



BLACK BOX[®]
NETWORK SERVICES

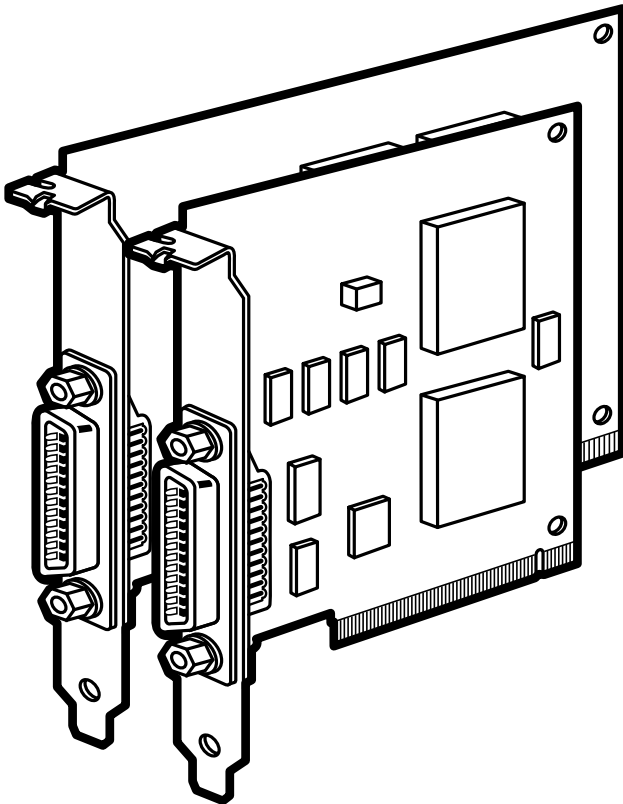


DECEMBER 1999

IC098C

IC099C

Personal 488 PCI Card Personal 488 ISA Card



**CUSTOMER
SUPPORT
INFORMATION**

Order **toll-free** in the U.S. 24 hours, 7 A.M. Monday to midnight Friday: **877-877-BBOX**
FREE technical support, 24 hours a day, 7 days a week: Call **724-746-5500** or fax **724-746-0746**
Mail order: **Black Box Corporation**, 1000 Park Drive, Lawrence, PA 15055-1018
Web site: www.blackbox.com • E-mail: info@blackbox.com

**FEDERAL COMMUNICATIONS COMMISSION
AND
CANADIAN DEPARTMENT OF COMMUNICATIONS
RADIO FREQUENCY INTERFERENCE STATEMENTS**

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio communication. It has been tested and found to comply with the limits for a Class A computing device in accordance with the specifications in Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when the equipment is operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user at his own expense will be required to take whatever measures may be necessary to correct the interference.

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This digital apparatus does not exceed the Class A limits for radio noise emission from digital apparatus set out in the Radio Interference Regulation of Industry Canada.

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de la classe A prescrites dans le Règlement sur le brouillage radioélectrique publié par Industrie Canada.



This equipment complies with the safety and emissions standards of the European Community.

TRADEMARKS

The trademarks mentioned in this manual are the sole property of their owners.

CONTENTS

1. Specifications	6
2. Introduction	7
2.1 Description	7
2.2 About this Manual	7
2.3 Hardware Connection	8
2.4 Software	9
3. Personal 488 PCI Card	10
3.1 What the Package Includes	10
3.2 Installing the New Hardware	11
4. Personal 488 ISA Card	13
4.1 What the Package Includes	13
4.2 Configuring the New Hardware	13
4.3 Installing the New Hardware	19
5. Installing the Hardware Drivers and Configuring the Software	21
5.1 Windows 95/98 Users Only	21
5.1.1 Plug-and-Play Devices	21
5.1.2 “Legacy” Devices	21
5.1.3 Driver Installation/Removal for IEEE Controllers	28
5.2 Windows NT Users Only	32
5.2.1 Plug-and-Play and “Legacy” Devices	32
5.2.2 Windows NT Service Packet 3 (SP3) Driver Installation/Removal for IEEE Controllers	34
6. API Command Reference	38
6.1 Introduction	38
6.2 Abort	38
6.3 Arm	39
6.4 AutoRemote	41
6.5 Buffered	42
6.6 BusAddress	43
6.7 CheckListener	44
6.8 Clear	45
6.9 ClearList	46
6.10 Close	47
6.11 ControlLine	48
6.12 DigArm	49
6.13 DigArmSetup	50

6.14 DigRead	51
6.15 DigSetup	52
6.16 DigWrite	53
6.17 Disarm	54
6.18 EnterX	55
6.19 Error	59
6.20 FindListener	60
6.21 Finish	61
6.22 GetError	62
6.23 GetErrorList	63
6.24 Hello	64
6.25 KeepDevice	65
6.26 Listen	66
6.27 Local	66
6.28 LocalList	67
6.29 Lol	68
6.30 MakeDevice	69
6.31 MakeNewDevice	70
6.32 MyListenAddr	71
6.33 MyTalkAddr	72
6.34 OnDigEvent	73
6.35 OnDigEventVDM	74
6.36 OnEvent	75
6.37 OnEventVDM	77
6.38 OpenName	79
6.39 OutputX	80
6.40 PassControl	84
6.41 PPoll	85
6.42 PPollConfig	86
6.43 PPollDisable	87
6.44 PPollDisableList	88
6.45 PPollUnconfig	88
6.46 Remote	89
6.47 RemoteList	90
6.48 RemoveDevice	90
6.49 Request	91
6.50 Reset	92
6.51 Resume	93
6.52 SendCmd	94
6.53 SendData	95
6.54 SendEoi	96
6.55 SPoll	96

6.56 SPollList	97
6.57 Status	99
6.58 Stop.	102
6.59 Talk.	102
6.60 Term.	103
6.61 TermQuery	104
6.62 TimeOut.	105
6.63 TimeOutQuery	106
6.64 Trigger	106
6.65 TriggerList	106
6.66 Unlisten	108
6.67 UnTalk	108
6.68 Wait.	109
7. Troubleshooting	110
7.1 Radio Interference Problems	110
7.2 IEEE 488 Bus Errors	110
7.3 Hardware-Software Conflicts	111
7.4 Checking Hardware and Software Settings	111
Appendix	113
A.1 IEEE 488 Bus and Serial Bus	113
A.2 IEEE 488 Bus Commands.	114
A.3 ASCII Codes	115
A.3.1 ASCII Code Summary	115
A.3.2 ASCII Code Details	117
Abbreviations	122
Index	127

1. Specifications

Maximum Data-Transfer Rate—1 Mbps

Connectors—Both cards: (1) standard IEEE-488 with metric studs;
IC098C: (1) DB9 female; IC099C: (1) IEEE-488

Indicators—None

Operating Temperature—32 to 158°F (0 to 70°C)

Relative Humidity—0 to 95%

Power—From the PC

Size—IC098C: PCI-slot card; IC099C: Half-slot card

Weight—Less than 1 lb. (0.5 kg)

2. Introduction

2.1 Description

The Personal 488 Cards are IEEE 488.2 compatible and are supported by 32-bit Driver488 software for Windows 95 or 98, and for Windows NT, named Driver488/W95 and Driver488/WNT respectively. This manual describes two versions of the Personal 488 Cards.

- **Personal488 PCI Card (IC098C)**—The IC098C interface board features plug-and-play and 32-bit PCI local bus compatibility. Provides 1-Mbps data-transfer rate. Offers full IEEE 488.2 support. Supported by Windows 95 or 98 and Windows NT drivers. Provides eight channels of general-purpose digital I/O.
- **Personal 488 ISA Card (IC099C)**—The IC099C interface board features 16-bit ISA-bus compatibility. Provides 1-Mbps data-transfer rate. Offers full IEEE 488.2 support. Supported by Windows 95 or 98 and Windows NT drivers. Provides eleven interrupt lines and seven DMA channels. CE compliant.

2.2 About this Manual

This manual is divided into the following sections:

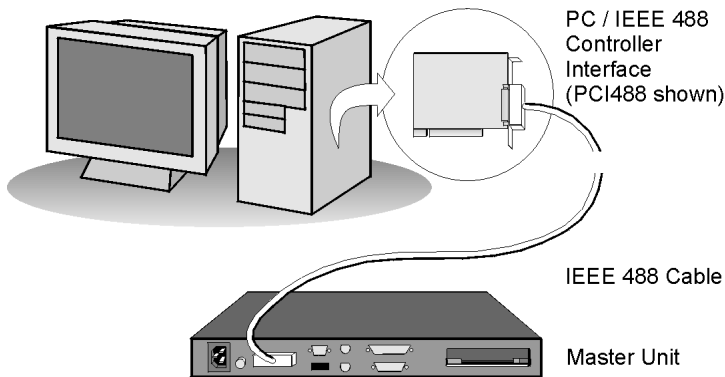
- **Chapter 1, Specifications**, lists the technical specifications for the Cards.
- **Chapter 2, Introduction (this chapter)**, gives a general description of both the interface hardware and the driver software associated with each of the Personal 488 Cards.
- **Chapter 3, Personal 488 PCI Card**, explains how to install and configure the Personal 488 PCI Card.
- **Chapter 4, Personal 488 ISA Card**, explains how to install and configure the Personal 488 ISA Card.
- **Chapter 5, Driver 488/W95 and Driver 488/WNT**, describes in more detail the Windows driver software that comes with each of the Personal 488 Cards, and includes instructions for configuring this software.
- **Chapter 6, API Command Reference**, provides descriptions for the entire API command set, covering both versions of Driver 488—Driver488/W95 and Driver488/WNT—and both Cards. The description format of the individual

API commands includes the command syntax, returned response, operating mode, bus states, and an example program excerpt.

- **Chapter 7, Troubleshooting**, provides a reference for possible solutions to technical problems. Before calling for technical support, refer to this chapter.
- The **Appendix** provides background information about the IEEE 488 bus, the serial bus, and ASCII controls.
- The **Index** provides a comprehensive alphabetical listing of the main terms and topics in this manual. Also, the Abbreviations on the last pages of this manual provides an overall list of abbreviations, including acronyms and ASCII control codes, as an additional reference for this manual and other related literature.

2.3 Hardware Connection

The Personal 488 controller interface must be properly connected to a data-acquisition device. The following diagram depicts an IEEE 488 connection from a Personal 488 controller interface board to a data-acquisition master unit.



*IEEE 488 Connection
(PC to Master Unit)*

Figure 2-1. Typical Hardware Connection.

2.4 Software

Driver488/W95 & Driver488/WNT

This driver software integrates IEEE 488.2 control into Windows 95 or 98 and Windows NT applications. It supports both IEEE 488 Cards, and provides true multi-tasking device locking. Plus, it's specifically designed for the 32-bit Windows environment, and includes interactive control.

NOTE

For proper operation of the cards using the Windows NT operating system, load the software BEFORE configuring and installing the hardware. See Section 5.2.

NOTE

Throughout this manual and in the software screens, the ISA card is sometimes called the Personal488/AT and the PCI card is sometimes called the Personal488/PCI.

3. Personal 488 PCI Card

This chapter describes the PCI version. If you have the ISA version, skip this chapter and go to **Chapter 4**.

3.1 What the Package Includes

The Personal488 PCI Card, including the IEEE 488 interface board and the Driver488 software, is carefully inspected, both mechanically and electrically, before shipment. After unpacking the product, carefully check for any obvious signs of physical damage that may have occurred during shipment. If you suspect damage, call Black Box immediately at 724-746-5500. Retain all shipping materials in case you need to ship the unit back to Black Box.

The Personal488 PCI Card package includes:

- Personal 488 PCI Card
- (1) CD-ROM
- (1) Ribbon cable
- (1) Faceplate
- This user manual

NOTE

For proper operation of the cards using the Windows NT operating system, load the software BEFORE configuring and installing the hardware. See Section 5.2.

Controller Interface

The Personal 488 PCI interface board is easy to install if you're using Windows 95 or 98—just plug-and-play. You don't have to physically configure the hardware. Instead, after you install your board as described in the following text, the board is configured automatically.

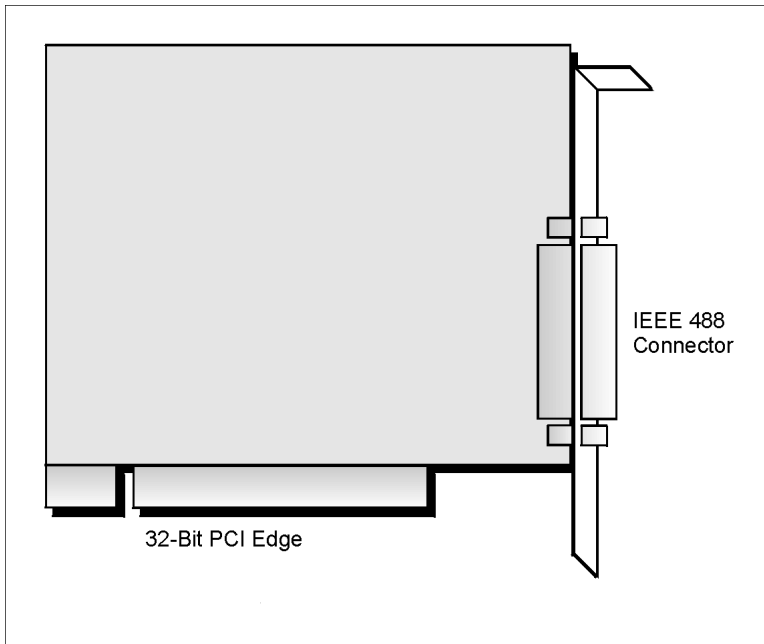


Figure 3-1. Personal 488 PCI Card.

3.2 Installing the New Hardware

Typical IEEE 488 interface boards are installed into expansion slots inside the PC's system unit. PCs have the following types of expansion slots:

- ISA expansion slots. ISA slots can either be an 8-bit slot with one card-edge receptacle (PC-bus compatible), or a 16-bit slot with two card-edge receptacles (AT-bus compatible). Eight-bit ISA boards may be used in either the 8-bit or 16-bit ISA slot, while 16-bit ISA boards may only be used in the 16-bit ISA slot.
- PCI expansion slots. PCI slots are 32-bit slots, used only by PCI boards.

For technical assistance, see **Chapter 7, Troubleshooting**, in this manual, or the troubleshooting section in your PC's manual. If you are still not sure of the problem, call Black Box Technical Support at 724-746-5500.

Installing the Personal 488 PCI Card in a PCI Slot

You'll find general instructions for installing the board here since the design of computer cases varies. Refer to your PC's reference manual if you need to.

1. Turn OFF the power to your computer and any other connected peripheral devices.
- Touch a large grounded metal surface to discharge any static-electricity buildup in your body.
- Avoid any contact with internal parts. Handle cards only by their edges.
- Disconnect the AC power before removing the cover.
2. Unplug all power cords and cables that may interfere from the back of the computer.
3. Remove your computer's cover by removing its mounting screws with a screwdriver. Slide the cover OFF. If necessary, refer to your PC's manual.
4. Your IEEE 488 controller interface must be installed in a 32-bit PCI-bus expansion slot. Select an available PCI expansion slot and remove its slot cover by unscrewing the holding screw and sliding it out. Save this screw for securing the interface after it is installed.
5. To install the IEEE 488 controller interface, carefully align the card-edge connector with the PCI slot on the motherboard, fitting the IEEE 488 port through the rear-panel opening. Push the board down firmly, but gently, until it is well seated.
6. Replace the slot-cover holding screw to secure the board in place.
7. Replace the computer's cover and screws. Then reconnect all power cords and cables to the back of the computer. If available, connect your external data-acquisition instrument to the IEEE 488 port connector on the interface.
8. Turn on your PC. At this point, the hardware installation is complete.

4. Personal 488 ISA Card

This chapter describes the ISA version. If you have the PCI version, read **Chapter 3**, then skip this chapter and go on to **Chapter 5**.

4.1 What the Package Includes

The Personal 488 ISA Card, including the IEEE 488 interface board and the Driver488 software, is carefully inspected, both mechanically and electrically, before shipment.

The package includes:

- Personal 488 ISA Card
- (1) CD-ROM
- This user manual

After unpacking all the items carefully, check for any obvious signs of physical damage that may have occurred during shipment. If anything is missing or damaged, call Black Box immediately at 724-746-5500. Retain all shipping materials in case you need to ship the unit back to Black Box.

4.2 Configuring the New Hardware

Figure 4-1 shows the board layout and default DIP-switch settings for the Personal 488 ISA Card.

NOTE

For proper operation of the cards using the Windows NT operating system, load the software BEFORE configuring and installing the hardware. See Section 5.2.

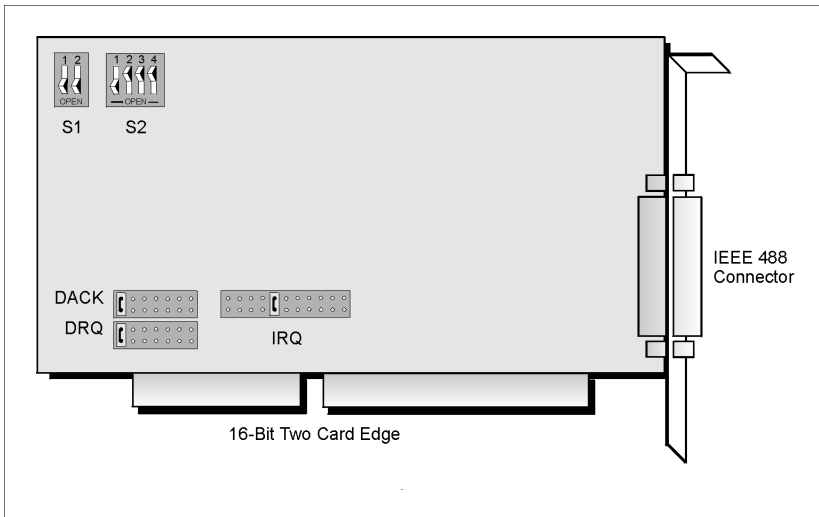


Figure 4-1. Board Layout.

The following text will guide you through the setup of your IEEE 488 controller interface. It includes instructions on how to verify the resource settings of ports in your system, and how to properly configure the switches and jumpers on your interface board.

To avoid a configuration conflict, you must first verify which I/O addresses, IRQs, and DMAs are being used by existing ports in your system, before you configure and install the Personal 488 ISA Card.

