RAMDATERR	Fatal	Internal RAM data test error
0x83	EXTDATERR	Fatal	External RAM data test error
0x84	EXTADRERR	Fatal	External RAM address test error
0x85	BADCTINDX	Fatal	Bad confidence test index

Crash	Symbolic Name	Error Type	Description
0x86	EXT0EVENT	Fatal	External *int0* event error
0x87	EXT1EVENT	Fatal	External *int1* event error
0x88	DMA0EVENT	Fatal	DMA *int0* event error
0x89	COMMEVENT	Fatal	Comm-int event error
0x8A	XMTNGEVNT	Fatal	Xmit-no-good event error
0x8B	RSPTOSVAR	Fatal	No-response timeout MAC-state
0x8C	RSPTOIDLE	Fatal	No- response timeout MAC-idle
0x8D	RCVOKSVAR	Fatal	Receive-ok MAC-state
0x8E	XMTOKSVAR	Fatal	Transmit-ok MAC-state
0x8F	NORCVBUF	Fatal	No receive buffer free
0x90	BADINXLEN	Fatal	Bad input-transfer length
0x91	RESBUFERR	Fatal	Reserved rcv-buf error
0x92	BADTCSVAR	Fatal	Bad trans-control state
0x93	BADWRKREQ	Fatal	Bad work request bit
0x94	OVFDATQUE	Fatal	Node-queue overflow
0x95	BADDATQUE	Fatal	Bad data-queue overflow
0x96	NOPATHERR	Fatal	Empty data-path error
0x97	BADPTHINX	Fatal	Bad path search index
0x98	BADDSPATH	Fatal	Bad data-slave path
0x100			Uncontrolled adapter crash
0x101			Adapter card initialization fault
0x102			Adapter card software reset fault
0x103			Adapter I/O timeout fault

2.10 Frequently Asked Questions

Helpful Hints

- We suggest that after you install the MBX Driver software, you run the *MbxDemo* program to ensure that the driver is configured and running properly. See *Verification of the Driver Operation* on page 27 for more details.
- If you are experiencing problems with performance, make sure both the adapter card and the driver are set up for *polled mode* or both are set up for *interrupt mode*.
- In polled mode the recommended rate to use for optimum performance is 20 millisecond polling rate. The interrupt mode of operation will provide higher message rate at the expense of higher CPU load. Low-end systems (such as 486 based systems) may provide a better overall performance with adapter cards configured to run in polled mode.
- Make sure there are no address conflicts with the board address. The *Windows NT Diagnostics* application that comes with Windows NT (found in the *Administrative Tools* group) may aid in detecting hardware conflicts.
- Make sure that you are communicating through the right adapter card.

Frequently Asked Questions

- I've installed the software. What's next?

  The next step is to configure a device. You'll need to know the memory address and interrupt, if any, the card will use. See *MBX Driver Configuration* on page 12 for more details.

- I've configured my device, but when the system boots up I get a message saying a service failed. What does that mean?

  An error has occurred during the loading of the driver. Use the *Event Viewer* to find out what error occurred.

- The Event Viewer has some error messages from CLMBX. How do I fix that?

  The two most common errors are the results of either a conflict with another device or the driver configuration not matching the card configuration. Verify that the card's memory address matches the address the driver is using. Also compare the state of the polled mode/interrupt mode jumper on the card with the polled mode/interrupt mode setting of the driver.

If the configurations match, there may be a conflict in the system or the card may be faulty. If possible, try a card that is known to be good in the system with the same settings. If errors still occur, try setting the card to polled mode and moving to a new memory address (C8000, D0000, D4000, and D8000 are usually good addresses to try). Make sure you change both the driver and card settings. [D0000 is sometimes used by some disk controller boards.]

Note that the *Event Viewer* doesn't clear itself after rebooting. Check the time-stamps of the messages to make sure that you aren't looking at an old error message.

- There might be a conflict with my adapter. What do I do?

  Try setting the card to polled mode and moving to a new memory address (C8000, D0000, D4000, and D8000 are usually good addresses to try). Make sure you change both the driver and card settings.

- When I configure an adapter, should I use polled mode or interrupt mode?

  We recommend polled mode. Interrupt mode gives you slightly higher performance, but it puts a greater load on the CPU. Also, you then have to find a free interrupt and worry about interrupt conflicts. For the majority of applications, running in polled mode with a 20 millisecond polling interval will provide sufficient throughput. Whichever mode you choose, make sure the jumper setting on the card matches the driver setting.

- The card seems to be working, but I can't see one of the nodes on the network. What's wrong?

  There are two things to check. First, make sure that the card is plugged into the network. Second, it's likely that two nodes both have the same network node address. Shutdown the system, change the card's network address by changing the DIP switch settings (refer to the *"Modicon IBM Host Based Devices User's Guide"* from Schneider Automation), and then restart the system. You should now be able to see all the nodes.

- I have two devices configured in the system. How do I communicate through the second one?

  MbxDemo uses the device number to determine which card to use. Option 1, *Set device number,* lets you choose which device the demo will use. If you are using some other software, contact the manufacturer for more information on using multiple devices.

- I have a notebook with a docking station. I installed an SA85 adapter card in the docking station and a PCMCIA adapter card in the main unit. I can not get both of these cards to work together. What's wrong?

  The PCMCIA socket requires a 4K memory window to operate. Your SA85 and PCMCIA cards can not share the same 4K memory page. For example, if your SA85 is configured at C8000 address then the PCMCIA card can not be placed C8800 address (although this would have been OK for another SA85 card). Instead select C9000 (or higher) address for the PCMCIA card.

Virtual MBX Driver

3

3.1 Introduction

The Virtual MBX Driver for the Microsoft Windows NT operating system allows all existing 16-bit DOS or Windows 3.x NETLIB/NetBIOS-compatible applications to run under Windows NT in their original binary form. This includes programs such as ModSoft, ModLink, MBPSTAT, and hundreds of other custom applications written by software developers and system integrators. All of these applications can now run concurrently with other 32-bit applications, sharing the same 32-bit device driver.

For DOS applications and Win16 applications that do not use the MBPLUS.DLL library, the Virtual MBX Driver fully emulates the operation of the old SA85.SYS and MBPHOST.SYS drivers from Modicon. Therefore, the SA85.SYS and MBPHOST.SYS drivers are no longer needed and should not be used.

The Virtual MBX Driver fully supports all NETLIB/NetBIOS features. This includes support for *Data Master/Slave*, *Program Master/Slave* and *Global Data*. The Virtual MBX Driver uses the MBX Driver, and its high performance 32-bit kernel mode device driver, for all of its local I/O functions. It can also use the Remote MBX Driver for network based installations. (The MBX Driver and Remote MBX Driver are purchased separately and either the MBX Driver or the Remote MBX Driver is required for the operation of the Virtual MBX Driver.)

The Virtual MBX Driver provides an excellent bridge for users who are porting their applications to Windows NT but are not ready to abandon their favorite 16-bit applications. Once the driver is installed, you can execute 16-bit programs exactly the same way as in their original environment. In addition, you can concurrently execute multiple instances of the same applications. For example, you can run multiple instances of MBPSTAT.EXE and monitor the operation of multiple Modbus Plus networks.

3.2 Installation

The Virtual MBX Driver version 4.00 is compatible with Windows NT versions 3.51 and 4.0. Differences between these versions of Windows NT require different installation procedures as listed below.

Procedure **Windows NT 3.51 Installation Procedure**

1. Exit all applications that are using MBX products like the *MBX Driver*, the *Remote MBX Driver*, or the *Virtual MBX Driver* before installing the new version. If the Virtual MBX Driver is loaded globally from the CONFIG.NT and AUTOEXEC.NT located in the System32 directory, remove references to MBP16.SYS and MBP16VEC in these files and reboot the system before proceeding with the installation.
2. Place the installation disk in the floppy drive (either Drive A: or B:).
3. Select ***Run...*** from the *Program Manager*'s ***File*** menu. Enter A:\SETUP (or B:\SETUP) and follow the on-screen instructions.
4. Configure the Virtual MBX Driver by following the steps outlined in *Virtual MBX Driver Configuration* on page 47.
5. Shut down and restart the system.