Step 1: Verifying/Recording the Current System Settings

The Windows Control Panel enables you to easily determine and configure the I/O addresses, IRQ setting, and DMA settings in your system for proper operation. Perform the following steps to verify your system settings.

1. Open the Control Panel window from the Start > Settings menu, click on the System icon, and select the Device Manager tab. Under the line “Ports (COM & LPT),” look for a list of used ports. For each port, highlight the port and click on the Properties button.
2. Properties already being used in the system are displayed under the Resources tab. Values NOT listed are available.

- For each listed port, record which Input/Output (I/O) address, if any, is being used.
 - For each listed port, record which Interrupt Request (IRQ) value, if any, is being used.
 - For each listed port, record which Direct Memory Access (DMA) value, if any, is being used.
3. Exit Windows and turn the system OFF.

The I/O base address, IRQ, and DMA settings are switch/jumper selectable via the following locations on the Personal 488 ISA Card: One 2-microswitch DIP switch labeled SW1, one 4-microswitch DIP switch labeled SW2, two 14-pin headers labeled DACK and DRQ, and one 22-pin header labeled IRQ. The DIP-switch settings and the way you arrange the jumpers on the headers set the hardware configuration.

For the next steps, make sure that the I/O address, IRQ, and DMA set on the interface board are different from any existing ports in your system. A conflict results when two I/O addresses, IRQs, or DMAs are the same. (As the exception, additional Personal 488 ISA Cards may share the same IRQ and DMA values.)

Step 2: Configuring the Interface I/O Base Address

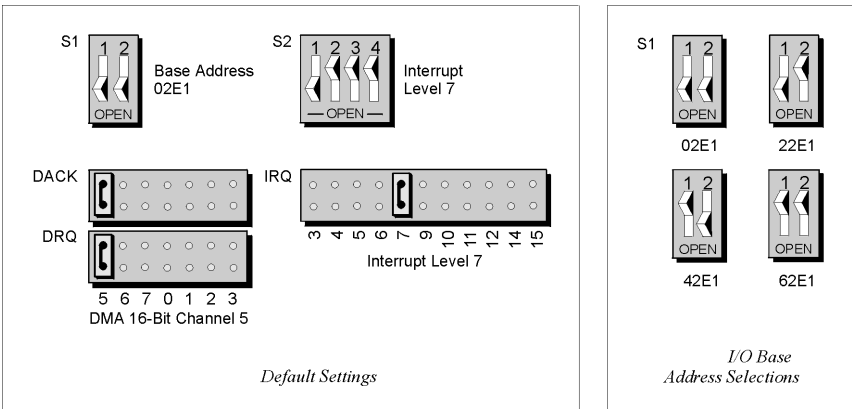


Figure 4-2. Personal 488 ISA Card I/O Base Address Selections.

1. The factory default I/O base address is 02E1. If this creates a conflict, reset SW1 according to the figure and following table. The register addresses will be automatically relocated at fixed offsets from the base address.
2. If you change the default, record the new Input/Output (I/O) address being used.

Table 4-1. Selected I/O Base Address.

Selected I/O Base Address				Register	
02E1	22E1	42E1	62E1	Read Register	Write Register
Automatic Offset Addresses					
02E1	22E1	42E1	62E1	Data In	Data Out
06E1	26E1	46E1	62E1	Interrupt Status 1	Interrupt Mask 1
0AE1	2AE1	4AE1	6AE1	Interrupt Status 2	Interrupt Mask 2
0EE1	2EE1	4EE1	6EE1	Serial Poll Status	Serial Poll Mode
12E1	32E1	52E1	72E1	Address Status	Address Mode
16E1	36E1	56E1	76E1	CMD Pass Through	Auxiliary Mode
1AE1	3AE1	5AE1	7AE1	Address 0	Address 0/1
1EE1	3EE1	5EE1	7EE1	Address 1	End of String

The I/O base address sets the addresses used by the computer to communicate with the Personal 488 ISA Card hardware on the board. The address is normally specified in hexadecimal and can be 02E1, 22E1, 42E1, or 62E1. The registers of the IOT7210 IEEE 488 controller chip and other auxiliary registers are then located at fixed offsets from the base address.

Most versions of Driver488 are capable of managing as many as four Personal 488 ISA Cards. To do so, you must arrange the interface configurations to avoid conflict among themselves. No two Cards may have the same I/O address, but they may, and usually should, have the same DMA channel and interrupt level.

Step 3: Configuring the Personal 488 ISA Card Interface Interrupt (IRQ)

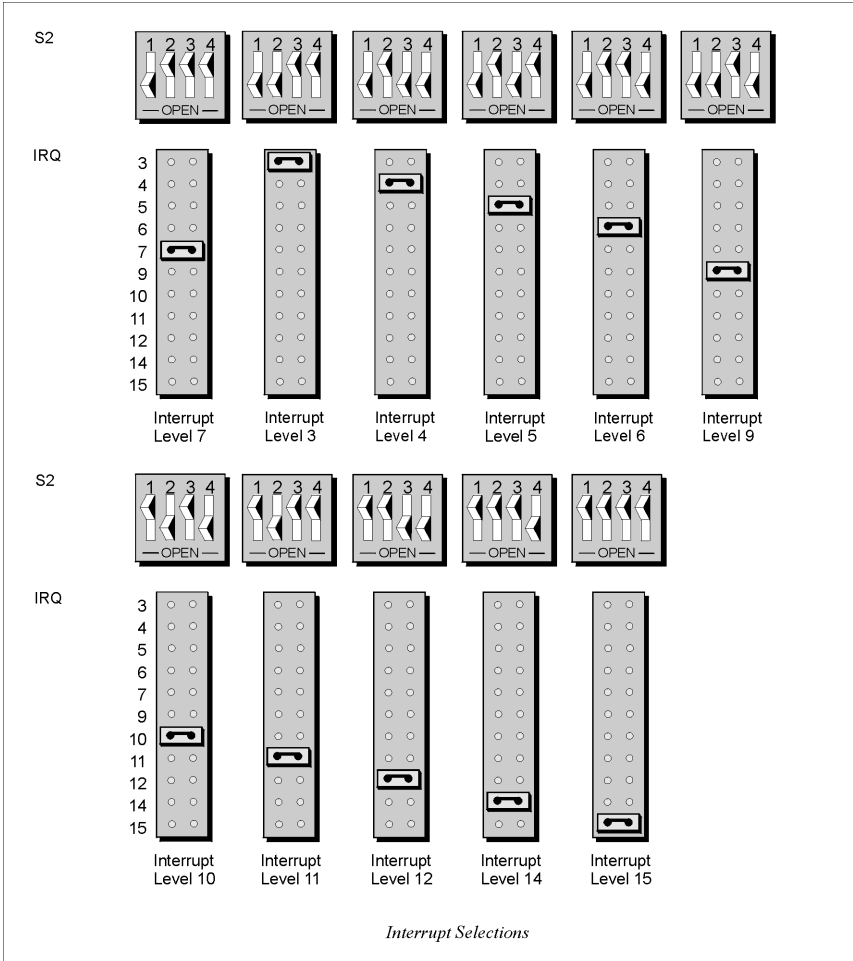


Figure 4-3. Configuring the Personal 488 ISA Card Interface Interrupt (IRQ).

1. The factory-default Interrupt (IRQ) is 7. If this creates a conflict, reset SW2 and jumper IRQ according to **Figure 4-3**. The switch and jumper settings must both indicate the same interrupt level to operate correctly with interrupts.

2. If you change the default, record the new Interrupt (IRQ) you're using.

You can set the Personal 488 ISA Card to interrupt the PC when certain hardware conditions occur. You can set the main board interrupt to IRQ level 3 through 7, 9 through 12, 14, or 15. Interrupts 10 through 15 are only available in a 16-bit slot on an AT-class machine. Interrupt 9 becomes synonymous with Interrupt 2 when used in a PC/XT bus.

Several Personal 488 ISA Cards may share the selected interrupt in the same AT chassis. The Card adheres to the AT-style interrupt-sharing conventions. When the Card requires service, the IRQ jumper determines which PC interrupt level is triggered. When an interrupt occurs, the interrupting device must be reset by writing to an I/O address that is different for each interrupt level. The switch settings may determine the I/O address to which the Card's interrupt-rearm circuitry responds.

Step 4: Configuring the Personal 488 ISA Card DMA Channel

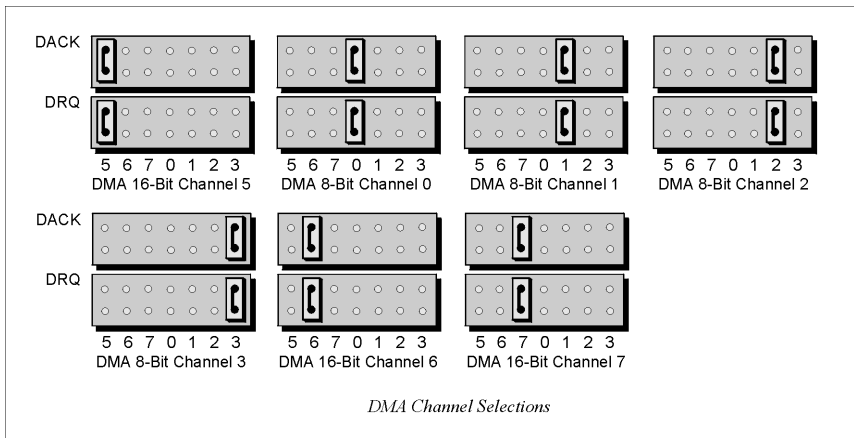


Figure 4-4. Personal 488 ISA Card DMA Channel Selections.

1. The factory-default DMA channel is 5. If this creates a conflict, reset jumpers DACK and DRQ according to **Figure 4-4**.
2. If you change the default, record the new DMA channel being used.

Direct Memory Access (DMA) is a high-speed method of transferring data from or to a peripheral, such as a digitizing oscilloscope, to or from the PC's memory.

The AT class machine has seven DMA channels. Channels 0 to 3 (8-bit), 5, 6, and 7 (16-bit) are available only in a 16-bit slot on an AT class machine. Channel 2 is usually used by the floppy-disk controller, and is unavailable. Channel 3 is often used by the hard-disk controller in PCs, XTs, and the PS/2 with the ISA bus, but is usually not used in ATs. Channels 5 to 7 are 16-bit DMA channels and offer the highest throughput (up to 1 Megabyte per second). Channels 0 to 3 are 8-bit DMA channels and although slower, they offer compatibility with existing applications that only used 8-bit DMA channels. Under some rare conditions, high-speed transfers on DMA Channel 1 can demand so much of the available bus bandwidth that a floppy controller cannot access the channel simultaneously.

4.3 Installing the New Hardware

Typical IEEE 488 interface boards are installed into expansion slots inside the PC's system unit. Typical PCs have the following types of expansion slots:

- ISA expansion slots. ISA slots can either be an 8-bit slot with one card-edge receptacle (PC-bus compatible), or a 16-bit slot with two card-edge receptacles (AT-bus compatible). Eight-bit ISA boards may be used in either the 8-bit or 16-bit ISA slot, while 16-bit ISA boards may only be used in the 16-bit ISA slot.
- PCI expansion slots. PCI slots are 32-bit slots, used only by PCI boards.

For technical assistance, see **Chapter 7, Troubleshooting**, in this manual, or the troubleshooting section in your PC's manual. If you are still not sure of the problem, contact the dealer or manufacturer of your interface board or PC.

NOTE

For proper operation of the cards using the Windows NT operating system, load the software BEFORE configuring and installing the hardware. See Section 5.2.

Installing the Personal 488 ISA Card into an ISA Slot

General instructions for installing the board are given since the design of computer cases varies. Refer to your PC's reference manual whenever in doubt.

1. Turn OFF the power to your computer and any other connected peripheral devices. Follow these precautions for static-electricity discharge:
 - Touch a large grounded metal surface to discharge any static-electricity buildup in your body.

- Avoid any contact with internal parts. Handle cards only by their edges.
 - Disconnect the AC power before removing the cover.
2. Unplug all power cords and cables that may interfere from the back of the computer.
 3. Remove your computer's cover by removing its mounting screws with a screwdriver. Slide the cover OFF. If necessary, refer to your PC's manual.
 4. Your Personal 488 ISA Card must be installed in an 8-bit ISA-bus expansion slot. Select an available ISA expansion slot and remove its slot cover by unscrewing the holding screw and sliding it out. Save this screw for securing the Card after it is installed.
 5. To install the Personal 488 ISA Card, carefully align the card-edge connector with the ISA slot on the motherboard, fitting the IEEE 488 port through the rear-panel opening. Push the board down firmly, but gently, until it is well seated.
 6. Replace the cover slot holding screw to secure the Card in place.
 7. Replace the computer's cover and screws. Then reconnect all power cords and cables to the back of the computer. If available, connect your external data-acquisition instrument to the IEEE 488 port connector on the interface.
 8. Turn on your PC.

At this point, the hardware installation is complete.

5. Installing the Hardware Drivers and Configuring the Software

NOTE

Throughout this manual and in the software screens, the ISA card is sometimes called the Personal488/AT and the PCI card is sometimes called the Personal488/PCI.

5.1 Windows 95 or 98 Users Only

NOTE

If you are using Windows NT, go to Section 5.2.

5.1.1 PCI VERSION

If the operating system you are using is Windows 95 or 98, follow these steps to install a Personal 488 PCI Card. If the operating system you are using is Windows NT, go to **Section 5.2**.

1. Remove power from the PC.
2. Physically install the device into a 32-bit expansion slot as described in **Chapter 3** or **4**.
3. Return power to the PC. After the computer powers up and detects the new device, you should see a screen prompt asking for a CD-ROM.
4. Place CD-ROM in the CD-ROM drive.
5. Follow the screen prompts.

This completes the plug-and-play device install procedure for Windows 95 and Windows 98 users.

5.1.2 ISA VERSION

NOTE

If you are using Windows NT, go to Section 5.2.

If you are using Windows 95 or 98, follow these steps to install a Personal 488 ISA Card.

1. Remove power from the PC.

2. Physically configure and install the device as described in **Chapter 3** or **4**.
3. Return power to the PC.
4. Access the **Add New Hardware** dialog box by selecting the Windows **Start** button and navigating as follows:

Start Settings Control Panel Add New Hardware



Figure 5-1. Navigating to Add New Hardware.

NOTE

The following screen images have been taken from Windows 95. The Windows 98 version screen images are similar.

5. The “Add New Hardware Wizard” displays an introductory message and prompts you to click **Next**.

NOTE

After you click **Next**, Windows 98 will automatically search for installed devices.

Add New Hardware Wizard

This wizard will help you quickly install a new piece of hardware.

To begin installing your new hardware, click **Next**.

Add New Hardware Wizard

If your hardware is already installed, you should have Windows detect it.

When Windows detects new hardware, it automatically determines the current settings for the device and installs the correct driver.

Do you want Windows to search for your new hardware?

Yes (Recommended)

→ **N**o ←

< **B**ack

Next >

Cancel

Figure 5-2. Add New Hardware Wizard.

6. Select “No” when asked, “Do you want Windows to search for your new hardware?” Then click **Next**.

Add New Hardware Wizard

Select the type of hardware you want to install.

Hardware types:






-  Mouse
-  Multi-function adapters
-  Network adapters
-  **Other devices**
-  PCMCIA socket

Figure 5-3. Selecting the Type of Hardware to Install.

7. Choose **Other Devices** from the list of hardware types.
8. When the Manufacturer/Models dialog box appears:

Select **Unknown Devices** for Manufacturer.

Select **Unsupported Device** for Models.

Then click the **Have Disk** button.

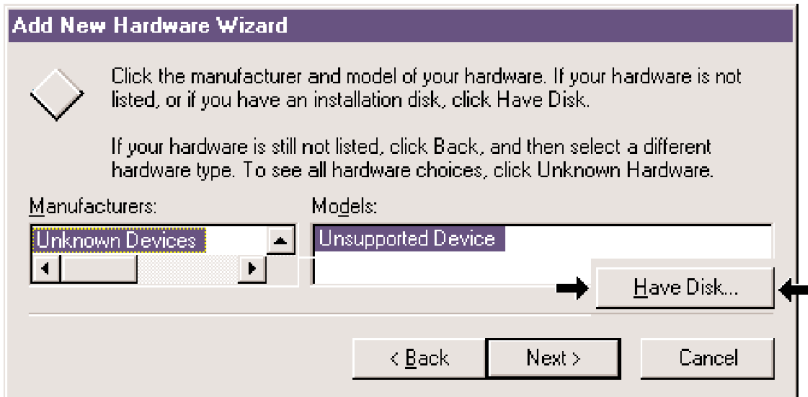


Figure 5-4. Selecting the Manufacturer/Model.

9. Insert the CD-ROM into the appropriate drive, click Browse, and select the CD's root directory. Windows will now look for an "inf" file.

The file "iotech.inf" will now appear in the text field of the Open window. Click **OK**.

10. Windows will now display a list of devices to install. Select the specific Personal488 device to be installed. You should see the device in the Wizard's list of models.

After making the selection, click **Next**.

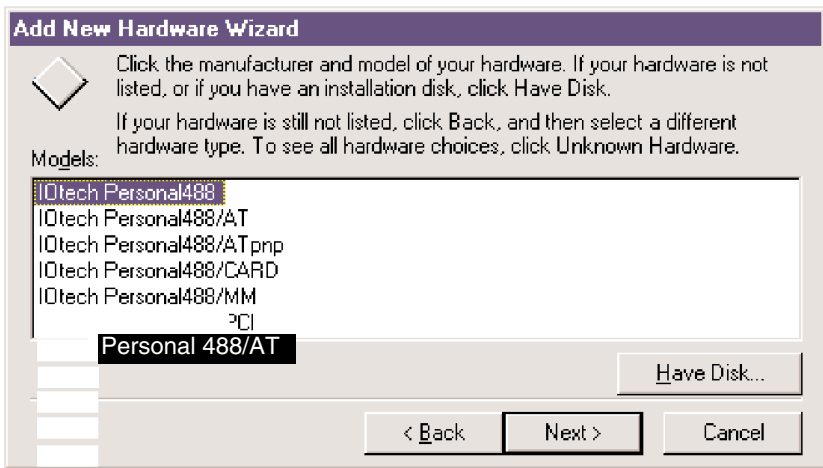


Figure 5-5. Selecting the Device You Wish to Install.

11. Check the default resource settings. If the settings do not match your board configuration (for example, if you changed a default jumper setting) you will need to use the Device Manager to enter the changes.

Regardless of whether or not you need to make settings changes, click **Next**. The disk file for the device is copied into the PC.

Follow the screen prompts. When prompted, restart the computer.

If you do not need to change settings, the installation is complete. If you do need to make setting changes, continue with **Step 12**.



Figure 5-6. Resources Settings screen.

NOTE

Steps 12 through 14 pertain to making settings changes for resource types. Perform these steps only if you need to change your device settings.

12. If you need to make settings changes, first access the Device Manager. Do this by beginning with the Desktop **Start** button.

Start Settings Control Panel System Device Manager

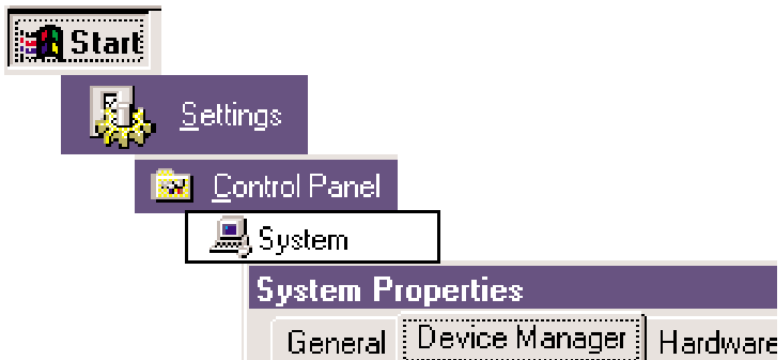


Figure 5-7. Accessing Device Manager.

13. In the Device Manager, select IEEE 488.2 Controllers, then the specific device. The example shows Personal488/AT (the Personal 488 ISA Card)

selected. The Personal 488 PCI card is called Personal 488/PCI.

After selecting the device, click the **Properties** button.

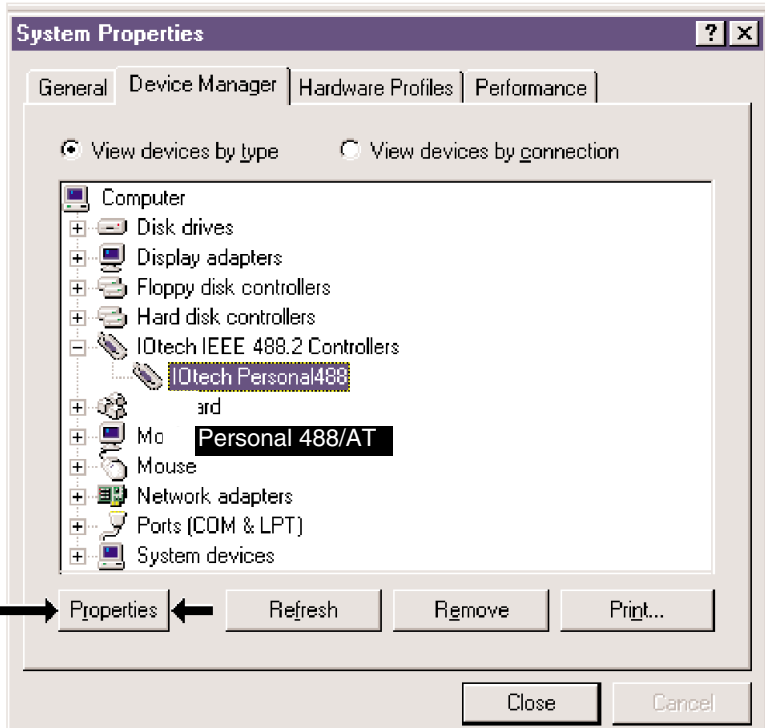


Figure 5-8. Device Manager screen.

When the device properties window opens, select the **Resources** tab. A screen similar to the one shown in **Figure 5-9** should appear.

Highlight the first setting you wish to change.

Click the **Change Setting** button. An edit window appears for the selected setting. Follow the screen prompts.

After you have made all needed changes, click OK.

You will be prompted to restart the computer, to put the new settings into effect.

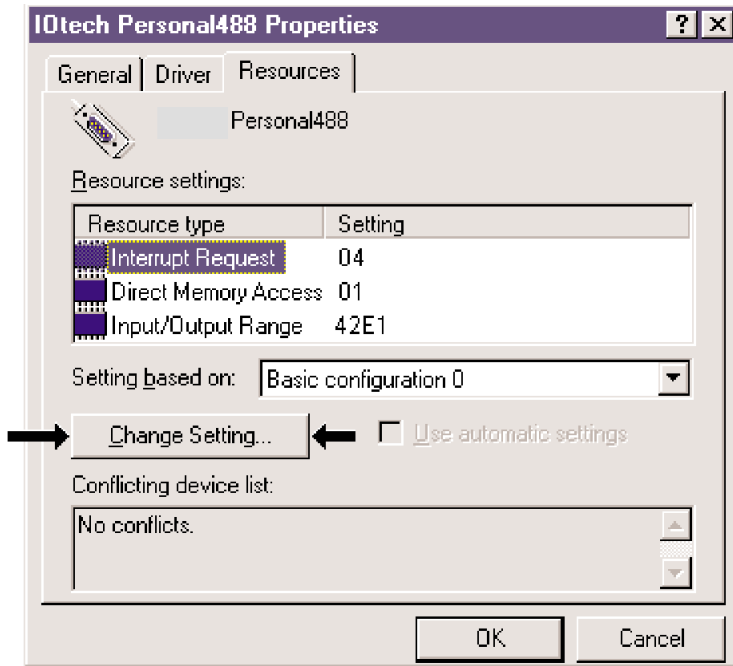


Figure 5-9. Change Setting screen.

This completes the installation procedure for the ISA version in Windows 95 and Windows 98.

5.1.3 WINDOWS 95/98 DRIVER INSTALLATION/REMOVAL FOR IEEE CONTROLLERS

This procedure assumes that you have the most recent revision of the driver/software. You must remove any previous versions of drivers and applicable software.

Removal

Follow the removal steps only if software is already installed or if you made a mistake during installation.

First, uninstall software using Add/Remove Programs in the control panel. The keywords of the program name will be "Personal IEEE." Highlight and remove.

Second, remove the IEEE 488-related .inf file located in "C:\Windows\Inf\Other" or "C:\Windows\Inf" folder.

Next, remove IEEE Controller hardware by entering the System device manager located in the control panel. Highlight and remove the specific controller.

Installation

For all 488 controllers, do *not* install the controller at this time.

1. Go to Control Panel, select Add New Hardware, and click Next. Windows will then ask if you want Windows to find the hardware.
2. Select No. An item window will then be displayed. If there is an IEEE 488.2 controller item in the list, highlight and click next. If there is no IEEE 488 item, highlight “?Other Devices” and click Next.
3. Click Have Disk. Windows will default to the A drive.
4. Browse to the CD’s root directory. This is where the iotech.inf file is located. Make sure this file is highlighted and click OK, then OK again.
5. Select the relevant controller and click Next. (For the ISA version, note the hardware settings recommended by Windows.)
6. Click Next, then Finish.
7. Select Add/Remove Programs, then Install.
8. Browse to the floppy drive and select the appropriate controller folder, W95_98\SDK\Disk 1, and finally Setup.exe.
9. Click Finish and agree to all defaults.
10. When the installation is complete, close the newly created program group, and an option to configure will be displayed. Use this option *only* if hardware settings need to be changed. You can change hardware settings by either using the System Resources button in the IEEE applet or by clicking System, Device Manager, then highlighting the device and clicking Properties, Resources, and Change Settings.
11. Power down the computer. For the ISA version, install the configured card, power up the computer, and proceed to the Verification process. For plug and play, simply install the card. Windows will automatically detect and associate the previously installed software.

Installing the Interface Software Support Files

1. In the Browse window, double click the appropriate product folder and operating system. The installation program SETUP.EXE will be in the subfolder named SDK under the Windows 95/98 operating system folder.
2. Locate SETUP.EXE and click FINISH.
3. The Installation program will step you through various options on installing these software support files.

NOTE

These files are NOT required to get the hardware to work properly, but we recommend using them if you plan to do any software development or if you need Help files.

4. Any or all of the installed software support files may be removed by going to the Control Panel from the Start > Settings menu, double-clicking on the Add/Remove Programs icon, then selecting "Personal IEEE 488 v 2.0," and clicking the Add/Remove... button.

At this point, the installation of software support files is complete.

Verification

- 1a. Run wintest.exe. "IEEE0(-1)" should be in the device handle box. Hit "OpenName" (immediately to the right of the device handle box). Does this open a new window with driver information? If yes, go to Step 2. If no, go to Step 1b.
- b. Check to make sure that the driver has been installed. (It's on a separate disk for the Windows 95 or 98 environment.)
- c. Check the address and interrupt (interrupt is set with switches and jumpers on ISA cards). On the card, make sure these settings match the settings in the control panel.
- d. Check the BIOS to see if the interrupt is reserved for an ISA card.
- e. Check to make sure that it is a Black Box card...look for a part number etched in the solder side of the board. It should look like a phone number.

- f. Check for conflicts with older revisions of the driver. (Check autoexec.bat for a call to the old .DRV version, also the path for different installation or a wrong path.) A successful test in steps 1 through 6 means that the card is an IOTech card, the address is OK. Also, it's not colliding with any of the other drivers, and the interrupt is probably OK.
2. Make sure there is a device on the bus, "Query→CheckListener." Fill in the primary parameter with the address of the peripheral. This will return a 0, 1, or -1.

If it returns 1, CheckListener was successful. Go to step 3.

If it returns 0 immediately, the device did not respond. Check the cabling, and the address of the peripheral.

If it returns 0 after the time-out period or returns -1, this is almost always an interrupt conflict. Check BIOS again, move the device to a "free" interrupt, and try again.

3. Go to "Device→MakeNew Device." Fill in the primary parameter with the address of the peripheral. There is no reason for this command to fail: it should return a number other than -1.
- 4a. Go to "Data Transfer→Output Commands→Output." Enter a short command to the peripheral, preferably one that can be observed on the device. Was the Bytecount correct? Was the command understood? If yes, go to step 5a. If no, go to step 4b.
- b. Check the DMA for conflicts, or disable it.
- c. Verify that the terminators match the peripheral settings.
- 5a. Write a message that gives a response back to the controller (take a reading, or check a setting). Then read that response with "Data Transfer→Output Commands→Enter." Did you get anything? Does it make sense?
 - b. If "Yes, yes->" go to step 6a.
 - c. If "Yes, no->" make sure that the DMA is disabled, and check the terminators.
 - d. If "No, no->" make sure a valid command was sent. Check the terminators.

- 6a. Engage the DMA, and send a long command (about 9 bytes or longer) or request a long response (same 9-byte threshold). Did it work? If yes, you are done. If no, go to step b.
- b. Change the DMA channel, and see if you need to reserve this resource in BIOS. Alternatively, you can leave the DMA disabled; this only costs some speed performance on longer transfers.

5.2 Windows NT Users Only

5.2.1 PLUG AND PLAY AND “LEGACY” DEVICES

NOTE

This section is for users of Windows NT. If you're using Windows 95/98, see Section 5.1.

1. Remove power from the PC.
2. Physically configure and install the device as described in **Chapter 3** or **4**.
3. Return power to the PC.
4. Insert the CD-ROM into your CD drive. The CD contains a folder for all of the IEEE488 interfaces. Located under each interface folder are the operating system folders. Using Windows Explorer, browse the \WNT\Disk1 folder for **Setup.exe**.
5. Run **Setup.exe**.
6. Follow the on-screen prompts and allow Windows NT to complete the install.



Figure 5-10. Driver Setup screen.

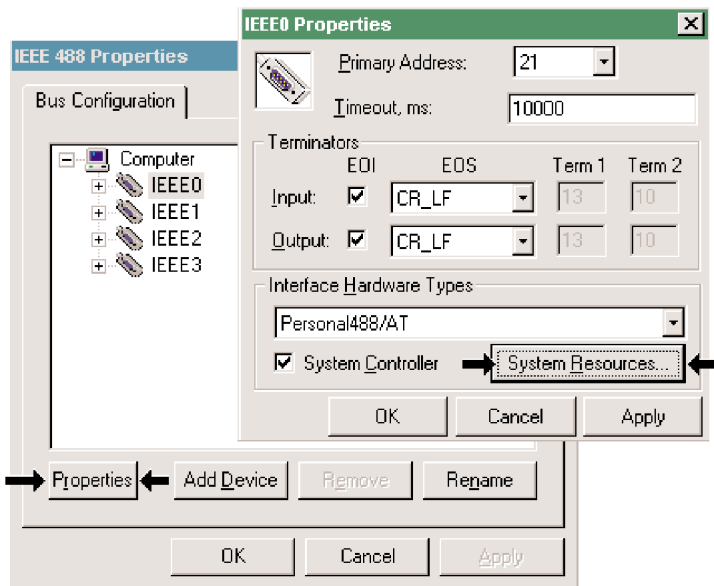


Figure 5-11. Selecting the Card.

7. In Windows NT, after selecting the device from the IEEE 488 Properties window, click the Properties button to access the device properties window. Then click the System Resources button to make changes to I/O, IRQ, and DMA settings as applicable. **Figure 5-11** shows selection of an AT488 type device designated as IEEE0.

Figure 5-12 represents the Windows NT Change System Resources dialog box, and the button used to access it.

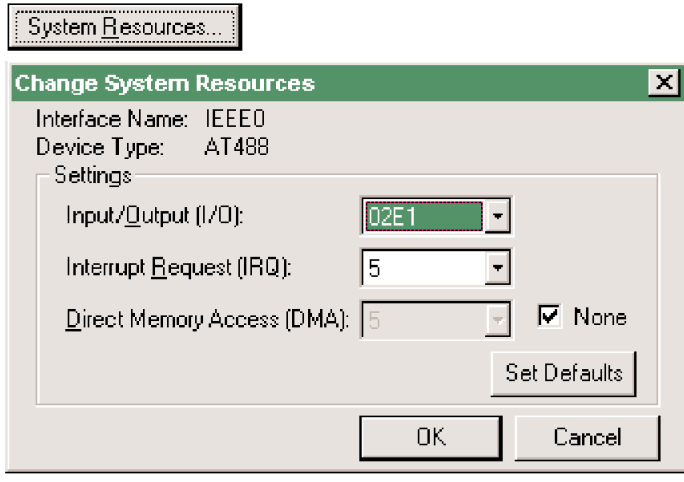


Figure 5-12. Windows NT Change System Resources screen.

8. When prompted, restart your computer to put the new settings into effect.

This completes the install procedure for users of Windows NT.

5.2.2 WINDOWS NT SERVICE PACKET 3 (SP3) DRIVER INSTALLATION/REMOVAL FOR IEEE CONTROLLERS

This procedure assumes that you have the most recent revision of the driver/software. You must remove any previous versions of drivers and applicable software before you begin.

Removal

Perform the removal steps only if you have a previous installation or if you made a mistake during installation.

1. Uninstall the software using Add/Remove Programs in the control panel. The keywords of the program name will be “Personal IEEE.”
2. Highlight and remove.
3. Answer yes to uninstall shared files if queried.
4. Shut down and Restart.

Installation

For all 488 controllers: Do NOT install the controller at this time. Follow these steps:

1. Go to Control Panel, select Add/Remove Programs, click Install, and Next. Windows NT will then default to the A drive.
2. Browse to the CD drive and select the appropriate controller folder, \WNT\Disk 1, and finally Setup.exe. The full path of the installation program is now displayed in the text box.
3. Click Finish and then Next. The Select Components window is now displayed.
4. Select the appropriate controller.
5. Click Next, and accept defaults.
6. When the installation is complete, close the newly created program group, and click Finish. (For the ISA version, use this IEEE 488 properties window to enter the system resources and change hardware settings as needed.)

NOTE

We recommend that you disable the DMA until you establish a non-conflicting IRQ.

7. Power down the computer.
8. Install the configured card and power up the computer. Proceed to the verification process.

NOTE

You might need to repeat the power down and settings routine until there is no conflict. Remember that not only the DIP switches and jumpers, but also the system resources, must also reflect the new settings.

Verification

- 1a. Run `wintest.exe`. “IEEE0 (-1)” should be in the device handle box.
- b. Hit “OpenName” (immediately to the right of the device handle box. Does this open a new window with driver information? If yes, go to step 2.
- c. If no, check to make sure that the driver has been installed. (It’s on a separate disk for the Windows 95 or 98 environment.)
- d. Check the address and interrupt (interrupt is set with switches and jumpers on ISA cards) on the card. Make sure that they match the settings in the control panel.
- e. Check BIOS to see if the interrupt is reserved for an ISA card.
- f. Check to make sure that it is an IOtech card. Look for a part number etched in the solder side of the board. It should look like a phone number.
- g. Check for conflicts with older revisions of the driver. (Check `autoexec.bat` for a call to the old `.DRV` version, also the path for a different installation.)

A successful test in step 1 means we know it is an IOtech card, the address is OK. Also, it is not colliding with any of our other drivers, and the interrupt is probably OK.

- 2a. Make sure that there is a device on the bus, “Query→CheckListener.”
- b. Fill in the primary parameter with the address of the peripheral. This will return a 0, 1, or -1.

If it returns 1, CheckListener was successful.

If it returns 0 immediately, the device did not respond. Check the cabling and address of the peripheral.

If it returns 0 (after the time-out period), or returns 1, this is almost always an interrupt conflict. Check BIOS again, move the device to a different “free” interrupt and try again.

3. Go to “Device→MakeNewDevice.” Fill in the primary parameter with the address of the peripheral. There is no reason for this command to fail; it should return a number other than -1.
4. Go to “DataTransfer→Output command→Output.” Enter a short command to the peripheral, preferably one that you can observe on the device. Was the

Bytecount correct? Was the command understood? If yes, go to step 5. If no, check the DMA for conflicts, or disable it. Verify that the terminators match the peripheral settings.