Procedure **Windows NT 4.0 Installation Procedure**

1. Exit all applications that are using MBX products like the *MBX Driver*, the *Remote MBX Driver*, or the *Virtual MBX Driver* before installing the new version. If the Virtual MBX Driver is loaded globally from the CONFIG.NT and AUTOEXEC.NT located in the System32 directory, remove references to MBP16.SYS and MBP16VEC in these files and reboot the system before proceeding with the installation.
2. Place the installation disk in the floppy drive (either Drive A: or B:).
3. Go to the *Control Panel*, select the ***Add/Remove Programs*** icon, and click the ***Install*** button. Click ***Next>*** and then ***Finish*** to install the software onto the hard drive.
4. Configure the Virtual MBX Driver by following the steps outlined in *Virtual MBX Driver Configuration* on page 47.
5. Shut down and restart the system.

3.3 Virtual MBX Driver Configuration

Microsoft Windows NT provides a Virtual DOS Machine (VDM) within the Microsoft Win32 subsystem to support real-mode and protected-mode MS-DOS applications. Windows NT launches a VDM instance as necessary to support MS-DOS or Windows 3.x applications. Several copies of the VDM may be running simultaneously without interfering with each other. The applications run as they would in a native MS-DOS/Windows 3.x environment, independently.

When the VDM is launched, it performs the following:

- allocates memory (in which to create a virtual DOS machine).
- reads the files **…\config.nt** and **…\autoexec.nt**;
- loads drivers and executes batch files as specified;
- loads the application and executes it.

The *config.nt* and *autoexec.nt* files located in the *\system32* directory are global for the entire Windows NT system. If you use a PIF file (shortcut under NT 4.0) to launch your application, you can specify your private versions of these files which may be located in any directory.

In order to run the Virtual MBX Driver with your application, the *config.nt* file must contain the following line:

device=%SystemRoot%\System\mbp16.sys

The Virtual MBX Driver also needs to be configured, by adding the following line to the *autoexec.nt* or typed from the command prompt:

%SystemRoot%\System\mbp16vec {*NetBIOS_vector*}

The *NetBIOS_vector* identifies the software interrupt vector (hex) used by the NetBIOS interface. If not specified, the default value, 5C, is used. Once the above lines are executed, the Virtual MBX Driver is fully installed.

Although 16-bit DOS applications do not require a PIF file to run, we recommend that all 16-bit applications use PIF files. This allows the user to configure each VDM for a specific application using different *autoexec.nt* and *config.nt* files. A 16-bit Windows application, like Concept, may need the Virtual MBX Driver loaded globally. In this case, insert the lines mentioned above to the global *config.nt* and *autoexec.nt* files located in the *system32* directory and restart the system.

The Virtual MBX Driver Setup program installs example PIF files (shortcuts under NT 4.0) and sample versions of the *autoexec.nt* and *config.nt* files containing the above commands. These examples can be duplicated and modified to run any 16-bit application.

PIF Files in Windows NT 3.51

PIF files in Windows NT 3.51 are used to run 16-bit applications and can be run from either the command prompt or from an icon. PIF files are created and modified using the PIF Editor provided by Microsoft in the ***MAIN*** program group. Icons can be created to run the PIF files using the ***New*** command in the ***File*** menu of the Program Manager. When prompted, select ***Program Item*** and ***Browse*** for the corresponding PIF file.

PIF Files in Windows NT 4.0

Windows NT 4.0 also uses PIF files. However, these files are created when a Shortcut is created for a 16-bit application. The PIF file is modified by right-clicking on the shortcut icon and selecting ***Properties***.

Remote MBX Driver

4

4.1 Introduction

The Remote MBX Driver for Microsoft Windows NT operating system provides remote connectivity for applications running on client nodes to access Modbus Plus networks from remote locations via standard LANs.

The Remote MBX Driver provides access to remote client nodes over any Windows NT-compatible computer network, enabling the remote client nodes to access the (configured) MBX Driver devices residing on server nodes running the MBX Remote Server (part of the MBX Driver for Windows NT, purchased separately). However, a host interface adapter, such as Schneider Automation's SA85 card, is not required on the client node. The Remote MBX Driver provides complete MBX functionality to the client node, including support for *Data Master/Slave*, *Program Master/Slave* and *Global Data*. Any node on the network can be configured as a client to a number of remote servers and at the same time communicate to its local Modbus Plus networks.

The Remote MBX Driver provides compatibility with applications written to utilize its high performance application programming interface (API) as well as the industry standard NETLIB interface specification from Schneider Automation. The 32-bit NETLIB compatibility provides an excellent bridge for developers who would like to port their 16-bit, NETLIB-compatible applications to Windows NT. Developers of new applications can use either the NETLIB or the high performance MBXAPI programming interface. To obtain the MBXAPI software development kit, including the MBXAPI specification, MBXAPI sample source code, and NETLIB sample source code, contact Schneider Automation or Cyberlogic. For complete reference of all NETLIB library functions refer to "*Modicon IBM Host Based Devices User's Guide*" from Schneider Automation (Order# 890 USE 102 00).

The performance of the Remote MBX Driver, along with other diagnostic information, can be monitored through the Performance Monitor utility that is provided by Microsoft with Windows NT. Performance information from multiple interface cards can be monitored simultaneously.

Running 16-Bit Software

A companion product, the Virtual MBX Driver, will allow all 16-bit NETLIB/NetBIOS-compatible applications (e.g., ModSoft) to run concurrently with all 32-bit applications in the same computer. The Virtual MBX Driver will allow multiple 16-bit applications as well as multiple instances of a single 16-bit application to run under the 32-bit Windows 95 operating system. (The Virtual MBX Driver is purchased separately and it requires either the MBX Driver or the Remote MBX Driver to operate).

4.2 Installation

The Remote MBX Driver version 4.00 is compatible with Windows NT versions 3.51 and 4.0. Differences between these versions of Windows NT require different installation procedures as listed below.

Procedure **Windows NT 3.51 Installation Procedure**

1. Exit all applications that are using MBX products like the *MBX Driver*, the *Remote MBX Driver*, or the *Virtual MBX Driver* before installing the new version. If the Virtual MBX Driver is loaded globally from the CONFIG.NT and AUTOEXEC.NT located in the System32 directory, remove references to MBP16.SYS and MBP16VEC in these files and reboot the system before proceeding with the installation.
2. Place the installation disk in the floppy drive (either Drive A: or B:).
3. Select ***Run...*** from the *Program Manager*'s ***File*** menu. Enter A:\SETUP (or B:\SETUP) and follow the on-screen instructions.
4. Configure the *Remote MBX Driver* using the *MBX Device Configuration* icon in the *Modicon WinConX - MBX Series* program group.
5. Shut down and restart the system.

Procedure **Windows NT 4.0 Installation Procedure**

1. Exit all applications that are using MBX products like the *MBX Driver*, the *Remote MBX Driver*, or the *Virtual MBX Driver* before installing the new version. If the Virtual MBX Driver is loaded globally from the CONFIG.NT and AUTOEXEC.NT located in the System32 directory, remove references to MBP16.SYS and MBP16VEC in these files and reboot the system before proceeding with the installation.
2. Place the installation disk in the floppy drive (either Drive A: or B:).
3. Go to the *Control Panel*, select the ***Add/Remove Programs*** icon, and click the ***Install*** button. Click ***Next>*** and then ***Finish*** to install the software onto the hard drive.
4. Configure the Remote MBX Driver by using the *MBX Device Configuration* shortcut icon in the *Remote MBX* subgroup located in the *WinConX* program group (ie. **Start/Programs/WinConX/Remote MBX/MBX Device Configuration**).
5. Shut down and restart the system.

4.3 Remote MBX Driver Configuration

The Remote MBX Driver is designed to operate in remote client nodes without a need for physical host interface adapters. Therefore, the Remote MBX Driver configuration involves creation and configuration of logical devices (as opposed to physical devices, such as an SA85 card). The configuration of the Remote MBX Driver is similar to the configuration of the MBX Driver, except where the MBX Driver deals with physical host interface adapters, the Remote MBX Driver will refer to logical devices.

The Remote MBX Driver requires a number of parameters to be set for each device. A utility program, called *MBX Device Configuration*, is provided to create and edit devices and their configuration.

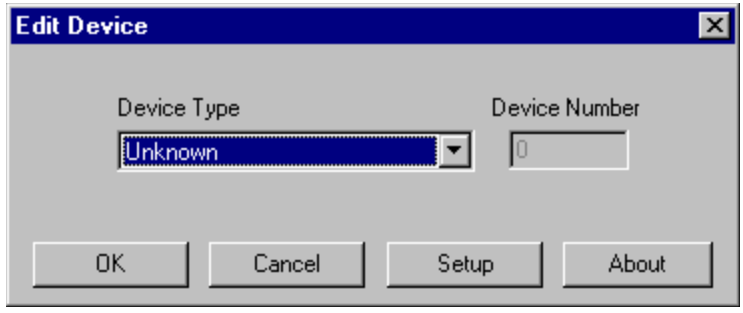
In order to start this utility program, double click the *MBX Device Configuration* icon in the *Remote MBX* subgroup located in the *WinConX* program group (ie. **Start/Programs/WinConX/Remote MBX/MBX Device Configuration**). On systems running Windows NT version 3.51, the *MBX Device Configuration* icon is located in the *Modicon WinConX - MBX Series* program group).

4.3.1 First Time Configuration

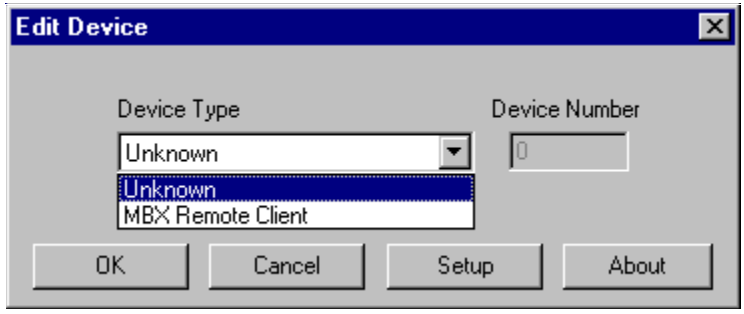
Procedure **First Time Configuration**

1. Run the device configuration utility (MBX Device Configuration). The following dialog will appear:

2. Click the ***New*** button. A dialog box will be displayed showing *Device Number* and *Device Type*.



3. Select the *Device Type* using the pull-down window. The Device Number refers to the number that you assign to every device installed in your system. **This is not the Modubs Plus node address!** By default, the configuration utility will try to use consecutive numbers fro the devices, starting from zero. However, this is not a requirement and the user can assign numbers that are not consecutive.



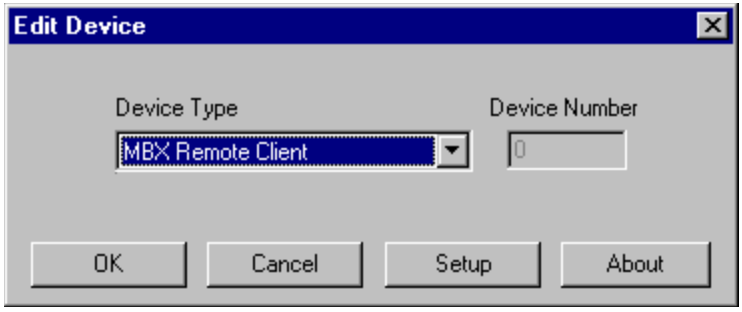
4. Click the ***Setup*** button to configure the device or ***OK*** button when finished.

4.3.2 Editing Current Configuration

Procedure **Editing Current Configuration**

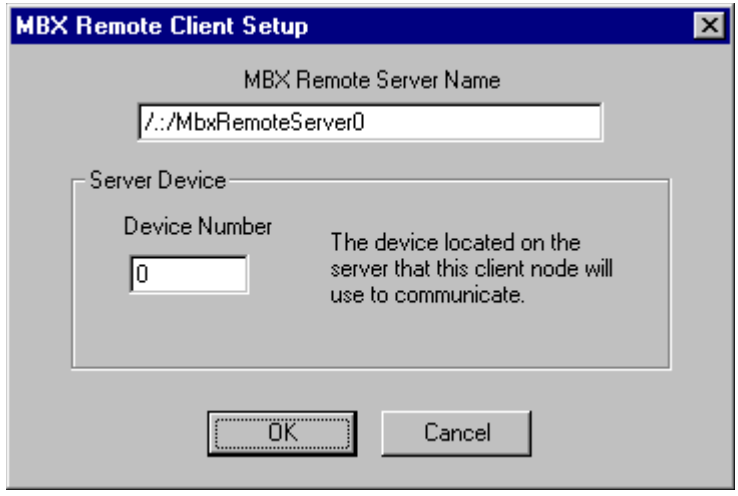
1. Run the device configuration utility (MBX Device Configuration). The following dialog will appear:

2. Select the device to be configured from the list and click the ***Edit*** button. Verify the *Device Type* and click the ***Setup*** button.



3. A dialog box will be displayed showing the parameters that need to be configured for that *Device Type*. All parameters are listed below by *Device Type*.

4.3.3 MBX Remote Client



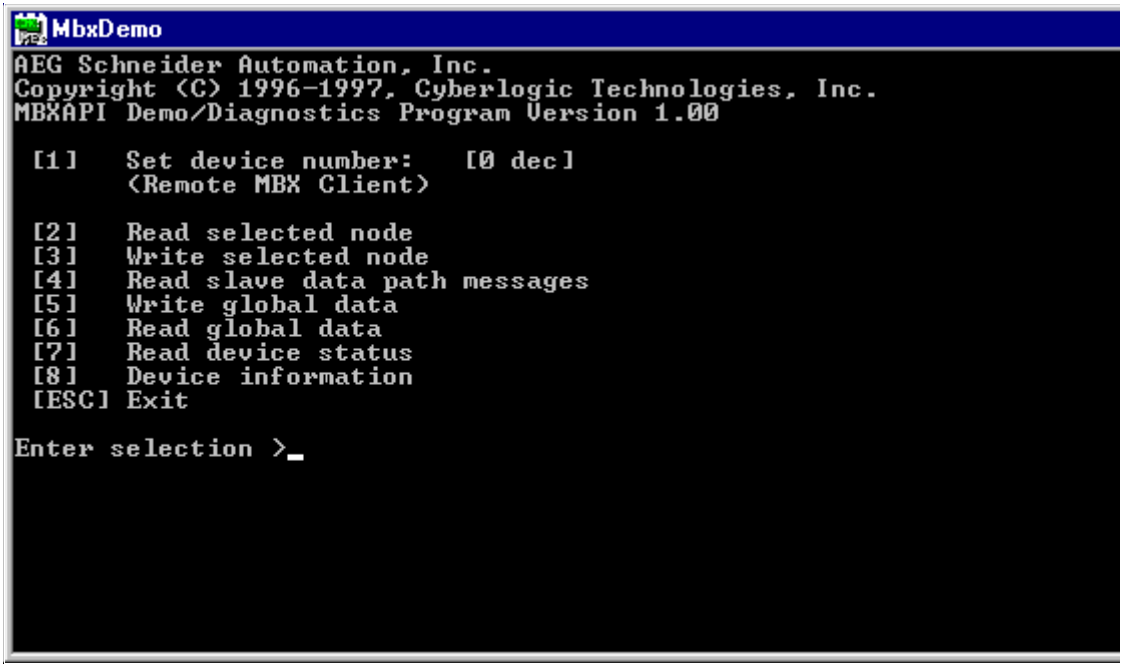
- **MBX Remote Server Name**: The name of the MBX Remote Server with which this client should connect. The name can take one of two forms. The first form (/.:/ServerName) is used when the server and client are in the same workgroup or domain. The second form (/…/ ServerDomainOrServerWorkgroupName/ServerName) is used when the server and client are in different workgroups or domains.
- **Device Number**: Designates which device on the server the client will use. As an example, assume a server has two host adapters: Device 0 – SA85, device 1 – AT984. If the client is configured to use Server Device Number 0, the server's SA85 card will be accessed. Using Server Device Number 1 accesses the server's AT984.