5. Write a message that gives a response back to the controller (take a reading, or check a setting). Then read that response with: “DataTransfer→Output commands→Enter.” Did you get anything? Does it make sense?
 - a. If “Yes, yes->” go to step 6.
 - b. If “Yes, no->” make sure that the DMA is disabled, and check the terminators.
 - c. If “No, no->” make sure a valid command was sent, and check the terminators.
- 6a. Engage the DMA, and send a long command (about 9 bytes or longer) or request a long response (same as 9-byte threshold). Did it work? If yes, you are done. If no, go to step b.
- b. Change the DMA channel, and see if you need to reserve this resource in BIOS. Alternatively, you can leave the DMA disabled; this only costs some speed performance on longer transfers.

6. API Command Reference

6.1 Introduction

This chapter contains the API command reference for Driver488/W95 and Driver488/WNT, using the C language. The following 67 commands are presented in alphabetical order for ease of use.

6.2 Abort

Syntax	INT WINAPI Abort(DevHandleT devHandle); devHandle refers to either an IEEE 488 hardware interface or an external device. If devHandle refers to an external device, the Abort command will act on the hardware interface to which the external device is attached.
Returns	-1 if error
Mode	SC or *SC(CA)
Bus States	IFC, *IFC (if SC) ATN(MTA (if *SC(CA)
Example	errorflag = Abort(ieee);
See Also	MyTalkAddr, Talk, UnTalk

As the System Controller (SC), whether Driver488 is the Active Controller or not, the Abort command causes the Interface Clear (IFC) bus management line to be asserted for at least 500 microseconds. By asserting IFC, Driver488 regains control of the bus even if one of the devices has locked it up during a data transfer. Asserting IFC also makes Driver488 the Active Controller. If a non-System-Controller was the Active Controller, it is forced to relinquish control to Driver488. Abort forces all IEEE 488 device interfaces into a quiescent state.

If Driver488 is a non-System-Controller in the Active Controller state (*SC(CA), it asserts Attention (ATN), which stops any bus transactions, and then sends its My Talk Address (MTA) to “Untalk” any other Talkers on the bus. It does not (and cannot) assert IFC.

6.3 Arm

Syntax	INT WINAPI Arm(DevHandleT devHandle, ArmCondT condition); devHandle refers to either an IEEE 488 hardware interface or an external device. If devHandle refers to an external device, the Arm command acts on the hardware interface to which the external device is attached. condition is one of the following: acError, acSRQ, acPeripheral, acController, acTrigger, acClear, acTalk, acListen, acIdle, acByteIn, acByteOut, or acChange.
Returns	-1 if DevHandleT is an illegal device or interface; otherwise, the current state of the event trigger flag
Mode	Any
Bus States	None
Example	errorflag = Arm(ieee, acSRQ acTrigger acChange);
See Also	Disarm, OnEvent

The Arm command allows Driver488 to signal to the user-specified function when one or more of the specified conditions occurs. Arm sets a flag for each implementation of the conditions which are user-indicated. Arm conditions may be combined using the bitwise OR operator.

The following Arm conditions are supported:

Condition	Description
acSRQ	The Service Request bus line is asserted.
acPeripheral	An addressed status change has occurred, and the interface is a Peripheral.
acController	An addressed status change has occurred, and the interface is an Active Controller.
acTrigger	The interface has received a device Trigger command.
acClear	The interface has received a device Clear command.
acTalk	An addressed status change has occurred, and the interface is a Talker.
acListen	An addressed status change has occurred, and the interface is a Listener.
acIdle	An addressed status change has occurred, and the interface is neither Talker nor Listener.
acByteIn	The interface has received a data byte.
acByteOut	The interface has been configured to output a data byte.
acError	A Driver488 error has occurred.
acChange	The interface has changed its addressed status. Its Controller/Peripheral or Talker/Listener/Idle states has changed.

6.4 AutoRemote

Syntax	<pre>INT WINAPI AutoRemote(DevHandleT devHandle, BOOL flag);</pre> <p>devHandle refers to either an IEEE 488 hardware interface or an external device. If devHandle refers to an external device, the AutoRemote command acts on the hardware interface to which the external device is attached.</p> <p>flag may be either OFF or ON</p>
Returns	-1 if DevHandleT is an illegal device or interface; otherwise, the previous state is returned
Mode	SC
Bus States	None
Example	<pre>errorcode = AutoRemote(ieee,ON);</pre>
See Also	Local, Remote, EnterX, OutputX

The AutoRemote command enables or disables the automatic assertion of the Remote Enable (REN) line by Output. When AutoRemote is enabled, Output automatically asserts REN before transferring any data. When AutoRemote is disabled, there is no change to the REN line. AutoRemote is on by default.

6.5 Buffered

Driver488/W95 only

Syntax	LONG WINAPI Buffered(DevHandleT devHandle);
devHandle	refers to either an IEEE 488 hardware interface or an external device. If devHandle refers to an external device, the Buffered command acts on the hardware interface to which the external device is attached.
Returns	-1 if error; otherwise long integer from 0 to 1,048,575(220-1)
Mode	Any
Bus States	None
Example	<pre>result = Buffered(ieee); printf(“%ld bytes were received.”,result);</pre>
See Also	EnterX, OutputX

The Buffered command returns the number of characters transferred by the latest Enter, Output, SendData, or SendEoi command. If an asynchronous transfer is in progress, the result is the number of characters that have been transferred at the moment the command is issued. This command is most often used after a counted Enter, EnterN, EnterNMore, etc., to determine if the full number of characters was received, or if the transfer terminated upon detection of term. It is also used to find out how many characters have currently been sent during an asynchronous DMA transfer.

6.6 BusAddress

Syntax	INT WINAPI BusAddress (DevHandleT devHandle, BYTE primary, BYTE secondary).
devHandle	refers to either an IEEE 488 hardware interface or an external device. primary is the IEEE 488 bus primary address of the specified device. secondary is the IEEE 488 bus secondary address of the specified device. If the specified device is an IEEE 488 hardware interface, this value must be -1, since there are no secondary addresses for the IEEE 488 hardware interface. For no secondary address, a -1 must be specified.
Returns	-1 if error
Mode	Any
Bus States	None
Example	<code>errorcode = BusAddress(dmm,14,0);</code>
See Also	MakeDevice

The BusAddress command sets the IEEE 488 bus address of the IEEE 488 hardware interface or an external device. Every IEEE 488 bus device has an address that must be unique within any single IEEE 488 bus system. The default IEEE 488 bus address for Driver488 is 21, but this may be changed if it conflicts with some other device.

6.7 CheckListener

Syntax	<pre>INT WINAPI CheckListener(DevHandleT devHandle, BYTE primary, BYTE secondary);</pre> <p>devHandle refers to either an IEEE 488 hardware interface or an external device. If devHandle refers to an external device, the CheckListener command acts on the hardware interface to which the external device is attached.</p> <p>primary is the primary bus address to check for a Listener (00 to 30)</p> <p>secondary is the secondary bus address to check for a Listener (00 to 31). For no secondary address, a -1 must be specified</p>
Returns	-1 if error; otherwise it returns a 1 if a listener was found at the specified address, or a 0 if a listener was not found at the specified address.
Mode	CA
Bus States	ATN(UNL, LAG, (check for NDAC asserted)
Example	<pre>result = CheckListener(ieee,15,4); if (result == 1) { printf("Device found at specified address.\n"); } if (result == 0) { printf("Device not found at specified address.\n"); }</pre>
See Also	FindListener, BusAddress

The CheckListener command checks for the existence of a device on the IEEE 488 bus at the specified address.

6.8 Clear

Syntax	<pre>INT WINAPI Clear(DevHandleT devHandle);</pre> <p>devHandle refers to either an IEEE 488 hardware interface or an external device. If devHandle refers to a hardware interface, then a Device Clear (DCL) is sent. If devHandle refers to an external device, a Selected Device Clear (SDC) is sent.</p>
Returns	-1 if error
Mode	CA
Bus States	<p>ATN(DCL (all devices))</p> <p>ATN(UNL, MTA, LAG, SDC (selected device))</p>
Examples	<pre>errorcode = Clear(ieee);</pre> <p>Sends the Device Clear (DCL) command to the IEEE interface board.</p> <pre>errorcode = Clear(wave);</pre> <p>Sends the Selected Device Clear (SDC) command to the WAVE device.</p> <pre>errorcode = Clear(dmm);</pre> <p>Sends the Selected Device Clear (SDC) command to the DMM device.</p>
See Also	Reset, ClearList

The Clear command causes the Device Clear (DCL) bus command to be issued to an interface or a Selected Device Clear (SDC) command to be issued to an external device. IEEE 488 bus devices that receive a Device Clear or Selected Device Clear command normally reset to their power-on state.

6.9 ClearList

Syntax	<code>INT WINAPI ClearList(DevHandlePT dhList);</code> dhList is a pointer to a list of device handles that refer to external devices. If a hardware interface is in the list, DCL is sent instead of SDC.
Returns	-1 if error
Mode	CA
Bus States	ATN(DCL (all devices)) ATN(UNL, MTA, LAG, SDC (selected device))
Example	<code>deviceList[0] = wave;</code> <code>deviceList[1] = scope;</code> <code>deviceList[2] = dmm;</code> <code>deviceList[3] = NODEVICE;</code> <code>errorcode = ClearList(deviceList);</code> Sends the Selected Device Clear (SDC) command to a list of devices.
See Also	Clear, Reset

The ClearList command causes the Selected Device Clear (SDC) command to be issued to a list of external devices. IEEE 488 bus devices that receive a Selected Device Clear command normally reset to their power-on state.

6.10 Close

Syntax	INT WINAPI Close(DevHandleT devHandle); devHandle refers to either an IEEE 488 interface or an external device.
Returns	-1 if error
Mode	Any
Bus States	Completion of any pending I/O activities
Example	errorcode = Close(wave);
See Also	OpenName, MakeDevice, Wait

The Close command waits for I/O to complete, flushes any buffers associated with the device that is being closed, and then invalidates the handle associated with the device.

6.11 ControlLine**Syntax** INT WINAPI ControlLine(DevHandleT devHandle);

ControlLine returns a bit mapped number.

devHandle refers to the I/O adapter. If devHandle refers to an external device, the ControlLine command acts on the hardware interface to which the external device is attached.

Returns -1 if error; otherwise, a bit map of the current state of the IEEE 488 interface. Under 32-bit Driver488 software, serial interfaces are not supported.**Mode** Any**Bus States** None**Example**
result = ControlLine(ieee);

printf("The response is %X\n",result);**See Also** TimeOut

The ControlLine command may be used only on IEEE 488 devices. Under 32-bit Driver488 software, serial interfaces are not supported. This command returns the status of the IEEE 488 bus control lines as an 8-bit unsigned value (bits 2 and 1 are reserved for future use), as shown below:

Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
EOI	SRQ	NRFD	NDAC	DAV	ATN	0	0

6.12 DigArm

PCI Card only

Syntax	INT WINAPI DigArm(DevHandleT devHandle, BOOL bArm);
	devHandle refers to an interface handle.
	bArm refers to a value that arms or disarms event generation. TRUE = Arm, FALSE = Disarm.
Returns	-1 if neither nibble is set for input, or other error
Mode	Any
Bus States	None
Example	DigArm(devHandle, TRUE); Arms digital input event generation.
See Also	DigArmSetup, DigSetup, OnDigEvent, OnDigEventVDM

The DigArm command arms or disarms the event-generation due to a digital I/O port match condition. The caller should configure the digital I/O port, the event-callback mechanism, and the match condition prior to arming the event generation. The following code snippet illustrates this sequence:

```
DigSetup(devHandle, FALSE, FALSE); // Configure both nibbles for input.
OnDigEventVDM(devHandle, MyFunc, 0); // On event, call function MyFunc.
DigArmSetup(devHandle, 0x0A5); // Trigger when inputs equals 0xA5.
DigArm(devHandle, TRUE); // Enable event generation.
```

Event generation is automatically disarmed when an event is triggered. The event-generation configuration, however, remains intact, so event generation can be re-armed just by calling DigArm. The other steps shown in the above code snippet do not need to be repeated unless the event configuration is to be changed.

Event generation may be disarmed (bArm = FALSE) at any time.

NOTES

1. This function does not configure the digital I/O port for input. The caller must use `DigSetup` to configure the port for input before performing arming event generation. If neither nibble is configured for input, the function returns -1 and sets the error code to `IOT_BAD_VALUE2`.
2. Event generation may be re-armed from within the event handler to provide continuous detection of match condition events. However, this is not guaranteed to catch every event if the digital input values are rapidly changing.
3. Any digital I/O port bits configured for output are treated as “don’t care” bits for the purposes of event generation. In other words, it is valid to arm an event when only one nibble of the port is configured for input. In this case, the other nibble is ignored when detecting the match condition.

6.13 DigArmSetup

PCI Card only

Syntax	<code>INT WINAPI DigArmSetup(DevHandleT devHandle, BYTE byMatchValue);</code> devHandle refers to an interface handle. byMatchValue refers to a value that is compared against the digital I/O inputs.
Returns	-1 if error
Mode	Any
Bus States	None
Example	<code>DigArmSetup(devHandle, 0xA5);</code> Sets the match value to 0xA5.
See Also	<code>DigArm</code> , <code>DigSetup</code>

The DigArmSetup command sets the match condition value. This value will be compared against the digital I/O port inputs to detect when an event occurs. The event must be armed (via DigArm) for event notification to take place.

The comparison operation depends on the current digital-port configuration. If both nibbles are configured for input, then the match value is compared to the entire byte value of the digital port. If only one of the nibbles is configured for input, then the value is compared against just that nibble. If no nibbles are configured for input, then the match value is ignored. The DigArm function will not allow event generation to be armed unless at least one of the nibbles is configured for input.

6.14 DigRead

PCI Card only

Syntax	INT WINAPI DigRead(DevHandleT devHandle);
	devHandle refers to an interface handle.
Returns	-1 if no part of the port is configured for input, or other error
	otherwise, integer between 0 and 255 if the entire digital I/O port is configured for input; or integer between 0 and 15 if only one nibble (either low or high) is configured for input
Mode	Any
Bus States	None
Example	int i = DigRead(devHandle); Returns the current value of the digital I/O port per the current configuration.
See Also	DigSetup, DigWrite

The DigRead command reads the current value of the digital IO port per the input/output configuration of the port. If the entire port is configured for input, a value between 0 and 255 is returned. If either the upper or lower nibble is

configured for input, and the other for output, a value between 0 and 15 is returned.

NOTE

This function does not configure the digital I/O port for input. The caller must use `DigSetup` to configure the port for input before performing any reads. If neither nibble is configured for input the function returns -1 and sets the error code to `IOT_BAD_VALUE2`.

6.15 DigSetup

PCI Card only

Syntax	<code>INT WINAPI DigSetup(DevHandleT devHandle, BOOL bLowOut, BOOL bHighOut);</code> devHandle refers to an interface handle. bLowOut refers to the lower nibble setup. TRUE = output, FALSE = input. bHighOut refers to the upper nibble setup. TRUE = output, FALSE = input.
Returns	-1 if error
Mode	Any
Bus States	None
Examples	<code>DigSetup(devHandle, TRUE , TRUE);</code> All 8 bits output. <code>DigSetup(devHandle, FALSE, TRUE);</code> Lower 4 bits input, upper 4 output. <code>DigSetup(devHandle, TRUE , FALSE);</code> Lower 4 bits output, upper 4 input. <code>DigSetup(devHandle, FALSE, FALSE);</code> All 8 bits input.
See Also	<code>DigRead</code> , <code>DigWrite</code>

The DigSetup command configures the digital I/O port for input and output on a per-nibble basis. Each of the two nibbles can be set for input or output. All combinations are supported. Once DigSetup is called, the configuration of the digital I/O port does not change until the next call to DigSetup. The port may be read and written to many times without affecting the port setup.

NOTE

The digital I/O port must be configured every time the driver is opened. The configuration is not stored between sessions.

6.16 DigWrite

PCI Card only

Syntax INT WINAPI DigWrite(DevHandleT devHandle, BYTE byDigData);

devHandle refers to an interface handle.

byDigData refers to a value to write to the digital output port, where the integer range is between 0 and 255 if the entire digital I/O port is configured for output, or between 0 and 15 if only one nibble (either low or high) is configured for output.

Returns -1 if no part of the digital I/O port is configured for output.

Mode Any

Bus States None

Example DigRead(devHandle, 0x0A);

Writes the given value to the digital I/O port per the current configuration.

See Also DigSetup, DigRead

The DigWrite command writes the given value to the digital I/O port per the input/output configuration of the port. If the entire port is configured for output, then the data value with a range from 0 to 255 is written to the port. If either the

upper or lower nibble is configured for input, and the other for output, then the data value is truncated to the range from 0 to 15 and it is written to the appropriate nibble per the current configuration.

NOTES

1. This function does not configure the digital I/O port for output. The caller must use `DigSetup` to configure the port before performing any reads or writes. If neither nibble is configured for output the function returns -1 and sets the error code to `IOT_BAD_VALUE2`.

2. Outputs do not persist after an interface is closed. At that time, all digital I/O lines are configured for input.

6.17 Disarm

Syntax INT WINAPI Disarm(DevHandleT devHandle, ArmCondT condition);

`devHandle` refers to either an IEEE 488 interface or an external device. If `devHandle` refers to an external device, then the Disarm command acts on the hardware interface to which the external device is attached.

`condition` specifies which of the conditions are no longer to be monitored. If `condition` is 0, then all conditions are Disarmed.

Returns -1 if error; otherwise, the current bit map of the event condition mask.

Mode Any

Bus States None

Examples `errorcode=Disarm(ieee,acTalk|acListen|acChange);`
`errorcode=Disarm(ieee,0);`

See Also Arm, OnEvent

The Disarm command prevents Driver488 from invoking an event handler and interrupting the PC, even when the specified condition occurs. Your program can still check for the condition by using the Status command. If the Disarm command is invoked without specifying any conditions, then all conditions are disabled. The Arm command may be used to re-enable interrupt detection.

6.18 EnterX

Syntax

```
LONG WINAPI EnterX(DevHandleT devHandle, LPBYTE  
data,DWORD count,BOOL forceAddr,TermT*term,BOOL  
async,LPDWORD compStat);
```

devHandle refers to either an IEEE 488 interface or an external device.

data is a pointer to the buffer into which the data is read.

count is the number of characters to read.

forceAddr is used to specify whether the addressing control bytes are to be issued for each EnterX command.

term is a pointer to a terminator structure that is used to set up the input terminators. If term is set to 0, the default terminator is used.

async is a flag that allows asynchronous data transfer. Note that this asynchronous flag is ignored in Driver488/WNT.

compStat is a pointer to an integer containing completion-status information.

Returns	-1 if error otherwise, the actual count of bytes transferred. The memory buffer pointed to by the data parameter is filled in with the information read from the device. Note that the actual count does not include terminating characters if term characters are specified by the term in function. In addition, term characters are not returned but are discarded.
Mode	CA
Bus States	With interface handle: *ATN, data With external device handle: ATN (UNL, MLA, TAG, *ATN, data)
Example	<pre>term.EOI = TRUE; term.nChar = 1; term.EightBits = TRUE; term.termChar[0] = '\r'; bytecount=EnterX(timer,data,1024,0,&term,1,&stat);</pre>
See Also	OutputX, Term, Buffered

NOTE

The asynchronous flag async is ignored in Driver488/WNT.

The EnterX command reads data from the I/O adapter. If an external device is specified, then Driver488 is addressed to Listen, and that device is addressed to Talk. If an interface is specified, then Driver488 must already be configured to receive data and the external device must be configured to Talk, either as a result of an immediately preceding EnterX command or as a result of one of the Send commands. EnterX terminates reception on either the specified count of bytes transferred, or the specified or default terminator being detected. Terminator characters, if any, are stripped from the received data before the EnterX command returns to the calling application.

The forceAddr flag is used to specify whether the addressing control bytes are to be issued for each EnterX command. If the device handle refers to an I/O adapter, then forceAddr has no effect and command bytes are not sent. For an external device, if forceAddr is TRUE then Driver488 always sends the UNL, MLA, and TAG command bytes. If forceAddr is FALSE, then Driver488 compares the current device with the previous device that used that interface adapter board for an EnterX command. If they are the same, then no command bytes are sent. If they are different, then EnterX acts as if the forceAddr flag were TRUE and sends the command bytes. The forceAddr flag is usually set TRUE for the first transfer of data from a device, and then set FALSE for additional transfers from the same block of data from that device.

Additional Enter Functions

Driver488 provides additional Enter routines that are short-form versions of the EnterX function. The following Enter functions are already defined in your header file.

ENTER

Syntax LONG WINAPI Enter(DevHandleT devHandle, LPBYTE data)

Remarks Enter is equivalent to the following call to EnterX:

EnterX(devHandle,data,sizeof(data),1,0L,0,0L);

The Enter function passes the device handle and a pointer to the data buffer to the EnterX function. It determines the size of the data buffer that you provided, and passes that value as the count parameter. It specifies forceAddr is TRUE, causing Driver488 to re-address the device. The default terminators are chosen by specifying a 0 as the term parameter. Asynchronous transfer is turned off by sending 0 for the async parameter, and the completion status value is ignored by sending 0 for the compStat parameter.

ENTERN

Syntax LONG WINAPI EnterN(DevHandleT devHandle,LPBYTE data,int count)

Remarks EnterN is equivalent to the following call to EnterX:
EnterX(devHandle,data,count,1,0L,0,0L);

The EnterN function passes the device handle, the pointer to the data buffer, and the size of the data buffer to the EnterX function. It specifies forceAddr is TRUE, causing Driver488 to re-address the device. The default terminators are chosen by specifying a 0 pointer as the term parameter. Asynchronous transfer is turned off by sending 0 for the async parameter, and the completion status value is ignored by sending 0 for the compStat parameter.

ENTERMORE

Syntax LONG WINAPI EnterMore(DevHandleT devHandle,LPBYTE data)

Remarks EnterMore is equivalent to the following call to EnterX:
EnterX(devHandle,data,sizeof(data),0,0L,0,0L);

The EnterMore function passes the device handle and the pointer to the data buffer to the EnterX function. It determines the size of the data buffer that you provided, and passes that value as the count parameter. It specifies forceAddr is FALSE, therefore, Driver488 does not address the device if it is the same device as previously used. The default terminators are chosen by specifying a 0 as the term parameter. Asynchronous transfer is turned off by sending 0 for the async parameter, and the completion status value is ignored by sending 0 for the compStat parameter.

ENTERNMORE

Syntax LONG WINAPI EnterNMore(DevHandleT devHandle,LPBYTE data,int count);

Remarks EnterNMore is equivalent to the following call to EnterX:
EnterX(devHandle,data,count,0,0L,0,0L);

The EnterNMore function passes the device handle, the pointer to the data buffer, and the size of the data buffer to the EnterX function. It specifies forceAddr is FALSE; therefore, Driver488 does not address the device if it is the same device as previously used. The default terminators are chosen by specifying a 0 as the term parameter. Asynchronous transfer is turned off by sending 0 for the async parameter, and the completion status value is ignored by sending 0 for the compStat parameter.

6.19 Error

Syntax	INT WINAPI Error(DevHandleT devHandle, BOOL display);
	devHandle refers to either an IEEE 488 interface or an external device.
	display indicates whether the error message display should be ON or OFF.
Returns	-1 if error
Mode	Any
Bus States	None
Example	errorcode = Error(ieee, OFF);
See Also	OnEvent, GetError, GetErrorList, Status

The Error command enables or disables automatic on-screen display of Driver488 error messages. Specifying ON enables the error message display, while specifying OFF disables the error message display. Error ON is the default condition.

6.20 FindListeners

Syntax

INT WINAPI FindListeners(DevHandleT devHandle,
BYTE primary, LPWORD listener, DWORD limit);

devHandle refers to either an IEEE 488 interface or an external device. If devHandle refers to an external device, then the FindListeners command acts on the hardware interface to which the external device is attached.

primary is the primary IEEE 488 bus address to check.

listener is a pointer to a list that contains all Listeners found on the specified interface board. You must allocate enough memory to accommodate all of the possible Listeners up to the limit that he specified.

limit is the maximum number of Listeners to be entered into the Listener list.

Returns

-1 if error
otherwise, the number of Listeners found on the interface

Mode

Any

Bus States

ATN(MTA, UNL, LAG

Example

WORD listeners[5];

errorcode = FindListeners(ieee,10,listeners,5);

See Also

CheckListener, BusAddress, Status

The FindListeners command finds all of the devices configured to Listen at the specified primary address on the IEEE 488 bus. The command first identifies the primary address to check and returns the number of Listeners found and their addresses. Then, it fills the user-supplied array with the addresses of the Listeners

found. The number of Listeners found is the value returned by the function. The returned values include the secondary address in the upper byte, and the primary address in the lower byte. If there is no secondary address, then the upper byte is set to 255; hence, a device with only a primary address of 16 and no secondary address is represented as 0xFF10 or -240 decimal.

6.21 Finish

Driver488/W95 only

Syntax

```
INT WINAPI Finish(DevHandleT devHandle);
```

devHandle refers to either an IEEE 488 interface or an external device. If devHandle refers to an external device, the Finish command acts on the hardware interface to which the external device is attached.

Returns

-1 if error

Mode

CA

Bus States

ATN

Example

```
errorcode = Finish(ieee);
```

See Also

Resume, PassControl

The Finish command asserts Attention (ATN) and releases any pending holdoffs after a Resume function is called with the monitor flag set.

6.22 GetError

Syntax	<code>ErrorCodeT WINAPI GetError(DevHandleT devHandle, LPSTR errText);</code> devHandle refers to either the IEEE 488 interface or the external device that has the associated error. errText is the string that will contain the error message. If errText is non-null, the string must contain at least 247 bytes.
Returns	-1 if error otherwise, it returns the error code number associated with the error for the specified device.
Mode	Any
Bus States	None
Example	<pre>errnum = GetError(ieee,errText); printf("Error number:%d;%s \n",errnum,errText);</pre>
See Also	Error, GetErrorList, Status

The GetError command is user-called after another function returns an error indication. The device handle sent to the function that returned the error indication is sent to GetError as its devHandle parameter. GetError finds the error associated with that device and returns the error code associated with that error. If a non-null error text pointer is passed, GetError also fills in up to 247 bytes in the string. The application must ensure that sufficient space is available.

6.23 GetErrorList

Syntax	<pre>ErrorCodeT WINAPI GetErrorList(DevHandlePT dhList, LPSTR errText, DevHandlePT errHandle);</pre> <p>dhList is a pointer to a list of external devices that was returned from a function, due to an error associated with one of the external devices in the list.</p> <p>errText is the text string that contains the error message. You must ensure that the string length is at least 247 bytes.</p> <p>errHandle is a pointer to the device handle that caused the error.</p>
Returns	-1 if error; otherwise, it returns the error number associated with the given list of devices.
Mode	Any
Bus States	None
Example	<pre>char errText[329]; int errHandle; int errnum; result = ClearList(list); if (result == -1) { errnum=GetErrorList(list,errText,&errHandle); printf("Error %d;%s,at handle %d\n", errnum, errText, errHandle); }</pre>
See Also	Error, GetError, Status

The `GetErrorList` command is user-called, after another function identifying a list of device handles returns an error indication. The device handle list sent to the function that returned the error indication, is sent to `GetErrorList`. `GetErrorList` finds the device which returned the error indication, returning the handle through `errHandle`, and returns the error code associated with that error. If a non-null error text pointer is passed, `GetError` also fills in up to 247 bytes in the string. The application must ensure that sufficient space is available.

6.24 Hello

Syntax

```
INT WINAPI Hello(DevHandleT devHandle, LPSTR  
message);
```

`devHandle` refers to either an IEEE 488 interface or an external device. If `devHandle` refers to an external device, the `Hello` command acts on the hardware interface to which the external device is attached.

`message` is a character pointer that contains the returned message.

Returns

-1 if error; otherwise, the version of the Dynamic Link Library (DLL) and the version of the device driver. The returned byte count will never exceed 247 bytes.

Mode

Any

Bus States

None

Example

```
char message[247];  
result = Hello(ieee,message);  
printf("%s\n",message);
```

See Also

Status, OpenName, GetError

The Hello command is used to verify communication with Driver488, and to read the software revision number. If a non-null string pointer is passed, Hello fills in up to 247 bytes in the string. The application must ensure that sufficient space is available. When the command is sent, Driver488 returns a string similar to the following:

Driver488 Revision X.X (C)199X ...

where X is the appropriate revision or year number.

6.25 KeepDevice

Syntax	INT WINAPI KeepDevice(DevHandleT devHandle); devHandle refers to an external device.
Returns	-1 if error
Mode	Any
Bus States	None
Example	errorcode = KeepDevice(scope);
See Also	MakeDevice, MakeNewDevice, RemoveDevice, OpenName

NOTE

KeepDevice will update an existing device or will create a new device in the Registry. This update feature is new and useful. For example, if you wish to change the bus address of the device and make it a permanent change.

The KeepDevice command changes the indicated temporary Driver488 device to a permanent device, visible to all applications. Permanent Driver488 devices are not removed when Driver488 is closed. Driver488 devices are created by MakeDevice and are initially temporary. Unless KeepDevice is used, all temporary Driver488 devices are forgotten when Driver488 is closed. The only way to remove the permanent device once it has been made permanent by the KeepDevice command, is to use the RemoveDevice command.

6.26 Listen

Syntax	<code>INT WINAPI Listen(DevHandleT devHandle, BYTE pri, BYTE sec);</code> devHandle refers to either an IEEE 488 interface or an external device. If devHandle refers to an external device, the Listen command acts on the associated interface. pri and sec specify the primary and secondary addresses of the device which is to be addressed to listen.
Returns	-1 if error
Mode	CA
Bus States	ATN, LAG
Example	<code>errorcode = Listen (ieee, 12, -1);</code>
See Also	Talk, SendCmd, SendData, SendEoi, FindListener

The Listen command addresses an external device to Listen.

6.27 Local

Syntax	<code>INT WINAPI Local(DevHandleT devHandle);</code> devHandle refers to either an IEEE 488 interface or an external device.
Returns	-1 if error
Mode	SC
Bus States	*REN

Examples `errorcode = Local(ieee);` To unassert the Remote Enable (REN) line, the IEEE 488 interface is specified.

`errorcode = Local(wave);` To send the Go To Local (GTL) command, an external device is specified.

See Also LocalList, Remote, AutoRemote

In the System Controller mode, the Local command issued to an interface device causes Driver488 to unassert the Remote Enable (REN) line. This causes devices on the bus to return to manual operation. A Local command addressed to an external device places the device in the local mode via the Go To Local (GTL) bus command.

6.28 LocalList

Syntax `INT WINAPI LocalList(DevHandlePT dhList);`

`dhList` refers to a pointer to a list of external devices.

Returns -1 if error

Mode CA

Bus States ATN(UNL, MTA, LAG,GTL

Example `deviceList[0] = wave;`
 `deviceList[1] = timer;`
 `deviceList[2] = dmm;`
 `deviceList[3] = NODEVICE;`
 `errorcode = LocalList(deviceList);`
 Sends the Go To Local (GTO) bus command to a list of external devices.

See Also Local, Remote, RemoteList, AutoRemote

In the System Controller mode, the LocalList command issued to an interface device, causes Driver488 to unassert the Remote Enable (REN) line. This causes

devices on the bus to return to manual operation. A LocalList command addressed to an external device, places the device in the local mode via the Go To Local (GTL) bus command.

6.29 Lol

Syntax	INT WINAPI Lol(DevHandleT devHandle); devHandle refers to either an IEEE 488 interface or an external device. If devHandle refers to an external device, the Lol command acts on the hardware interface to which the external device is attached.
Returns	-1 if error
Mode	CA
Bus States	ATN(LLO
Example	errorcode = Lol(ieee);
See Also	Local, LocalList, Remote, RemoteList

The Lol command causes Driver488 to issue an IEEE 488 LocalLockout (LLO) bus command. Bus devices that support this command are thereby inhibited from being controlled manually from their front panels.

6.30 MakeDevice

Syntax	<pre>INT WINAPI MakeDevice(DevHandleT devHandle, LPSTR name);</pre> <p>devHandle refers to an existing external device.</p> <p>name is the device name of the device that is to be made and takes the configuration of the device given by devHandle.</p>
Returns	<p>-1 if error; otherwise, the DevHandleT of the new device. Note that the new device is an exact copy (except for the name) of the specified device as it currently exists in memory and not in the Registry.</p>
Mode	Any
Bus State	None
Example	<pre>dmm = MakeDevice(scope,"DMM"); BusAddress(dmm,16,-1);</pre> <p>Create a device named DMM, attached to the same I/O adapter as scope, and set its IEEE 488 bus address to 16.</p>
See Also	MakeNewDevice, KeepDevice, RemoveDevice, OpenName, Close

The MakeDevice command creates a new temporary Driver488 device that is an identical copy of an already existing Driver488 external device. The new device is attached to the same I/O adapter of the existing device and has the same bus address, terminators, timeouts, and other characteristics. The newly created device is temporary and is removed when Driver488 is closed. KeepDevice may be used to make the device permanent. To change the default values assigned to the device, it is necessary to call the appropriate configuration functions such as BusAddress, IOAddress, and TimeOut.

6.31 MakeNewDevice

Syntax DevHandleT WINAPI MakeNewDevice(LPSTR iName, LPSTR aName, BYTE primary, BYTE secondary, TermPT In, TermPT Out, DWORD tOut);

devHandle refers to the new external device.

iName is the user name of the interface on which the device is to be created.

aName is the user name of the device.

primary and secondary are the secondary and primary bus addresses to be specified. For no secondary address, a -1 must be specified.

In and Out are pointers to terminator structures specified to set up the respective input and output terminators of the device.

tOut is the timeout parameter to be specified.

Returns -1 if error; otherwise, the DevHandleT of the new device, based on the parameters specified.

Mode Any

Bus State None

Example DevHandleT anotherDevice;
anotherDevice = MakeNewDevice("IEEE0",
"Scope", 13, -1, NULL, NULL, 10000);
Specifies parameters for: Pointer to the interface, pointer to the device name, primary and secondary addresses, pointers to the term In and Out structures, and timeout in milliseconds.

See Also MakeDevice, KeepDevice, RemoveDevice, OpenName, Close

This function is similar to the MakeDevice function, except that MakeNewDevice will create a new device based on the parameters specified, instead of simply cloning an existing device.

The MakeNewDevice command does not save the parameters of the newly created device in the system registry. To save the device, call the KeepDevice function.

NOTE

The MakeNewDevice command will only create, not save, a new device. Interface descriptions are created and maintained by the configuration utility and the IEEE 488 configuration applet in the Windows Control Panel.

6.32 MyListenAddr

Syntax INT WINAPI MyListenAddr (DevHandleT devHandle);

devHandle refers to either an interface or an external device. If devHandle refers to an external device, the MyListenAddr command acts on the associated interface.

Returns -1 if error

Mode CA

Bus States ATN, MLA

Example errorcode = MyListenAddr (ieee);

See Also MyTalkAddr, Talk, Listen, SendCmd

The MyListenAddr command addresses the interface to Listen.

6.33 MyTalkAddr

Syntax	INT WINAPI MyTalkAddr (DevHandleT devHandle); devHandle refers to either an interface or an external device. If devHandle refers to an external device, the MyTalkAddr command acts on the associated interface.
Returns	-1 if error
Mode	CA
Bus States	ATN, MTA
Example	errorcode = MyTalkAddr (ieee);
See Also	MyListenAddr, Listen, SendCmd

The MyTalkAddr command addresses the interface to Talk.

6.34 OnDigEvent

PCI Card Only

Syntax	<pre>INT WINAPI OnDigEvent(DevHandleT devHandle, HWND hwnd, OpaqueP IParam);</pre> <p>devHandle refers to an interface handle.</p> <p>hwnd is the window handle to receive event notification.</p> <p>IParam value will be passed in the notification message.</p>
Returns	-1 if error
Mode	Any
Bus States	None
Example	<pre>OnDigEvent(devHandle, TRUE, 0x10L);</pre> <p>Sets the event notification to be via a window message to the specified window handle. The value 0x10 will be passed with the message.</p>
See Also	DigArm, OnDigEventVDM, OnEvent

The OnDigEvent command sets the handle of a window to receive a notification message when a digital match event is triggered. This function uses the same mechanism as the OnEvent command. For details, see the description of OnEvent.

NOTE

This function sets the event generation mechanism to be a window-notification message, replacing any previously defined event-notification mechanism. Only one event-notification mechanism can be used at one time.