4.4 Verification of the Driver Operation

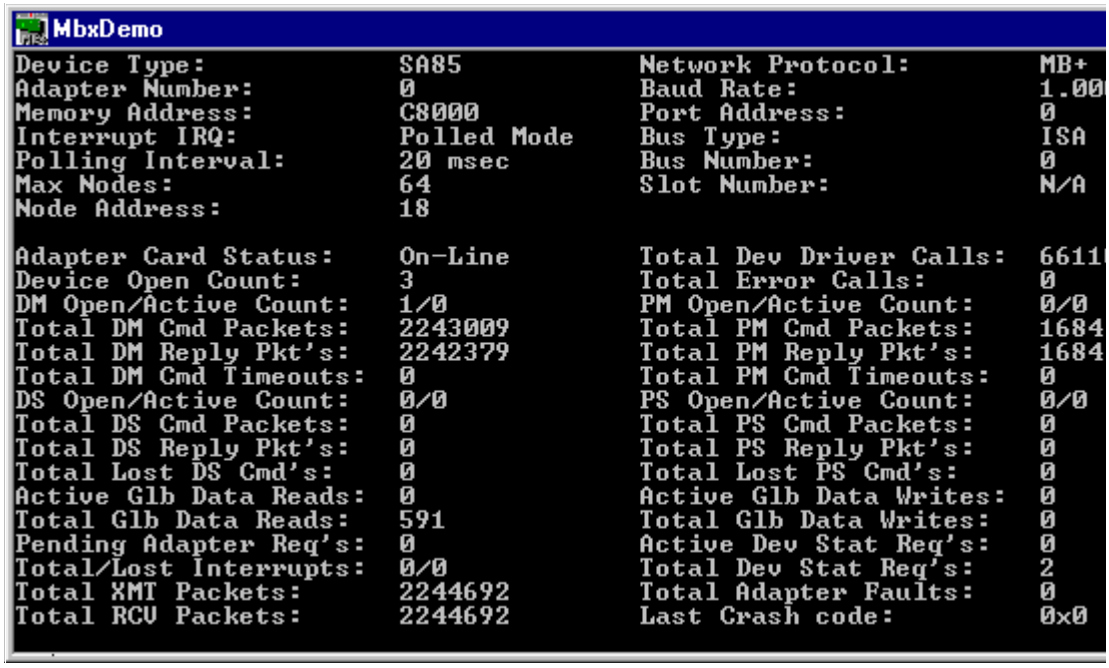
After successful installation and device configuration, the user can run the 32-bit *MbxDemo* program to verify the operation of the Remote MBX Driver.

Procedure **Verifying the Operation of the Remote MBX Driver**

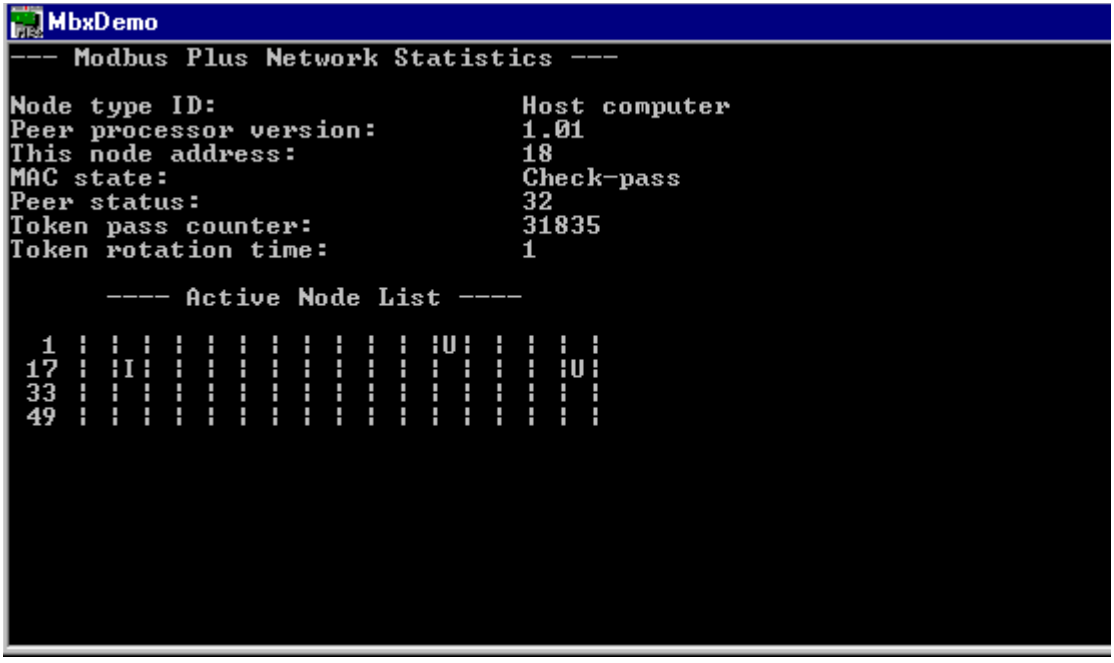
1. The initial screen of the *MbxDemo* program provides a number of selections. Select number ***[8] Device Information***.



2. This screen shows configuration, statistical, and diagnostic information about the driver, the device, and the network from the server's view. Hit ***<Esc>*** to get back to the main menu.

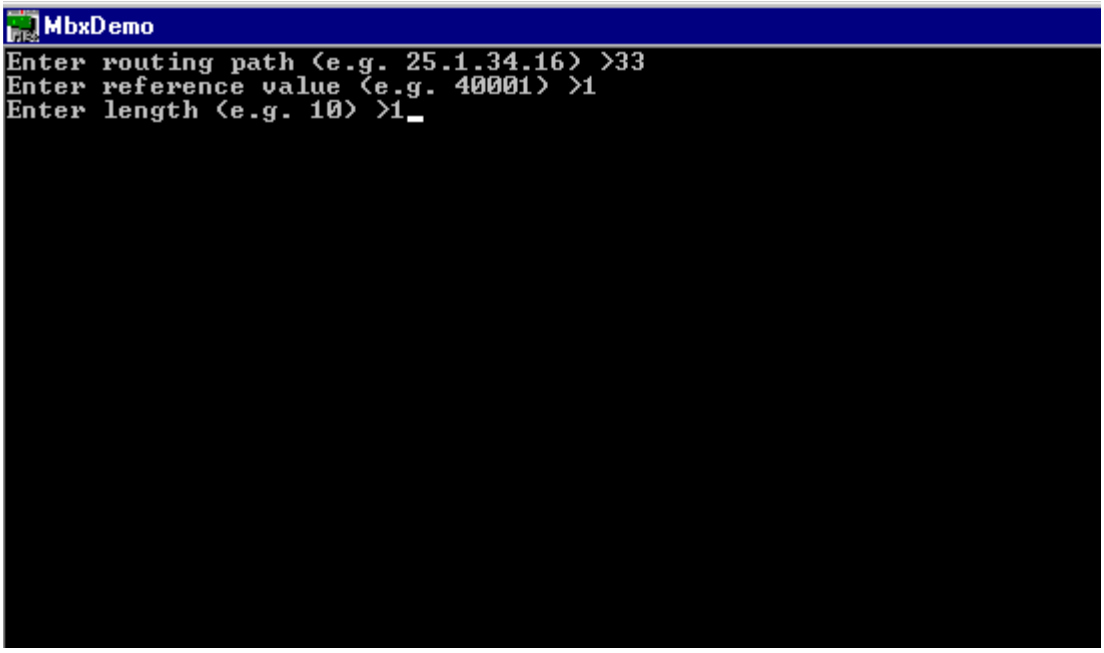


3. Select number ***[7] Read device status***. The following screen will appear. This screen should show all active nodes on the network.