6.35 OnDigEventVDM

PCI Card Only

Syntax	<pre>INT WINAPI OnDigEventVDM(DevHandleT devHandle, DigEventFuncT func, OpaqueP IParam);</pre> <p>devHandle refers to an interface handle.</p> <p>func is a user-defined function to be called when the digital match event is triggered.</p> <p>IParam value will be passed in the notification message.</p>
Returns	-1 if error
Mode	Any
Bus States	None
Example	<pre>OnDigEventVDM(devHandle, MyFunc, 0x10L);</pre> <p>Sets the event notification to be via a function call to the specified callback function. The value 0x10 will be passed to the function.</p>
See Also	DigArm, OnDigEventVDM, OnEventVDM

The OnDigEventVDM command sets the address of a “C”-style (__stdcall) function to be called when a digital match event occurs. This function uses a similar mechanism as the OnEventVDM command. The prototype of the callback function for OnDigEventVDM is:

```
void DigEventFunc(DevHandleT devHandle, LPARAM IParam)
```

The IParam value which is passed to OnDigEventVDM is passed on to the callback function when the event occurs. For details, see the description of OnEventVDM.

NOTE

This function sets the event-generation mechanism to be a callback function, replacing any previously defined event notification mechanism. Only one event-notification mechanism can be used at one time.

6.36 OnEvent

Syntax	<p>INT WINAPI OnEvent(DevHandleT devHandle, HWND hWnd, OpaqueP IParam);</p> <p>devHandle refers to either an interface or an external device.</p> <p>hWnd is the window handle to receive the event notification.</p> <p>IParam value will be passed in the notification message.</p>
Returns	-1 if error
Mode	Any
Bus States	None
Example	<pre>ieee = OpenName ("ieee"); OnEvent (ieee, hWnd, (OpaqueP) 12345678L); Arm (ieee, acSRQ acError); break;</pre>
See Also	OnEventVDM, Arm, Disarm

The OnEvent command causes the event handling mechanism to issue a message upon occurrence of an Armed event. The message will have a type of WM_IEEE488EVENT, whose value is retrieved via:

```
RegisterWindowMessage ((LPSTR) "WM_IEEE488EVENT");
```

The associated wParam is an event mask indicating which Armed event(s) caused the notification, and the lParam is the value passed to OnEvent. Note that although there is a macro for WM_IEEE488EVENT in the header file for each language, this macro resolves to a function call and therefore cannot be used as a case label. The preferred implementation is to include a default case in the message handling case statement and directly compare the message ID with WM_IEEE488EVENT. The following is a full example of a program using the OnEvent function:

```
LONG FAR WINAPI export
WndProc(HWND hWnd, unsigned iMessage, WORD wParam, LONG lParam);
{
HANDLE
ieee;
switch (iMessage)
{
case WM_CREATE:
ieee = OpenName ("ieee");
OnEvent (ieee, hWnd, (OpaqueP) 12345678L);
Arm (ieee, acSRQ | acError);
break;
default:
if (iMessage == WM_IEEE488EVENT) {
char buff [80];
wsprintf (buff, "Condition = %04X,
Param = %081X", wParam, lParam);
MessageBox (hWnd, (LPSTR) buff,
(LPSTR) "Event Noted", MB OK);
return TRUE;
}
}
}
```

6.37 OnEventVDM

Syntax	<p>INT WINAPI OnEventVDM(DevHandleT devHandle, EventFuncT func);</p> <p>devHandle refers to either an interface or an external device.</p> <p>func is a user-specified interrupt-handler function that is to perform some user-defined function, when one of the Armed conditions occur.</p>
Returns	-1 if error
Mode	Any
Bus States	None
Example	<p>Arm(ieeee0, acSRQ);</p> <p>OnEventVDM(ieeee0, srqHandler);</p> <p>Arms SRQ detection and sets up SRQ function handler</p>
See Also	OnEvent, Arm, Disarm

The OnEventVDM (VDM refers to Virtual DOS Machine) allows a call back to a user-specified function in a console mode application. The following is a full example of a console mode program using the OnEventVDM function:

```
#include <windows.h>
#include <stdio.h>
#include "iotieee.h"
// For debugging
#define qsk(v,x) (v=x, printf(#x "returned %d/n", v))
void
srqHandler(DevHandlerT devHandle, UINT mask)
{
    LONG xfered;
    printf("\007\n\nEVENT-FUNCTION on %d mask 0x%04x\n",
        devHandle, mask);
    qsk(xfered, Spoll(devHandle));
    printf("\n\n");
}
void
main(void)
{
    LONG result, xfered;
    int ioStatus, x;
    DevHandlerT ieee0, wave14, wave16;
    TermT myTerm;
    UCHAR buffer[500];
    printf("\n\nSRQTEST program PID %d\n",GetCurrentProcessId ());
    qsk(iee0, OpenName("iee0"));
    qsk(wave14, OpenName("Wave14"));
    qsk(wave16, OpenName("Wave16"));
    qsk(result, Abort(wave14));
    qsk(result, Abort(wave16));
    qsk(x, Hello(iee0, buffer));
    printf("\n%$n\n", buffer);
    myTerm.EOI = 1;
    myTerm.nChar = 0;
    myTerm.termChar[0] = 'r';
    myTerm.termChar[1] = 'n';
    // Arm SRQ detection and set up SRQ function handler
    qsk(x, Arm(iee0, acSRQ));
    qsk(x, OnEventVDM(iee0, srqHandler));
    // Tell the Wave to assert SRQ in 3 seconds
    qsk(xfered, Output(wave16, "t3000x", 6L, 1, 0, &myTerm, 0, &ioStatus));
}
```

```
printf("Completion code: 0x%04x\n", ioStatus);

// Normally, your program would be off doing other work; for
// this example we will just hold here for a short time.
For(result = 0; result < 30000; result++) {
    printf("Result is %06d\r", result);
}
printf("\n\n");
qsk(xfered, Spoll(wave16));
qsk(x, Close(wave14));
qsk(x, Close(wave16));
qsk(x, Close(ieee0));
}
```

6.38 OpenName

Syntax	DevHandleT WINAPI OpenName(LPSTR name); name is the name of an interface or external device.
Returns	-1 if error otherwise, the device handle associated with the given name
Mode	Any
Bus State	None
Examples	dmm = OpenName("DMM"); Opens the external device DMM dmm = OpenName("IEEE:DMM"); Specifies the interface to which the external device is connected
See Also	MakeDevice, Close

The `OpenName` command opens the specified interface or external device and returns a device handle for use in accessing that device.

6.39 OutputX

Syntax

```
LONG WINAPI OutputX(DevHandleT devHandle,  
LPBYTE data, DWORD count, BOOL last, BOOL  
forceAddr, TermT *terminator, BOOL async, LPDWORD  
compStat);
```

`devHandle` refers to either an interface or an external device. If `devHandle` refers to an external device, the `OutputX` command acts on the hardware interface to which the external device is attached.

`data` is a string of bytes to send.

`count` is the number of bytes to send.

`last` is a flag that forces the device output terminator to be sent with the data.

`forceAddr` is used to specify whether the addressing control bytes are to be issued for each `OutputX` command.

`terminator` is a pointer to a terminator structure that is used to set up the input terminators. If `terminator` is set to 0, the default terminator is used.

`async` is a flag that allows asynchronous data transfer. Note that this asynchronous flag is ignored in `Driver488/WNT`.

`compStat` is a pointer to an integer containing completion-status information.

Returns	-1 if error otherwise, the number of characters transferred
Mode	CA
Bus States	With interface handle: REN (if SC and AutoRemote), *ATN, ATN With external device handle: REN (if SC and AutoRemote), ATN(MTA, UNL, LAG, *ATN, ATN
Example	term.EOI = TRUE; term.nChar = 1; term.EightBits = TRUE; term.termChar[0] = '\r'; data = "U0X"; count = strlen(data); bytecnt=Output(timer,data,count,1,0,&term,0,&stat);
See Also	EnterX, Term, TimeOut, Buffered

NOTE

The asynchronous flag `async` is ignored in `Driver488/WNT`.

The OutputX command sends data to an interface or external device. The Remote Enable (REN) line is first asserted if Driver488 is the System Controller and AutoRemote is enabled. Then, if a device address (with optional secondary address) is specified, Driver488 is addressed to Talk and the specified device is addressed to Listen. If no address is specified, then Driver488 must already be configured to send data, either as a result of a preceding OutputX command, or as the result of a Send command. Terminators are automatically appended to the output data as specified.

The forceAddr flag is used to specify whether the addressing control bytes are to be issued for each OutputX command. If the device handle refers to an interface, forceAddr has no effect and command bytes are not sent. If the device handle refers to an external device and forceAddr is TRUE, Driver488 addresses the interface to Talk and the external device to Listen. If forceAddr is FALSE, Driver488 compares the current device with the most recently addressed device on that interface. If the addressing information is the same, no command bytes are sent. If they are different, OutputX acts as if the forceAddr flag were TRUE and sends the addressing information.

The terminator is a pointer to a terminator structure that is used to set up the input terminators. This pointer may be a null pointer, requesting use of the default terminators for the device, or it may point to a terminator structure requesting no terminators. The async is a flag that allows asynchronous data transfer. If this flag is TRUE, the OutputX command returns to the caller as soon as the data transfer is underway. FALSE indicates that the OutputX command should not return until the transfer is complete. The compStat is a pointer to an integer containing completion status information. A null pointer indicates that completion status is not requested. In the case of an asynchronous transfer, this pointer must remain valid until the transfer is complete.

Additional Output Functions

Driver488 provides additional Output functions that are short-form versions of the OutputX function. The following Output functions are already defined in your header file.

OUTPUT

Syntax LONG WINAPI Output(DevHandleT devHandle,LPBYTE data);

Remarks Output is equivalent to the following call to OutputX:

```
OutputX(devHandle,data,strlen(data),1,1,0L,0,0L);
```

The Output function passes the device handle and a pointer to the data buffer to the OutputX function. It determines the size of the data buffer that you provided, and passes that value as the count parameter. It specifies that the forceAddr flag is set TRUE, which causes Driver488 to address the device if an external device is specified. The default terminators are chosen by specifying a 0 pointer as the terminator parameter. Synchronous transmission is specified by

sending 0 for the async parameter, and the completion status value is ignored by sending a 0 for the compStat pointer.

OUTPUTN

Syntax LONG WINAPI OutputN(DevHandleT devHandle,LPBYTE data,DWORD count);

Remarks OutputN is equivalent to the following call to OutputX:
OutputX(devHandle,data,count,0,1,0L,0,0L);

The OutputN function passes the device handle and a pointer to the data buffer to the OutputX function. It specifies that the forceAddr flag is set TRUE, which causes Driver488 to address the device if an external device is specified. The default terminators are chosen by specifying a 0 pointer as the terminator parameter. Synchronous transmission is specified by sending 0 for the async parameter, and the completion-status value is ignored by sending a 0 for the compStat pointer.

OUTPUTMORE

Syntax LONG WINAPI OutputMore(DevHandleT devHandle, LPBYTE data);

Remarks OutputMore is equivalent to the following call to OutputX:
OutputX(devHandle,data,strlen(data),1,0,0L,0,0L);

The OutputMore function passes the device handle and a pointer to the data buffer to the OutputX function. It determines the size of the data buffer that you provided, and passes that value as the count parameter. It specifies that the forceAddr flag is set FALSE, so Driver488 does not re-address the device if it is the same device as that previously used. The default terminators are chosen by specifying a 0 pointer as the terminator parameter. Synchronous transmission is specified by sending 0 for the async parameter, and the completion-status value is ignored by sending a 0 pointer for the compStat pointer.

OUTPUTNMORE

Syntax LONG WINAPI OutputNMore (DevHandleT devHandle, LPBYTE data, DWORD count);

Remarks OutputNMore is equivalent to the following call to OutputX:
OutputX(devHandle,data,0,0,0L,0,0L);

The OutputNMore function passes the device handle and a pointer to the data buffer to the OutputX function. It specifies that the forceAddr flag is set FALSE, so Driver488 does not re-address the device if it is the same device as that previously used. The default terminators are chosen by specifying a 0 pointer as the terminator parameter. Synchronous transmission is specified by sending 0 for the async parameter, and the completion-status value is ignored by sending a 0 pointer for the compStat pointer.

6.40 PassControl

Syntax INT WINAPI PassControl(DevHandleT devHandle);

devHandle refers to an external device to which control is passed.

Returns -1 if error

Mode CA

Bus States ATN(UNL, MLA, TAG, UNL, TCT, *ATN

Example errorcode = PassControl(scope);

See Also Abort, Reset, SendCmd

The PassControl command allows Driver488 to give control to another controller on the bus. After passing control, Driver488 enters the Peripheral mode. If Driver488 was the System Controller, then it remains the System Controller, but it is no longer the Active Controller. The Controller now has command of the bus

until it passes control to another device or back to Driver488. The System Controller can regain control of the bus at any time by issuing an Abort command.

6.41 PPoll

Syntax	INT WINAPI PPoll(DevHandleT devHandle); devHandle refers to either an interface or an external device. If devHandle refers to an external device, then the PPoll command acts on the hardware interface to which the external device is attached.
Returns	-1 if error otherwise, a number in the range 0 to 255
Mode	CA
Bus States	ATN(EOI, *EOI)
Example	errorcode = PPoll(iieee);
See Also	PPollConfig, PPollUnconfig, PPollDisable, SPoll

The PPoll (Parallel Poll) command is used to request status information from many bus devices simultaneously. If a device requires service then it responds to a Parallel Poll by asserting one of the eight IEEE 488 bus data lines (DIO1 through DIO8, with DIO1 being the least significant). In this manner, up to eight devices may simultaneously be polled by the controller. More than one device can share any particular DIO line. In this case, it is necessary to perform further Serial Polling (SPoll) to determine which device actually requires service.

Parallel Polling is often used upon detection of a Service Request (SRQ), though it may also be performed periodically by the controller. In either case, PPoll responds with a number from 0 to 255 corresponding to the eight binary DIO lines. Not every device supports Parallel Polling. Refer to the manufacturer's documentation for each bus device to determine if PPoll capabilities are supported.

6.42 PPollConfig

Syntax INT WINAPI PPollConfig(DevHandleT devHandle, BYTE pprresponse);

devHandle refers to either an interface or an external device to configure for the Parallel Poll.

pprresponse is the decimal equivalent of the four binary bits S, P2, P1, and P0 where S is the Sense bit, and P2, P1, and P0 assign the DIO bus data line used for the response.

Returns -1 if error

Mode CA

Bus States ATN(UNL, MTA, LAG, PPC

Example errorcode = PPollConfig (dmm,0x0D);

Configure device DMM to assert DIO6 when it desires service (ist = 1) and it is Parallel Polled (0x0D = &H0D = 1101 binary; S=1, P2=1, P1=0, P0=1; 101 binary = 5 decimal = DIO6).

See Also PPoll, PPollUnconfig, PPollDisable

The PPollConfig command configures the Parallel Poll response of a specified bus device. Not all devices support Parallel Polling and, among those that do, not all support the software control of their Parallel Poll response. Some devices are configured by internal switches.

The Parallel Poll response is set by a four-bit binary number response: S, P2, P1, and P0. The most significant bit of response is the Sense (S) bit. The Sense bit is used to determine when the device will assert its Parallel Poll response. Each bus device has an internal individual status (ist). The Parallel Poll response is asserted when this ist equals the Sense bit value S. The ist is normally a logic 1 when the device requires attention, so the S bit should normally also be a logic 1. If the S bit

is 0, then the device asserts its Parallel Poll response when its ist is a logic 0. That is, it does not require attention. However, the meaning of ist can vary between devices, so refer to your IEEE 488 bus device documentation. The remaining 3 bits of response: P2, P1, and P0, specify which DIO bus data line is asserted by the device in response to a Parallel Poll. These bits form a binary number with a decimal value from 0 through 7, specifying data lines DIO1 through DIO8, respectively.

6.43 PPolIDisable

Syntax	INT WINAPI PPolIDisable(DevHandleT devHandle);
	devHandle is either an interface or an external device that is to have its Parallel Poll response disabled.
Returns	-1 if error
Mode	CA
Bus States	ATN{UNL, MTA, LAG, PPC, PPD
Example	errorcode = PPolIDisable(dmm); Disables Parallel Poll of device DMM.
See Also	PPoll, PPolConfig, PPolUnconfig

The PPolIDisable command disables the Parallel Poll response of a selected bus device.

6.44 PPollDisableList

Syntax	INT WINAPI PPollDisableList(DevHandlePT dhList); dhList is a pointer to a list of external devices that are to have their Parallel Poll response disabled.
Returns	-1 if error
Mode	CA
Bus States	ATN(UNL, MTA, LAG, PPC, PPD
Example	<pre>deviceList[0] = wave; deviceList[1] = timer; deviceList[2] = dmm; deviceList[3] = NODEVICE; errorcode = PPollDisableList(deviceList);</pre>
See Also	PPoll, PPollConfig, PPollUnconfig

The PPollDisableList command disables the Parallel Poll response of selected bus devices.

6.45 PPollUnconfig

Syntax	INT WINAPI PPollUnconfig(DevHandleT devHandle); devHandle refers to a hardware interface. If devHandle refers to an external device, then the PPollUnconfig command acts on the hardware interface to which the external device is attached.
Returns	-1 if error
Mode	CA
Bus States	ATN(PPU
Example	<pre>errorcode = PPollUnconfig(ieee);</pre>

See Also PPoll, PPollConfig, PPollDisable

The PPollUnconfig command disables the Parallel Poll response of all bus devices.

6.46 Remote

Syntax INT WINAPI Remote(DevHandleT devHandle);
 devHandle refers to either an interface or an external device. If devHandle refers to an interface, then the Remote Enable (REN) line is asserted. If devHandle refers to an external device, then that device is addressed to Listen and placed into the Remote state.

Returns -1 if error

Mode SC

Bus States With interface: REN

With external device: REN, ATN(UNL, MTA, LAG

Examples errorcode = Remote(ieee);

Asserts the REN bus line

errorcode = Remote(scope);

Asserts the REN bus line and addresses the scope device specified to Listen, to place it in the Remote state.

See Also Local, LocalList, RemoteList

The Remote command asserts the Remote Enable (REN) bus management line. If an external device is specified, then Remote will also address that device to Listen, placing it in the Remote state.

6.47 RemoteList

Syntax INT WINAPI RemoteList(DevHandlePT dhList);

dhList is a pointer to a list of devices.