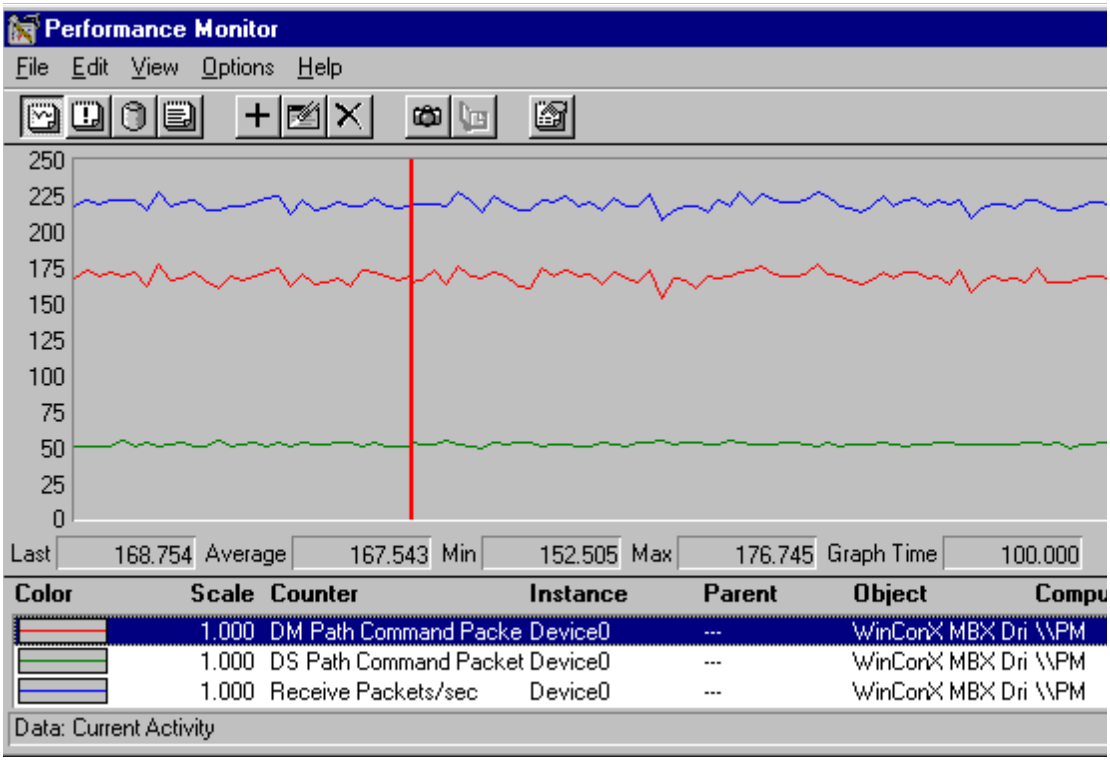
4. Hit ***<Esc>*** to get back to the main menu.

5. Select number ***[2] Read selected node***. You should be able to read data from any active node on the network.



4.5 The Performance Monitor

The *Performance Monitor* is supplied by Microsoft as part of the Windows NT operating system as a diagnostic tool. Applications that support the *Performance Monitor* allow the users to monitor relevant performance information. The Remote MBX Driver supports the *Performance Monitor* and it should be used as a fine-tuning and diagnostic tool.



Select ***Add to Chart*** option from the ***Edit*** menu and then select ***WinConX MBX Driver*** from the ***Object*** list to display performance information.

Multiple adapter cards can be monitored simultaneously.

4.6 Frequently Asked Questions

- I've installed the software. What's next?

  The next step is to configure a device. See the *Remote MBX Driver Configuration* on page 51 for more details.

- I have two devices configured in the system. How do I communicate through the second one?

  MbxDemo uses the device number to determine which card to use. Option 1, *Set device number,* lets you choose which device the demo will use. If you are using some other software, contact the manufacturer for more information on using multiple devices.

- I have to communicate to two server nodes, each located in a separate physical subnet. Both subnets are connected through a bridge. Can I communicate to both nodes?

  As part of the device configuration, the user specifies the name of the locator node. The locator can only locate server nodes that are part of the same physical subnet. Since only one locator node can be specified per client system, only one of the server nodes will be accessible (the one with the locator node).

The 32-bit Modbus Plus Software Development Kit (SDK)

A

A.1 Overview

The Modbus Plus Software Development Kit (separately available from Schneider Automation, Part #SW-LNET-SDK) contains the 32-bit NETLIB Library. This provides an excellent bridge for developers who want to port their 16-bit applications to Windows 95 and Windows NT. All you have to do is recompile your application with a 32-bit C compiler and link it with the 32-bit NETLIB.LIB library. (Use the NETLIB.H and NETBIOS.H include files provided in the ..\RemoteNetlibLibrary directory, instead of the Schneider Automation equivalent files.)

A.2 Linking Existing Applications

To demonstrate the above capability, we have recompiled and linked Schneider Automation's original 16-bit test programs with the 32-bit NETLIB library and provided them in the ..\RemoteNetlibLibrary directory. (Schneider Automation's test programs are distributed with Schneider Automation's DOS/Windows/OS/2 versions of their SA85 drivers). No changes have been made to these programs other than some minor variable declaration and include file adjustments to satisfy the Microsoft 32-bit compiler.

A.3 Developing New Applications

The 32-bit NETLIB library can also be used for new applications that are targeted for Windows 95 and Windows NT. The library is implemented to comply with and take full advantage of the multi-threaded features of Windows 95 and NT.

A.4 NETLIB Library Functions

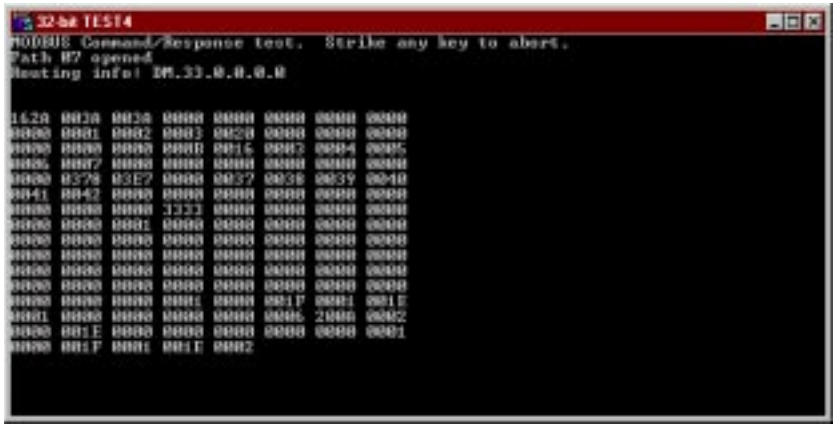
For a complete reference of all NETLIB Library functions, refer to Modicon IBM Host Based Devices User's Guide (Part# 890 USE 102 00).

A.5 System Requirements

Microsoft's Visual C++ (Version 4.2 or greater), or Borland C++ (Version 5.1 or greater), must be used when compiling. The Software Development Kit Version 4.0 is compatible with MBX drivers Version 4.0.

A.6 Software Development Kit

The following section uses Schneider Automation's TEST4.C demo program to show the minimal amount of effort required to convert from the original 16-bit to a 32-bit version that will run with our Local NETLIB Library.



A.6.1 Include Files

As part of the SDK we provide two include files, NETBIOS.H and NETLIB.H, that should be used in place of the 16-bit versions of these files.