Returns -1 if error

Mode SC(CA)

Bus States REN, ATN(UNL, MTA, LAG)

Example

```
deviceList[0] = wave;
deviceList[1] = timer;
deviceList[2] = dmm;
deviceList[3] = NODEVICE;
errorcode = RemoteList(deviceList);
```

Asserts the REN bus line and addresses a list of specified devices to Listen, to place these specified devices in the Remote state.

See Also Remote, Local, LocalList

The RemoteList command asserts the Remote Enable (REN) bus management line. If external devices are specified, then RemoteList will also address those devices to Listen, placing them in the Remote state.

6.48 RemoveDevice

Syntax INT WINAPI RemoveDevice(DevHandleT devHandle);
devHandle specifies an interface or an external device to remove.

Returns -1 if error

Mode Any

Bus States	None
Example	<code>errorcode = RemoveDevice(dmm);</code>
See Also	MakeDevice, KeepDevice

The RemoveDevice command removes the specific temporary or permanent Driver488 device that was created with either the MakeDevice command or the startup configuration. This command also removes a device that was made permanent through a KeepDevice command.

6.49 Request

Syntax	<p>INT WINAPI Request(DevHandleT devHandle, BYTE spstatus);</p> <p>devHandle refers to either an interface or an external device. If devHandle refers to an external device, the Request command acts on the hardware interface to which the external device is attached.</p> <p>spstatus is the Service Request status in the range 0 to 255.</p>
Returns	-1 if error
Mode	*CA
Bus States	<p>SRQ (if rsv is set)</p> <p>*SRQ (if rsv is not set)</p>
Examples	<p><code>errorcode = Request(iieee,0);</code></p> <p>Clears SRQ and Serial Poll Response.</p> <p><code>errorcode = Request(iieee,64+2+4);</code></p>

Generates an SRQ (decimal 64) with DIO2 (decimal 2) and DIO3 (decimal 4) set in the Serial Poll Response.

See Also Status, ControlLine

In Peripheral mode, Driver488 is able to request service from the Active Controller by asserting the Service Request (SRQ) bus signal. The Request command sets or clears the Serial Poll status (including Service Request) of Driver488. Request takes a numeric argument in the decimal range 0 to 255 (hex range &H0 to &HFF) that is used to set the Serial Poll status. When Driver488 is Serial Polled by the Controller, it returns this byte on the DIO data lines.

The data lines are numbered DIO8 through DIO1. DIO8 is the most significant line and corresponds to a decimal value of 128 (hex &H80). DIO7 is the next most significant line and corresponds to a decimal value of 64 (hex &H40). DIO7 has a special meaning: It is the Request for Service (rsv) bit. If rsv is set, then Driver488 asserts the Service Request (SRQ) bus signal. If DIO7 is clear (a logic 0), then Driver488 does not assert SRQ. When Driver488 is Serial Polled, all eight bits of the Serial Poll status are returned to the Controller. The rsv bit is cleared when Driver488 is Serial Polled by the Controller. This causes Driver488 to stop asserting SRQ.

6.50 Reset

Syntax INT WINAPI Reset(DevHandleT devHandle);
devHandle refers to either an interface or an external device. If devHandle refers to an external device, the Reset command acts on the hardware interface to which the external device is attached.

Returns -1 if error

Mode Any

Bus States None

Example errorcode=Reset(ieee);

See Also Abort, Term, TimeOut

The Reset command provides a warm start of the interface. It is equivalent to issuing the following command process, including clearing all error conditions:

1. Stop.
2. Disarm.
3. Reset hardware (resets to Peripheral if not System Controller).
4. Abort (if System Controller).
5. Error ON.
6. Local.
7. Request 0 (if Peripheral).
8. Clear Change, Trigger, and Clear status.
9. Reset I/O adapter settings to installed values (BusAddress, TimeOut, IntLevel and DmaChannel).

6.51 Resume

Driver488/W95 only

Syntax

INT WINAPI Resume(DevHandleT devHandle, BOOL monitor);

devHandle refers to either an interface or an external device. If devHandle refers to an external device, then the Resume command acts on the hardware interface to which the external device is attached.

monitor is a flag that when it is ON, Driver488 monitors the data.

Returns

-1 if error

Mode

CA

Bus States

*ATN

Examples `errorcode = Resume(iieee,OFF);`
 Do not go into monitoring mode.
 `errorcode = Resume(iieee,ON);`
 `errorcode = Finish(iieee);`
 Go into monitoring mode.

See Also Finish

The Resume command unasserts the Attention (ATN) bus signal. Attention is normally kept asserted by Driver488, but it must be unasserted to allow transfers to take place between two peripheral devices. In this case, Driver488 sends the appropriate Talk and Listen addresses, and then must unassert Attention with the Resume command.

If monitor is specified, Driver488 monitors the handshaking process but does not participate in it. Driver488 takes control synchronously when the last terminator or EOI is encountered. At that point, the transfer of data stops. The Finish command must be called to assert Attention and release any pending holdoffs to be ready for the next action.

6.52 SendCmd

Syntax `INT WINAPI SendCmd(DevHandleT devHandle, LPBYTE`
 `commands, DWORD len);`

`devHandle` refers to an interface handle.

`commands` points to a string of command bytes to be sent.

`len` is the length of the command string.

Response None

Mode CA

Bus States User-defined

Example `char command[] = "U?0";`

```
errorcode = SendCmd(iieee, &command, size of
command);
```

See Also SendData, SendEoi

The SendCmd command sends a specified string of bytes with Attention (ATN) asserted, causing the data to be interpreted as IEEE 488 command bytes.

6.53 SendData

Syntax INT WINAPI SendData(DevHandleT devHandle, LPBYTE data, DWORD len);

devHandle refers to an interface handle.

data points to a string of data bytes to be sent.

len is the length of the data string.

Response None

Mode Any

Bus States User-defined

Example char data[] = "W0X";
errorcode = SendData(iieee, data, strlen (data));

See Also SendCmd, SendEoi

The SendData command provides byte-by-byte control of data transfers and gives greater flexibility than the other commands. This command can specify exactly which operations Driver488 executes.

6.54 SendEoi

Syntax	<code>INT WINAPI SendEoi(DevHandleT devHandle, LPBYTE data, DWORD len);</code> devHandle refers to an interface handle. data points to a string of data bytes to be sent. len is the length of the data string.
Response	None
Mode	Any
Bus States	User-defined
Example	<code>char data[] = "W0X";</code> <code>errorcode = SendEoi(jeee, data, strlen (data));</code>
See Also	SendCmd, SendData

The SendEoi command provides byte-by-byte control of data transfers and gives greater flexibility than the other commands. This command can specify exactly which operations Driver488 executes. Driver488 asserts EOI during the transfer of the final byte.

6.55 SPoll

Syntax	<code>INT WINAPI SPoll(DevHandleT devHandle);</code> devHandle refers to either an interface or a specific external device.
Returns	-1 if error; otherwise, 0 or 64 (hardware interface) in the range 0 to 255 (external device)

Mode	Any
Bus States	ATN(UNL, MLA, UNT, TAG, SPE, *ATN, ATN(SPD, UNT
Examples	errorcode = SPoll(ieee); Returns the internal SRQ status errorcode = SPoll(dmm); Returns the Serial Poll response of the specified device
See Also	SPollList, PPoll

In Active Controller mode, the SPoll (Serial Poll) command performs a Serial Poll of the bus device specified and responds with a number from 0 to 255 representing the decimal equivalent of the eight-bit device response. If rsv (DIO7, decimal value 64) is set, then that device is signaling that it requires service. The meanings of the other bits are device-specific.

Serial Polls are normally performed in response to assertion of the Service Request (SRQ) bus signal by some bus device. In Active Controller mode, with the interface device specified, the SPoll command returns the internal SRQ status. If the internal SRQ status is set, it usually indicates that the SRQ line is asserted. Driver488 then returns a 64. If it is not set, indicating that SRQ is not asserted, then Driver488 returns a 0. With an external device specified, SPoll returns the Serial Poll status of the specified external device.

In Peripheral mode, the SPoll command is issued only to the interface, and returns the Serial Poll status. If rsv (DIO7, decimal value 64) is set, then Driver488 has not been Serial Polled since the issuing last Request command. The rsv is reset whenever Driver488 is Serial Polled by the Controller.

6.56 SPollList

Syntax	INT WINAPI SPollList(DevHandlePT dhList, LPBYTE result, BYTE flag);
	dhList is a pointer to a list of external devices.
	result is an array that is filled in with the Serial Poll results of the corresponding external devices.
	flag refers to either ALL, WHILE_SRQ, or UNTIL_RSV.

Returns -1 if error

Mode *CA

98

ATN(UNL, MLA, TAG, SPE, *ATN, ATN(SPD, UNT

Example

```
deviceList[0] = wave;
deviceList[1] = timer;
deviceList[2] = dmm;
deviceList[3] = NODEVICE;
result = SPollList(deviceList, resultList, ALL);
Returns the Serial Poll response for a list of device
handles.
```

See Also SPoll, PPoll

In Active Controller mode, the SPollList (Serial Poll) command performs a Serial Poll of the bus devices specified and responds with a number from 0 to 255 (representing the decimal equivalent of the eight-bit device response) for each device on the list. If rsv (DIO7, decimal value 64) is set, then that device is signaling that it requires service. The meanings of the other bits are device-specific.

Serial Polls are normally performed in response to assertion of the Service Request (SRQ) bus signal by some bus device. In Active Controller mode with the interface device specified, the SPollList command returns the internal SRQ status for each device. If the internal SRQ status is set, it usually indicates that the SRQ line is asserted. Driver488 then returns a 64. If it is not set, indicating that SRQ is not asserted, then Driver488 returns a 0. With an external device specified, SPollList returns the Serial Poll status of the specified external device.

In Peripheral mode, the SPollList command is issued only to the interface and returns the Serial Poll status. If rsv (DIO7, decimal value 64) is set, then Driver488 has not been Serial Polled since the last Request command was issued. The rsv is reset whenever Driver488 is Serial Polled by the Controller.

The untilflag refers to either ALL, WHILE_SRQ, or UNTIL_RSV. If ALL is chosen, all the devices are Serial Polled and their results placed into the result array. If untilflag is WHILE_SRQ, Driver488 Serial Polls the devices until the SRQ bus signal becomes unasserted, and the results are put into the result array. If untilflag is UNTIL_RSV, Driver488 Serial Polls the devices until the first device whose rsv bit is set, is found and the results are put into the result array.

6.57 Status

Syntax	<p>INT WINAPI Status(DevHandleT devHandle, IEEEStatusPT result);</p> <p>devHandle refers to either an IEEE 488 interface or an external device. If devHandle refers to an external device, Status acts on the hardware interface to which the external device is attached.</p> <p>result is a pointer to a Status structure.</p>
Returns	-1 if error
Mode	Any
Bus States	None
Example	<pre>result = Status(ieee,&StatusResult); if (statusResult.transfer == TRUE) { printf("We have a transfer in progress\n"); } else { printf("There is no transfer in progress\n"); }</pre>
See Also	GetError, SPoll

PERSONAL 488 ISA CARD, PERSONAL 488 PCI CARD

The Status command returns various items detailing the current state of Driver488. They are returned in a data structure, based on the following table:

Status Item	Flag	Values and Description
Controller Active	.CA	TRUE: Active Controller; FALSE: Not CA.
System Controller	.SC	TRUE: System Controller; FALSE: Not SC.
Primary Bus Address	.Primaddr	0 to 30: Two-digit decimal number.
Secondary Bus Address	.Secaddr	0 to 31: Two-digit decimal number, or -1 if no address.
Address Change	.addrChange	TRUE: Address change has occurred; FALSE: Not so.
Talker	.talker	TRUE: Talker; FALSE: Not Talker.
Listener	.listener	TRUE: Listener; FALSE: Not Listener.
ByteIn	.bytein	TRUE: Byte in, ready to read; FALSE: Not so.
ByteOut	.byteout	TRUE: Byte out, ready to output; FALSE: Not so.
Service Request	.SRQ	TRUE: SRQ is asserted; FALSE: SRQ is not asserted.
Triggered	.triggered	TRUE: Trigger command received; FALSE: Not so.
Cleared	.cleared	TRUE: Clear command received; FALSE: Not so.
Transfer in Progress	.transfer	TRUE: Transfer in progress; FALSE: Not so.

These Status items are more fully described in the following paragraphs:

- The Controller Active flag (.CA) is true if Driver488 is the Active Controller. If Driver488 is not the System Controller, then it is initially a Peripheral and it becomes a controller when Driver488 receives control from the Active Controller.
- The System Controller flag (.SC) is true if Driver488 is the System Controller. The System Controller mode may be configured during installation or by using the SysController command.
- The Primary Bus Address (.Primaddr) is the IEEE 488 bus device primary address assigned to Driver488 or the specified device. This will be an integer from 0 to 30. The Secondary Bus Address (.Secaddr) is the IEEE 488 bus device secondary address assigned to the specified device. This will be either -1 to indicate no secondary address, or an integer from 0 to 31. Note that the Personal 488 Card can never have a secondary address.
- The Address Change indicator (.addrChange) is set whenever Driver488 become a Talker, Listener, or the Active Controller, or when it becomes no longer a Talker, Listener, or the Active Controller. It is reset when Status is read. The Talker (.talker) and Listener (.listener) flags reflect the current Talker/Listener state of Driver488. As a Peripheral, Driver488 can check this status to see if it has been addressed to Talk or addressed to Listen by the Active Controller. In this way, the desired direction of data transfer can be determined.
- The ByteIn (.byteIn) indicator is set when the I/O adapter has received a byte that can be read by an Enter command. The ByteOut (.byteOut) indicator is set when the I/O adapter is ready to output data. The Service Request field (.SRQ), as an active controller, reflects the IEEE 488 bus SRQ line signal. As a peripheral, this status reflects the rsv bit that can be set by the Request command and is cleared when the Driver488 is Serial Polled. For more details, refer to the SPoll command in this chapter.
- The Triggered (.triggered) and Cleared (.cleared) indicators are set when, as a Peripheral, Driver488 is triggered or cleared. These two indicators are cleared when Status is read. The Triggered and Cleared indicators are not updated while asynchronous transfers are in progress. The Transfer in Progress (.transfer) indicator signifies an asynchronous transfer in progress.

6.58 Stop

Driver488/W95 only

Syntax	INT WINAPI Stop(DevHandleT devHandle); devHandle refers to either an interface or an external device. If devHandle refers to an external device, the Stop command acts on the hardware interface to which the external device is attached.
Returns	-1 if error
Mode	Any
Bus States	ATN (Controller) None (Peripheral)
Example	errorcode = Stop(ieee);
See Also	EnterX, OutputX, Buffered

The Stop command halts any asynchronous transfer that may be in progress. If the transfer has completed already, then Stop has no effect. The actual number of characters transferred is available from the Buffered command.

6.59 Talk

Syntax	INT WINAPI Talk(DevHandleT devHandle, BYTE pri, BYTE sec); devHandle refers to either an interface or an external device. If devHandle refers to an external device, the Talk command acts on the associated interface. pri and sec specify the primary and secondary addresses of the device which is to be addressed to Talk.
---------------	---

Returns	-1 if error
Mode	CA
Bus States	ATN, TAG
Example	errorcode = Talk (ieee, 12, -1);
See Also	Listen, SendCmd

The Talk command addresses an external device to Talk.

6.60 Term

Syntax	<pre>INT WINAPI Term(DevHandleT devHandle, TermT *terminator, DWORD TermType);</pre> <p>devHandle refers to either an interface or an external device.</p> <p>terminator is a pointer to the terminator structure.</p> <p>TermType can be either TERMIN, TERMOUT, or TERMIN+TERMOUT, specifying whether input, output, or both are being set.</p>
Returns	-1 if error
Mode	Any
Bus States	None
Example	<pre>term.EOI = TRUE; term.nChar = 1; term.EightBits = TRUE; term.termChar[0] = 13; errorcode = Term(ieee,&term,TERMIN);</pre>
See Also	TermQuery, EnterX, OutputX, Status

The Term command sets the end-of-line (EOL) terminators for input from, and output to, I/O adapter devices. These terminators are sent at the end of output data and expected at the end of input data, in the manner of CR LF as used with printer data.

During output, Term appends the bus output terminator to the data before sending it to the I/O adapter device. Conversely, when Driver488 receives the bus input terminator, it recognizes the end of a transfer and returns the data to the calling application. The terminators never appear in the data transferred to or from the calling application. The default terminators for both input and output are set by the startup configuration and are normally CR LF EOI, which is appropriate for most bus devices.

End-Or-Identify (EOI) has a different meaning when it is specified for input than when it is specified for output. During input, EOI specifies that input is terminated on detection of the EOI bus signal, regardless of which characters have been received. During output, EOI specifies that the EOI bus signal is to be asserted during the last byte transferred.

6.61 TermQuery

Syntax INT TermQuery(DevHandleT devHandle, TermT *terminator, INT TermType);

devHandle refers to either an interface or an external device.

terminator is a pointer to the terminator structure.

TermType can be either TERMIN, TERMOUT, or TERMIN+TERMOUT, specifying whether input, output, or both are being set.

Returns -1 if error

Mode Any

Bus States None

See Also Term, EnterX, OutputX, Status

The TermQuery function queries the terminators setting. Terminators are defined by the TermT structure.

6.62 TimeOut

Syntax	INT WINAPI TimeOut(DevHandleT devHandle, DWORD millisec);
	devHandle refers to either an IEEE 488 interface or an external device.
	millisec is a numeric value given in milliseconds.
Returns	-1 if error
Mode	Any
Bus States	None
Example	errorcode = TimeOut(ieee,100); Sets the timeout value to 100 msec.
See Also	TimeOutQuery, Reset

The TimeOut command sets the number of milliseconds that Driver488 waits for a transfer before declaring a timeout error. Driver488 checks for timeout errors on every byte it transfers, except in the case of asynchronous transfers. While the first byte of an asynchronous transfer is checked for timeout, subsequent bytes are not. Your program must check for timely completion of an asynchronous transfer.

Timeout checking may be suppressed by specifying a timeout value of zero seconds, which specifies an infinite timeout. The default time out is specified in the startup configuration, normally 10 seconds. The timeout interval may be specified to the nearest 0.001 seconds (1 millisecond). However, due to the limitations of the computer, the actual interval is always a multiple of 55 milliseconds and there is an uncertainty of 55 msec in the actual interval. Timeout intervals from 1 to 110 milliseconds are rounded to 110 milliseconds. Larger intervals are rounded to the nearest multiple of 55 msec (e.g. 165, 220, 275 msec, etc.).