NETBIOS.H

```
/*====================================================================
*            Copyright (C) 1994, CYBERLOGIC Technologies Inc.
*====================================================================
*
* Module Name :
*      netbios.h
*
* Abstract:
*      This file should be included by every module that makes calls to
*      functions in the netlib.dll interface library file.
*
*
* ---------+-----+-------------------------------------------------------
*   DATE   | BY  |  DESCRIPTION / REASON FOR MODIFICATION
* ---------+-----+-------------------------------------------------------
* 03-09-94 | PM  |  Start of development
* ---------+-----+-------------------------------------------------------
* ..-..-94 |     |
*==========+=====+=======================================================*/
#define NETBIOS_INCLUDED
//
// Structure of Network Control Block (NCB)
//
#ifndef NCB_INCLUDED
#define NCB_INCLUDED
#ifdef __cplusplus
extern "C" {
#endif
typedef struct _NCB
{
   UCHAR   NCB_COMMAND;          // Command
   UCHAR   NCB_RETCODE;          // Function return code
   UCHAR   NCB_LSN;              // Local session number
   UCHAR   NCB_NUM;              // Number of network name
   PUCHAR  NCB_BUFFER;           // Far pointer to message buffer
   USHORT  NCB_LENGTH;           // Length of message buffer
   UCHAR   NCB_CALLNAME[16];     // Name of session user is talking to
   UCHAR   NCB_NAME[16];         // User's network name
```

```
   UCHAR   NCB_RTO;              // Receive time-out in 500 ms. incrs.
   UCHAR   NCB_STO;              // Send time-out - 500 ms. increments
   void  (*NCB_POST_ADDRESS) (struct _NCB_ *);// Address of "no-wait" interrupt call
   UCHAR   NCB_LANA_NUM;         // Adapter number (must be 0 or 1)
   UCHAR   NCB_CMD_CPLT;         // Command completion status
   UCHAR   NCB_RESERVE[14];      // Reserved area for Token-Ring
} NCB,*PNCB;
#else
//
// _NCB structure is already defined in nb30.h file.
// Microsoft uses lower case while Schneider Automation uses upper case letters
// for all names in this structure. The caller can use either convention.
//
#define NCB_COMMAND       ncb_command
#define NCB_RETCODE       ncb_retcode
#define NCB_LSN           ncb_lsn
#define NCB_NUM           ncb_num
#define NCB_BUFFER        ncb_buffer
#define NCB_LENGTH        ncb_length
#define NCB_CALLNAME      ncb_callname
#define NCB_NAME          ncb_name
#define NCB_RTO           ncb_rto
#define NCB_STO           ncb_sto
#define NCB_POST_ADDRESS  ncb_post
#define NCB_LANA_NUM      ncb_lana_num
#define NCB_CMD_CPLT      ncb_cmd_cplt
#define NCB_RESERVE       ncb_reserve
#endif
//
// NetBIOS functions list - "WAIT" calls wait until command completes
// while the others jump to the routine in NCB_POST when the NetBIOS
// command completes and does an interrupt.
//
#define RESET                 0x32   // Reset adapter card and tables
#define CANCEL                0x35   // Cancel command. NCB_BUFFER = cmd.
#define STATUS                0xb3   // Status information for adapter
#define STATUS_WAIT           0x33   //
#define TRACE                 0xf9   // Token-Ring protocol trace
#define TRACE_WAIT            0x79   //
#define UNLINK                0x70   // Unlink from IBM Remote Program
#define ADD_NAME              0xb0   // Add name to name table
#define ADD_NAME_WAIT         0x30   //
#define ADD_GROUP_NAME        0xb6   // Add group name to name table
```

```
#define ADD_GROUP_NAME_WAIT       0x36   //
#define DELETE_NAME               0xb1   // Delete name from name table
#define DELETE_NAME_WAIT          0x31   //
#define CALL                      0x90   // Start session with NCB_NAME name
#define CALL_WAIT                 0x10   //
#define LISTEN                    0x91   // Listen for call
#define LISTEN_WAIT               0x11   //
#define HANG_UP                   0x92   // End session with NCB_NAME name
#define HANG_UP_WAIT              0x12   //
#define SEND                      0x94   // Send data via NCB_LSN
#define SEND_WAIT                 0x14   //
#define SEND_NO_ACK               0xf1   // Send data without waiting for ACK
#define SEND_NO_ACK_WAIT          0x71   //
#define CHAIN_SEND                0x97   // Send multiple data buffers
#define CHAIN_SEND_WAIT           0x17   //
#define CHAIN_SEND_NO_ACK         0xf2   // Send multiple buffers without ACK
#define CHAIN_SEND_NO_ACK_WAIT    0x72   //
#define RECEIVE                   0x95   // Receive data from a session
#define RECEIVE_WAIT              0x15   //
#define RECEIVE_ANY               0x96   // Receive data from any session
#define RECEIVE_ANY_WAIT          0x16   //
#define SESSION_STATUS            0xb4   // Status of all sessions for name
#define SESSION_STATUS_WAIT       0x34   //
#define SEND_DATAGRAM             0xa0   // Send un-ACKed message
#define SEND_DATAGRAM_WAIT        0x20   //
#define SEND_BCST_DATAGRAM        0xa2   // Send broadcast message
#define SEND_BCST_DATAGRAM_WAIT   0x22   //
#define RECEIVE_DATAGRAM          0xa1   // Receive un-ACKed message
#define RECEIVE_DATAGRAM_WAIT     0x21   //
#define RECEIVE_BCST_DATAGRAM     0xa3   // Receive broadcast message
#define RECEIVE_BCST_DATAGRAM_WAIT 0x23  //
#define SA85_OFF                  0x37   // Turn SA85 driver off (RESET to turn on)
#define SA85_MASK                 0x38   // Return status of mask bit
#define SA85_ION                  0x39   // Enable sa85 interrupts
#define SA85_IOFF                 0x3a   // Disable sa85 interrupts
#define SET_SLAVE_LOGIN           0x3b   // Set/clear slave login status
//
// NetBIOS error return codes - returned in NCB_RETCODE
//
#ifndef ERR_success
#define ERR_success           0      // NetBIOS command completed normally
#define ERR_bad_buffer_length 1      // Bad send or status buffer size
#define ERR_invalid           3      // invalid NetBIOS command
```

```
#define ERR_timeout            5      // Command time-out has expired
#define ERR_buffer_too_small   6      // Receive buffer not big enough
#define ERR_bad_session_num    8      // Bad value in NCB_LSN
#define ERR_no_RAM             9      // LAN card doesn't have enough memory
#define ERR_session_closed     0xa    // This session is closed
#define ERR_cancel             0xb    // Command has been closed
#define ERR_dup_local_name     0xd    // Name already exists for this PC
#define ERR_name_table_full    0xe    // Local name table is full
#define ERR_active_session     0xf    // Can't delete name - used in session
#define ERR_sess_table_full    0x11   // Local session table is full
#define ERR_no_listen          0x12   // Remote PC not listening for call
#define ERR_bad_name_num       0x13   // Bad value in NCB_NUM field
#define ERR_no_answer          0x14   // No answer to CALL or no such remote
#define ERR_no_local_name      0x15   // No such name in local name table
#define ERR_duplicate_name     0x16   // Name is in use elsewhere on net
#define ERR_bad_delete         0x17   // Name incorrectly deleted
#define ERR_abnormal_end       0x18   // Session aborted abnormally
#define ERR_name_error         0x19   // 2 or more identical names in use!
#define ERR_bad_packet         0x1a   // Bad NetBIOS packet on network
#define ERR_card_busy          0x21   // network card is busy
#define ERR_too_many_cmds      0x22   // Too many NetBIOS commands queued
#define ERR_bad_card_num       0x23   // bad NCB_LANA_NUM - must be 0 or 1
#define ERR_cancel_done        0x24   // command finished while cancelling
#define ERR_no_cancel          0x26   // Command can't be cancelled
#define ERR_busy               0xff   // Still processing command
#ifdef __cplusplus
}
#endif
#endif // NCB_INCLUDED
```

NETLIB.H

```
/*=====================================================================
*            Copyright (C) 1994, CYBERLOGIC Technologies Inc.
*=====================================================================
*
* Module Name :
*      netlib.h
*
* Abstract:
*      This file should be included by every module that makes calls to
*      functions in the netlib.dll interface library file.
*
*
* ---------+-----+-------------------------------------------------------
*   DATE   | BY  |  DESCRIPTION / REASON FOR MODIFICATION
* ---------+-----+-------------------------------------------------------
* 03-09-94 | PM  |  Start of development
* ---------+-----+-------------------------------------------------------
* ..-..-94 |     |
*==========+======+=======================================================*/
#ifndef _NETLIB_H_
#define _NETLIB_H_
//
// Define API decoration for direct importing of DLL references.
//
#ifdef _DLLEXPORT_NETLIB_
#define NETLIBAPI
#else
#define NETLIBAPI DECLSPEC_IMPORT
#endif // _DLLEXPORT_NETLIB_
#ifdef __cplusplus
extern "C" {
#endif
/*====================================================================*/
/* Include files
/*====================================================================*/
#ifndef NETBIOS_INCLUDED
#include "netbios.h"
#endif
/*====================================================================*/
/* Function prototypes
/*====================================================================*/
NETLIBAPI
int
```

```
APIENTRY
ncb_reset(
   int adaptno
   );
NETLIBAPI
int
APIENTRY
ncb_sa85off(
   int adaptno
   );
NETLIBAPI
int
APIENTRY
ncb_status(
   PNCB ncbp,
   int adaptno
   );
NETLIBAPI
int
APIENTRY
ncb_send(
   PNCB ncbp,
   int length,
   PCHAR buffer,
   UCHAR timeout
   );
NETLIBAPI
int
APIENTRY
ncb_receive_wait(
   PNCB ncbp,
   PCHAR buffer,
   UCHAR timeout
   );
NETLIBAPI
PNCB
APIENTRY
ncb_open(
   PCHAR name,
   int lan
   );
NETLIBAPI
int
```

```
APIENTRY
ncb_receive(
   PNCB ncbp,
   PCHAR buffer
   );
NETLIBAPI
int
APIENTRY
ncb_close(
   PNCB ncbp
   );
NETLIBAPI
int
APIENTRY
ncb_send_datagram(
   PNCB ncbp,
   int length,
   PCHAR buffer,
   UCHAR timeout,
   int adaptno
   );
NETLIBAPI
int
APIENTRY
ncb_receive_datagram(
   PNCB ncbp,
   int node,
   PCHAR buffer,
   UCHAR timeout,
   int adaptno
   );
NETLIBAPI
int
APIENTRY
ncb_cancel(
   PNCB ncbp
   );
NETLIBAPI
int
APIENTRY
ncb_set_slave_login(
   PNCB ncbp,
   UCHAR login_status
```

```
   );
NETLIBAPI
int
APIENTRY
ncb_set_sw_interrupt(
   int swInterrupt
   );
#ifdef __cplusplus
}
#endif
#endif //_NETLIB_H_
```

A.6.2 Source Files

The following shows both the 16-bit as well as the 32-bit source code for the TEST4.C programs. We highlighted the differences between the two versions to show the minimal effort required to do the conversion.

16 Bit Version of TEST4.C

```
/*name test4.c*/
/* Copyright (C) Schneider Automation, Inc. 1989, All Rights Reserved. */
/*
This test program uses the SA85.SYS device driver to read 125 holding
registers from the MODBUS slave which is found at the given node number.
In order to use this program, enter the following command:
   A>TEST4 12
where "12" is the slave node we want to read from.
*/
/*
include
*/
#include <stdio.h>
#include <dos.h>
#include <process.h>
#include <string.h>
#include <conio.h>
#include <signal.h>
#include "netbios.h"
#include "netlib.h"
/*
prototypes
*/
int main(int argc, char *argv[] );
void dump(int qty, unsigned int *buff);
```

```
void control_c(void);
/*
global variables
*/
char mbuffer[256];
char path[ 80 ];
int completed;
/*
main()
*/
int main( argc, argv )
int argc;
char *argv[];
{
   NCB *nd;
   int ret_val;
   int i;
   printf( "MODBUS Command/Response test. Strike any key to abort." );
   if( argc < 2 ) {
      printf( "Usage is: A>TEST4 <slave node>");
      exit( 1 );
   }
   ret_val = sscanf( argv[ 1 ], "%d", &i );
   if( ret_val != 1 || i < 1 || i > 64 ) {
      printf( "Node number must be in the range 1 to 64. " );
      exit( 1 );
   }
   if( signal( SIGINT, control_c ) == SIG_ERR ) {
      printf( "Unable to redirect Control-C." );
      exit( 1 );
   }

   sprintf( path, "DM.%d.0.0.0.0", i );
   if ((nd = ncb_open( path, 0 )) == NULL) {
      printf("Unable to open DATA MASTER path.");
      exit(1);
      }
   printf("Path %02X opened", nd->NCB_NUM);
   printf("Routing info: %c%c.%d.%d.%d.%d.%d",
      nd->NCB_CALLNAME[0],
      nd->NCB_CALLNAME[1],
      nd->NCB_CALLNAME[2],
      nd->NCB_CALLNAME[3],
```

```
    nd->NCB_CALLNAME[4],
    nd->NCB_CALLNAME[5],
    nd->NCB_CALLNAME[6]);
  completed = 0;          /* global variable, do while zero */
  while( !completed ) {
              if( kbhit() )
      completed = 1;
    /*
     * Note: filling in mbuffer[0] with the slave address
     * is actually unecessary, since it was specified above
     * in the ncb_open. It is done here, and the buffer
     * pointer adjusted by 1 in ncb_send, to provide ease-of-use
     * and compatibility with existing modbus applications.
    */
    mbuffer[0] = 0x5;   /*slave address*/
    mbuffer[1] = 0x3;   /*command*/
    mbuffer[2] = 0x0;   /*offset high*/
    mbuffer[3] = 0x0;   /*offset low*/
    mbuffer[4] = 0x0;   /*reg count high*/
    mbuffer[5] = 125;   /* reg count low*/
    if (ncb_send(nd, 6, mbuffer, 10) != 0) {
                /*send the command*/
      printf("Send error: %d.", nd->NCB_RETCODE);
      ret_val = ncb_close(nd);    /*close the path*/
      exit(1);
      }
    if (ncb_receive_wait(nd, mbuffer, 10) != 0)
                /*try to receive*/
      printf("Receive error: %d.", nd->NCB_RETCODE);
    else
                        dump( 125, (unsigned int *) &mbuffer[ 3 ]);
  }
  ncb_close(nd);      /*close the path*/
  exit(0);          /* good exit */
  return( 0 );
}
/*
dump
Dump the contents of the buffer for the length specified.
*/
void dump(qty, buff)
int qty;                                    /* qty of bytes to dump */
unsigned int *buff;       /* buffer with the data */
```

```
        {                                                    /* dump_lines */
        int i = 0, j;
  unsigned int out_value;
        printf("");
        do      {
                printf("");
                for(j = 0;  j < 8 && i < qty;  j++,  i++) {
      out_value = ((*buff << 8) & 0xff00) | ((*buff >> 8) & 0xff);
      printf("%04X ", out_value);
      buff++;
    }
  } while (i < qty);
        printf("");
        }                                                    /* dump_lines */
/*
control_c
This routine replaces the control-C handler supplied by DOS. When you
type a control_C, the program will vector to this routine, which will
set the completed flag to non-zero. In this test program, this will cause
the infinite loop within main to terminate.
*/
void control_c()
{
  signal( SIGINT, SIG_IGN );        /* disable control-c */
  completed = 1;                    /* will cause main loop to complete */
  signal( SIGINT, control_c );      /* reset control-c handler */
}
```

32 Bit Version of TEST4.C

```
/*name test4.c*/
/* Copyright (C) Schneider Automation, Inc. 1989,  All Rights Reserved. */
/*
This test program uses the SA85.SYS device driver to read 125 holding
registers from the MODBUS slave which is found at the given node number.
In order to use this program, enter the following command:
  A>TEST4 12
where "12" is the slave node we want to read from.
*/
/*
include
*/
#define STRICT
#include <windows.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <signal.h>
#include "netbios.h"
#include "netlib.h"
/*
prototypes
*/
int main(int argc, char *argv[] );
void dump(int qty, PUSHORT buff);
void control_c(int sig);
/*
global variables
*/
char mbuffer[256];
char path[ 80 ];
int completed;
/*
main()
*/
int main( argc, argv )
int argc;
char *argv[];
{
   NCB *nd;
   int ret_val;
   int i;
   printf( "MODBUS Command/Response test. Strike any key to abort." );
   if( argc < 2 ) {
      printf( "Usage is: A>TEST4 <slave node>");
      exit( 1 );
   }
   ret_val = sscanf( argv[ 1 ], "%d", &i );
   if( ret_val != 1 || i < 1 || i > 64 ) {
      printf( "Node number must be in the range 1 to 64. " );
      exit( 1 );
   }
   if( signal( SIGINT, control_c ) == SIG_ERR ) {
      printf( "Unable to redirect Control-C." );
      exit( 1 );
   }
```

```
sprintf( path, "DM.%d.0.0.0.0", i );
if ((nd = ncb_open( path, 0 )) == NULL) {
   printf("Unable to open DATA MASTER path.");
   exit(1);
   }
printf("Path %02X opened", nd->NCB_NUM);
printf("Routing info: %c%c.%d.%d.%d.%d.%d",
   nd->NCB_CALLNAME[0],
   nd->NCB_CALLNAME[1],
   nd->NCB_CALLNAME[2],
   nd->NCB_CALLNAME[3],
   nd->NCB_CALLNAME[4],
   nd->NCB_CALLNAME[5],
   nd->NCB_CALLNAME[6]);
completed = 0;          /* global variable, do while zero */
while( !completed ) {
            if( kbhit() )
      completed = 1;
   /*
    * Note: filling in mbuffer[0] with the slave address
    * is actually unecessary, since it was specified above
    * in the ncb_open. It is done here, and the buffer
    * pointer adjusted by 1 in ncb_send, to provide ease-of-use
    * and compatibility with existing modbus applications.
   */
   mbuffer[0] = 0x5;   /*slave address*/
   mbuffer[1] = 0x3;   /*command*/
   mbuffer[2] = 0x0;   /*offset high*/
   mbuffer[3] = 0x0;   /*offset low*/
   mbuffer[4] = 0x0;   /*reg count high*/
   mbuffer[5] = 125;   /* reg count low*/
   if (ncb_send(nd, 6, mbuffer, 10) != 0) {
            /*send the command*/
      printf("Send error: %d.", nd->NCB_RETCODE);
      ret_val = ncb_close(nd);    /*close the path*/
      exit(1);
      }
   if (ncb_receive_wait(nd, mbuffer, 10) != 0)
            /*try to receive*/
      printf("Receive error: %d.", nd->NCB_RETCODE);
   else
                     dump( 125, (PUSHORT) &mbuffer[ 3 ]);
}
```

```
   ncb_close(nd);      /*close the path*/
   exit(0);            /* good exit */
   return( 0 );
}
/*
dump
Dump the contents of the buffer for the length specified.
*/
void dump(qty, buff)
int qty;                                    /* qty of bytes to dump */
PUSHORT buff;               /* buffer with the data */
            {                                                       /* dump_lines */
            int i = 0, j;
   unsigned int out_value;
            printf("");
            do            {
                        printf("");
                        for(j = 0;  j < 8 && i < qty;  j++,  i++) {
         out_value = ((*buff << 8) & 0xff00) | ((*buff >> 8) & 0xff);
         printf("%04X ", out_value);
         buff++;
      }
   } while (i < qty);
            printf("");
            }                                                       /* dump_lines */
/*
control_c
This routine replaces the control-C handler supplied by DOS. When you
type a control_C, the program will vector to this routine, which will
set the completed flag to non-zero. In this test program, this will cause
the infinite loop within main to terminate.
*/
void control_c(int sig)
{
   signal( SIGINT, SIG_IGN );          /* disable control-c */
   completed = 1;                      /* will cause main loop to complete */
   signal( SIGINT, control_c );        /* reset control-c handler */
}
```

A.6.3 Makefile

The following makefile is used to build all of the demo programs. It has been tested with the 32-bit version of Microsoft Visual C++.

```
#
#     +============================================================+
#     |      Copyright (C) 1995, CYBERLOGIC Technologies, Inc.     |
#     +============================================================+
#
#
# Nmake macros for building Windows 32-Bit apps
#
# To run production release enter:
#   >nmake nodebug=1
#
# To remove all but the final target files enter:
#   >nmake clean
#
# To remove all target files enter:
#   >nmake cleanall
#
TARGETOS=WIN95
!include <ntwin32.mak>
!IFDEF NODEBUG
!    IF $(NODEBUG) == 1
BLDENV = free
!    ELSE
BLDENV = checked
!    ENDIF
!ELSE
BLDENV = checked
!ENDIF
!IF "$(CYTARGETPATH)" == ""
BLDTARGET = .
!ELSE
BLDTARGET = $(CYTARGETPATH)(BLDENV)
!ENDIF
BLDOBJ = obj
!IF "$(CYLIBPATH)" == ""
BLDLIB = .
!ELSE
BLDLIB = $(CYLIBPATH)(BLDENV)
```

```
!ENDIF
!IF "$(CYINCPATH)" == ""
BLDINC = .
!ELSE
BLDINC = $(CYINCPATH)
!ENDIF
all:    $(BLDTARGET)est4.exe
        $(BLDTARGET)est4b.exe
        $(BLDTARGET)est4c.exe
        $(BLDTARGET)estslav.exe
        $(BLDTARGET)lobtest.exe
        $(BLDTARGET)eadnode.exe
$(BLDOBJ)est4.obj: test4.c $(BLDINC)etlib.h $(BLDINC)etbios.h
   $(cc) $(cdebug) $(cflags) $(cvarsdll) -I$(BLDINC)
      /Fo$(BLDOBJ)est4.obj test4.c
$(BLDOBJ)est4b.obj: test4b.c $(BLDINC)etlib.h $(BLDINC)etbios.h
   $(cc) $(cdebug) $(cflags) $(cvarsdll) -I$(BLDINC)
      /Fo$(BLDOBJ)est4b.obj test4b.c
$(BLDOBJ)est4c.obj: test4c.c $(BLDINC)etlib.h $(BLDINC)etbios.h
   $(cc) $(cdebug) $(cflags) $(cvarsdll) -I$(BLDINC)
      /Fo$(BLDOBJ)est4c.obj test4c.c
$(BLDOBJ)estslav.obj: testslav.c $(BLDINC)etlib.h $(BLDINC)etbios.h
   $(cc) $(cdebug) $(cflags) $(cvarsdll) -I$(BLDINC)
      /Fo$(BLDOBJ)estslav.obj testslav.c
$(BLDOBJ)lobtest.obj: globtest.c $(BLDINC)etlib.h $(BLDINC)etbios.h
   $(cc) $(cdebug) $(cflags) $(cvarsdll) -I$(BLDINC)
      /Fo$(BLDOBJ)lobtest.obj globtest.c
$(BLDOBJ)eadnode.obj: readnode.c $(BLDINC)etlib.h $(BLDINC)etbios.h
   $(cc) $(cdebug) $(cflags) $(cvarsdll) -I$(BLDINC)
      /Fo$(BLDOBJ)eadnode.obj readnode.c
$(BLDTARGET)est4.exe: $(BLDOBJ)est4.obj $(BLDLIB)etlib.lib
   $(link) $(linkdebug) $(conflags) -out:$(BLDTARGET)est4.exe
      $(BLDOBJ)est4.obj $(BLDLIB)etlib.lib $(guilibsdll)
$(BLDTARGET)est4b.exe: $(BLDOBJ)est4b.obj $(BLDLIB)etlib.lib
   $(link) $(linkdebug) $(conflags) -out:$(BLDTARGET)est4b.exe
      $(BLDOBJ)est4b.obj $(BLDLIB)etlib.lib $(guilibsdll)
$(BLDTARGET)est4c.exe: $(BLDOBJ)est4c.obj $(BLDLIB)etlib.lib
   $(link) $(linkdebug) $(conflags) -out:$(BLDTARGET)est4c.exe
      $(BLDOBJ)est4c.obj $(BLDLIB)etlib.lib $(guilibsdll)
$(BLDTARGET)estslav.exe: $(BLDOBJ)estslav.obj $(BLDLIB)etlib.lib
   $(link) $(linkdebug) $(conflags) -out:$(BLDTARGET)estslav.exe
      $(BLDOBJ)estslav.obj $(BLDLIB)etlib.lib $(guilibsdll)
$(BLDTARGET)lobtest.exe: $(BLDOBJ)lobtest.obj $(BLDLIB)etlib.lib
```

```
    $(link) $(linkdebug) $(conflags) -out:$(BLDTARGET)lobtest.exe
        $(BLDOBJ)lobtest.obj $(BLDLIB)etlib.lib $(guilibsdll)
$(BLDTARGET)eadnode.exe: $(BLDOBJ)eadnode.obj $(BLDLIB)etlib.lib
    $(link) $(linkdebug) $(conflags) -out:$(BLDTARGET)eadnode.exe
        $(BLDOBJ)eadnode.obj $(BLDLIB)etlib.lib $(guilibsdll)
# Clean all
cleanall: clean
    -del $(BLDTARGET)est4.exe
    -del $(BLDTARGET)est4b.exe
    -del $(BLDTARGET)est4c.exe
    -del $(BLDTARGET)estslav.exe
    -del $(BLDTARGET)lobtest.exe
    -del $(BLDTARGET)eadnode.exe
# Clean all but .DLL, .LIB and .EXE
clean:
    -del $(BLDOBJ).obj
    -del $(BLDTARGET)est4.map
    -del $(BLDTARGET)est4.pdb
    -del $(BLDTARGET)est4b.map
    -del $(BLDTARGET)est4b.pdb
    -del $(BLDTARGET)est4c.map
    -del $(BLDTARGET)est4c.pdb
    -del $(BLDTARGET)estslav.map
    -del $(BLDTARGET)estslav.pdb
    -del $(BLDTARGET)lobtest.map
    -del $(BLDTARGET)lobtest.pdb
    -del $(BLDTARGET)eadnode.map
    -del $(BLDTARGET)eadnode.pdb
```

Modicon, Square D and Telemecanique are PLC brand names from Schneider. These products are sold in the US by Square D; in Canada, Latin America, Europe, Africa, Asia/Pacific and Middle East by Schneider; in Germany by AEG Schneider Automation; in China and Persian Gulf by Schneider Automation; in South Africa by ASA Systems Automation; in Austria by Online.

Schneider Automation, Inc.
One High Street
North Andover, MA 01845
Tel: (1) 508-794-0800
Fax: (1) 508-975-9400

Schneider Automation GmbH
Steinheimer Strasse 117
D-63500 Seligenstadt
Tel: (49) 6182 81-2584
Fax: (49) 6182 81-2860

Schneider Automation S.A.
245, Route des Lucioles-BP147
F-06903 Sophia-Antipolis Cedex
Tel: (33) 92 96 20 00
Fax: (33) 93 65 37 15

890 USE 110 00

0398