6.63 TimeOutQuery

Syntax

```
INT WINAPI TimeOutQuery(DevHandleT  
devHandle,DWORD millisec);
```

devHandle refers to either an IEEE 488 interface or an external device.

millisec is a numeric value given in milliseconds.

Returns

-1 if error

Mode

Any

Bus States

None

See Also

TimeOut, Reset

The TimeOutQuery function queries the time out setting, given in milliseconds.

6.64 Trigger

Syntax

```
INT WINAPI Trigger(DevHandleT devHandle);
```

devHandle refers to either an IEEE 488 interface or an external device.

Returns

-1 if error

Mode

CA

Bus States

With interface handle: ATN(GET

With external device handle: ATN(UNL, MTA, LAG, GET

Examples

```
errorcode = Trigger(ieee);
```

Issues a Group Execute Trigger (GET) bus command to those devices that are already in the Listen state as the result of a previous Output or Send command

errorcode = Trigger(dmm);
 Issues a Group Execute Trigger (GET) bus command to the device specified

See Also TriggerList, Status, SendCmd

The Trigger command issues a Group Execute Trigger (GET) bus command to the specified device. If no interface devices are specified, then the GET only affects those devices that are already in the Listen state as a result of a previous Output or Send command.

6.65 TriggerList

Syntax INT WINAPI TriggerList(DevHandlePT dhList);

dhList is a pointer to a list of external devices.

Returns -1 if error

Mode CA

Bus States ATN(UNL, MTA, LAG, GET)

Example
 deviceList[0] = wave;
 deviceList[1] = timer;
 deviceList[2] = dmm;
 deviceList[3] = NODEVICE;
 errorcode = TriggerList(deviceList);
 Issues a Group Execute Trigger (GET) bus command to a list of specified devices.

See Also Trigger, SendCmd, Status

The TriggerList command issues a Group Execute Trigger (GET) bus command to the specified devices. If no interface devices are specified, then the GET affects those devices that are already in the Listen state as a result of a previous Output or Send command.

6.66 UnListen

Syntax	INT WINAPI UnListen (DevHandleT devHandle); devHandle refers to either an interface or an external device. If devHandle refers to an external device, the UnListen command acts on the associated interface.
Returns	-1 if error
Mode	CA
Bus States	ATN, UNL
Example	errorcode = UnListen (ieee);
See Also	Listen, UnTalk, SendCmd, Status

The UnListen command unaddresses an external device that was addressed to Listen.

6.67 UnTalk

Syntax	INT WINAPI UnTalk (DevHandleT devHandle); devHandle refers to either an interface or an external device. If devHandle refers to an external device, the UnTalk command acts on the associated interface.
Returns	-1 if error
Mode	CA
Bus States	ATN, UNT
Example	errorcode = UnTalk (ieee);
See Also	Talk, UnListen, SendCmd, Status

The UnTalk command unaddresses an external device that was addressed to Talk.

6.68 Wait

Driver488/W95 only

Syntax INT WINAPI Wait(DevHandleT devHandle);

devHandle refers to either an interface or an external device. If devHandle is an external device, the Wait command acts on the hardware interface to which the external device is attached.

Returns -1 if error

Mode Any

Bus States Determined by previous Enter or Output command

Example errorcode = Wait(ieee);

See Also EnterX, OutputX, Buffered, Status

The Wait command causes Driver488 to wait until any asynchronous transfer has completed before returning to your program. It can be used to guarantee that the data has actually been received before beginning to process it, or that it has been sent before overwriting the buffer. It is especially useful with the Enter command, when a terminator has been specified. In that case, the amount that is actually received is unknown, and so your program must check with Driver488 to determine when the transfer is done. Timeout checking, if enabled, is performed while Waiting.

7. Troubleshooting

7.1 Radio Interference Problems

Personal488 hardware systems generate, use and can radiate radio-frequency energy, and, if not installed and not used correctly, may cause harmful interference to radio communications. If this equipment does cause harmful interference to radio or television reception, which you can determine by turning the equipment off and on, we encourage you to try to correct the interference by one or more of the following measures:

- **Antenna Adjustment:** Reorient or relocate the receiving antenna.
- **Spatial Separation:** Increase the separation between the equipment and receiver.
- **Circuit Separation:** Connect the equipment to an outlet on a circuit different from that to which the receiver is connected.

Otherwise, consult the dealer or an experienced radio/television technician for help.

7.2 IEEE 488 Bus Errors

- **Connections:** Check to make sure that all of the IEEE 488 bus cables are securely fastened to their respective terminals, and that the IEEE 488 Standard has been met.
- **Primary Addresses:** Check to make sure that each of the IEEE 488 bus primary addresses has a unique value between 0 and 30. No two interface boards or external devices should have the same primary address within any single IEEE 488 bus system. The default IEEE 488 bus primary address is 21, but you can change this if it conflicts with some other device.
- **Timeouts:** Check to make sure that the timeout period is preferred over setting IEEE 488 bus terminators. If the timeout period elapses while waiting to transfer data or while waiting for unspecified terminators, a time out error occurs.
- **Bus Terminators:** Check to make sure that the IEEE 488 bus terminators sent by the device and the IEEE 488 bus terminators expected by the software driver match up. Typically, these terminators are carriage return (CR) and line feed (LF), followed by an End-Or-Identify (EOI).

For more information on the configuration of Driver488 software settings for IEEE 488 interfaces and external devices, see **Chapter 5**.

NOTE

If you make any changes to the configuration parameters—such as the Primary Address, Timeout, and Bus Terminator—you must restart the device driver for the changes to take effect. For Driver488/W95, close all applications using the interface board and restart your programs. For Driver488/WNT, you can use the following commands to restart your device driver: `net stop drvvr488` and `net start drvvr488`.

7.3 Hardware-Software Conflicts

NOTE

The following list applies only to the ISA card.

- I/O Base Address: Check to make sure that the I/O base address you selected through the configuration utility of Driver488 matches the interrupt setting configured through the appropriate DIP switch on your IEEE 488 interface.
- Interrupt Setting: Check to make sure that the interrupt setting selected through the configuration utility of Driver488, matches the interrupt setting configured through the appropriate jumper(s) and DIP switch(es) on your IEEE 488 interface.
- Direct Memory Access (DMA) Setting: Check to make sure that the DMA setting selected through the configuration utility of Driver488 matches the DMA setting configured through the appropriate jumper(s) on your IEEE 488 interface.

7.4 Checking Hardware and Software Settings

Checking the Interface Board Settings

NOTE

The following applies only to the ISA card.

Remove the interface board from the computer, and refer to **Chapter 4**.

Checking the Driver488/W95 Software Settings

1. Open the Control Panel window from the Start > Settings menu, click on the System icon, and select the Device Manager tab. Under the line “Ports (COM

& LPT),” look for a list of used ports. For each port, highlight the port and click on the Properties button.

2. Properties already being used in the system are displayed under the Resources tab. Values NOT listed are available.

For more information on the configuration of Driver488 software settings for IEEE 488 interfaces and external devices, see **Chapter 5**.

Checking the Driver488/WNT Software Settings

1. From the console mode, or DOS prompt, execute the program WINMSD.EXE.
2. When the Windows NT Diagnostic dialog box appears, you can check the settings from the available tab displays.

For more information on the configuration of Driver488 software settings for IEEE 488 interfaces and external devices, see **Chapter 5**.

Appendix

A.1 IEEE 488 Bus and Serial Bus

Table A-1. IEEE 488 Bus and Serial Bus Lines.

Bus State	Bus Line	Data Transfer (DIO) Lines							
		8	7	6	5	4	3	2	1
Bus Management Lines									
IFC	Interface Clear								
REN	Remote Enable								
IEEE 488 Interface: Bus Management Lines									
ATN	Attention (\$04)	0	0	0	0	0	1	0	0
EOI	End-Or-Identify (\$80)	1	0	0	0	0	0	0	0
SRQ	Service Request (\$40)	0	1	0	0	0	0	0	0
IEEE 488 Interface: Handshake Lines									
DAV	Data Valid (\$08)	0	0	0	0	1	0	0	0
NDAC	Not Data Accepted (\$10)	0	0	0	1	0	0	0	0
NRFD	Not Ready For Data (\$20)	0	0	1	0	0	0	0	0
Serial Interface: Bus Management Lines									
DTR	Data Terminal Ready (\$02)	0	0	0	0	0	0	1	0
RI	Ring Indicator (\$10)	0	0	0	1	0	0	0	0
RTS	Request To Send (\$01)	0	0	0	0	0	0	0	1
Serial Interface: Handshake Lines									
CTS	Clear To Send (\$04)	0	0	0	0	0	1	0	0
DCD	Data Carrier Detect (\$08)	0	0	0	0	1	0	0	0
DSR	Data Set Ready (\$20)	0	0	1	0	0	0	0	0
Hexadecimal and Decimal Values									
	Hexadecimal Value	\$80	\$40	\$20	\$10	\$08	\$04	\$02	\$01
	Decimal Value	128	064	032	016	008	004	002	001

A.2 IEEE 488 Bus Commands

Table A-2. IEEE 488 Bus Commands.

Bus State	IEEE 488 Bus Command (ATN is asserted "1")	Data Transfer (DIO) Lines							
		8	7	6	5	4	3	2	1
DCL	Device Clear	0	0	0	1	0	1	0	0
GET	Group Execute Trigger (\$08)	0	0	0	0	1	0	0	0
GTL	Go To Local (\$01)	0	0	0	0	0	0	0	1
LAG	Listen Address Group (\$20-3F)	0	0	1	a	d	d	r	n
LLO	Local Lock Out (\$11)	0	0	0	1	0	0	0	1
MLA	My Listen Address	0	0	1	a	d	d	r	n
MTA	My Talk Address	0	1	0	a	d	d	r	n
PPC	Parallel Poll Config	0	1	1	0	s	P2	P1	P0
PPD	Parallel Poll Disable (\$07)	0	0	0	0	0	1	1	1
PPU	Parallel Poll Unconfig (\$60-7F)	0	0	0	1	0	1	0	1
SCG	Second Command Group (\$60-7F)	0	1	1	c	o	m	m	d
SDC	Selected Device Clear (\$04)	0	0	0	0	0	1	0	0
SPD	Serial Poll Disable (\$19)	0	0	0	1	1	0	0	1
SPE	Serial Poll Enable (\$18)	0	0	0	1	1	0	0	0
TAG	Talker Address Group (\$40-5F)	0	1	0	a	d	d	r	n
TCT	Take Control (\$09)	0	0	0	0	1	0	0	1
UNL	Unlisten (\$3F)	0	0	1	1	1	1	1	1
UNT	Untalk (\$5F)	0	1	0	1	1	1	1	1
Hexadecimal and Decimal Values									
	Hexadecimal Value	\$80	\$40	\$20	\$10	\$08	\$04	\$02	\$01
	Decimal Value	128	064	032	016	008	004	002	001

A.3 ASCII Codes

Decimal Values 00 to 31—ACG and UCG Characteristics

ASCII Control Codes (Decimal 00 to 31)

Dec Value	Hex Value (\$)	Character & Abbreviation	Name	Bus Message
Addressed Command Group (ACG)				
00	\$00	None/NUL	Null	None
01	\$01	^A/SOH	Start of Header	Go to Local (GTL)
02	\$02	^B/STX	Start of Text	None
03	\$03	^C/ETX	End of Text	None
04	\$04	^D/EOT	End of Transmission	Selected Device Clear (SDC)
05	\$05	^E/ENQ	Inquiry	None
06	\$06	^F/ACK	Acknowledgement	None
07	\$07	^G/BEL	Bell	Parallel Poll Disable (PPD)
08	\$08	^H/BS	Backspace	Group Execute Trigger (GET)
09	\$09	^I/HT	Horizontal Tab	Take Control (TCT)
10	\$0A	^J/LF	Line Feed	None
11	\$0B	^K/VT	Vertical Tab	None
12	\$0C	^L/FF	Form Feed	None
13	\$0D	^M/CR	Carriage Return	None
14	\$0E	^N/SO	Shift Out	None
15	\$0F	^O/SI	Shift In	None
Universal Command Group (UCG)				
16	\$10	^P/DLE	Data Link Escape	None
17	\$11	^Q/DC1	Device Control 1	Local Lockout (LLO)
18	\$12	^R/DC2	Device Control 2	None
19	\$13	^S/DC3	Device Control 3	None
20	\$14	^T/DC4	Device Control 4	Device Clear (DCL)
21	\$15	^U/NAK	Negative Acknowledgement	Parallel Poll Unconfig (PPU)
22	\$16	^V/SYN	Synchronous Idle	None
23	\$17	^W/ETB	End of Transmission Block	None
24	\$18	^X/CAN	Cancel	Serial Poll Enable (SPE)
25	\$19	^Y/EM	End of Medium	Serial Poll Disable (SPD)
26	\$1A	^Z/SUB	Substitute	None
27	\$1B	^[_/ESC	Escape	None

ASCII Control Codes (Decimal 00 to 31) (continued)

Dec Value	Hex Value (\$)	Character & Abbreviation	Name	Bus Message
<i>Universal Command Group (UCG) (continued)</i>				
28	\$1C	^VFS	File Separator	None
29	\$1D	^JGS	Group Separator	None
30	\$1E	^VRS	Record Separator	None
31	\$1F	^_US	Unit Separator	None

NOTE

1. ASCII control codes are sometimes used to “formalize” a communications session between communication devices.
2. DC1, DC2, DC3, DC4, FS, GS, RS, and US all have user-defined meanings, and may vary in use between sessions or devices.
3. DC4 is often used as a general “stop transmission character.”
4. Codes used to control cursor position may be used to control print devices, and move the print head accordingly. However, not all devices support the full set of positioning codes.

Decimal Values 00 to 31—ACG and UCG Descriptions

Dec	Name	Description
<i>Addressed Command Group (ACG)</i>		
00	Null (NUL)	Space filler character. Used in output timing for some device drivers.
01	Start of Header (SOH)	Marks beginning of message header.
02	Start of Text (STX)	Marks beginning of data block (text).
03	End of Text (ETX)	Marks end of data block (text).
04	End of Transmission (EOT)	Marks end of transmission session.
05	Inquiry (ENQ)	Request for identification or information.
06	Acknowledgement (ACK)	“Yes” answer to questions or “ready for next transmission.” Used in asynchronous protocols for timing.
07	Bell (BEL)	Rings bell of audible alarm on terminal.
08	Backspace (BS)	Moves cursor position back one character.
09	Horizontal Tab (HT)	Moves cursor position to next tab stop on line.
10	Line Feed (LF)	Moves cursor position down one line.
11	Vertical Tab (VT)	Moves cursor position down to next tab line.
12	Form Feed (FF)	Moves cursor position to top of next page.
13	Carriage Return (CR)	Moves cursor to left margin.
14	Shift Out (SO)	Next characters do not follow ASCII definitions.
15	Shift In (SI)	Next characters revert to ASCII meaning.
<i>Universal Command Group (UCG)</i>		
16	Data Link Escape (DLE)	Used to control transmissions using escape sequences.
17	Device Control 1 (DC1)	Not defined.
18	Device Control 2 (DC2)	Usually user-defined.
19	Device Control 3 (DC3)	Not defined. Normally used for OFF controls.
20	Device Control 4 (DC4)	Usually user-defined.
21	Negative Acknowledgement (NAK)	“No” answer to questions or “errors found, re-transmit.” Used in asynchronous protocols for timing.
22	Synchronous Idle (SYN)	Sent by asynchronous devices when idle to insure sync.

Decimal Values 00 to 31—ACG and UCG Descriptions (continued)

Dec	Name	Description
Universal Command Group (UCB) (continued)		
23	End of Transmission Block (ETB)	Marks block boundaries in transmission.
24	Cancel (CAN)	Indicates previous transmission should be disregarded.
25	End of Medium (EM)	Marks end of physical media, as in paper tape.
26	Substitute (SUB)	Used to replace a character known to be wrong.
27	Escape (ESC)	Marks beginning of an Escape control sequence.
28	File Separator (FS)	Marker for major portion of transmission.
29	Group Separator (GS)	Marker for submajor portion of transmission.
30	Record Separator (RS)	Marker for minor portion of transmission.
31	Unit Separator (US)	Marker for most minor portion of transmission.

NOTE

- 1. ASCII control codes are sometimes used to “formalize” a communications session between communication devices.**
- 2. DC1, DC2, DC3, DC4, FS, GS, RS, and US all have user-defined meanings, and may vary in use between sessions or devices.**
- 3. DC4 is often used as a general “stop transmission character.”**
- 4. Codes used to control cursor position may be used to control print devices, and move the print head accordingly. However, not all devices support the full set of positioning codes.**

Decimal Values 32 to 63—LAG

Dec	Hex	Character	Name	Bus Message
<i>Listen Address Group (LAG)</i>				
32	\$20	<space>	Space	Bus address 00
33	\$21	!	Exclamation Point	Bus address 01
34	\$22	"	Quotation Mark	Bus address 02
35	\$23	#	Number Sign	Bus address 03
36	\$24	\$	Dollar Sign	Bus address 04
37	\$25	%	Percent Sign	Bus address 05
38	\$26	&	Ampersand	Bus address 06
39	\$27	'	Apostrophe	Bus address 07
40	\$28	(Opening parenthesis	Bus address 08
41	\$29)	Closing parenthesis	Bus address 09
42	\$2A	*	Asterisk	Bus address 10
43	\$2B	+	Plus Sign	Bus address 11
44	\$2C	,	Comma	Bus address 12
45	\$2D	-	Hyphen or Minus sign	Bus address 13
46	\$2E	.	Period	Bus address 14
47	\$2F	/	Slash	Bus address 15
48	\$30	0	Zero	Bus address 16
49	\$31	1	One	Bus address 17
50	\$32	2	Two	Bus address 18
51	\$33	3	Three	Bus address 19
52	\$34	4	Four	Bus address 20
53	\$35	5	Five	Bus address 21
54	\$36	6	Six	Bus address 22
55	\$37	7	Seven	Bus address 23
56	\$38	8	Eight	Bus address 24
57	\$39	9	Nine	Bus address 25
58	\$3A	:	Colon	Bus address 26
59	\$3B	;	Semicolon	Bus address 27
60	\$3C	<	Less Than Sign	Bus address 28
61	\$3D	=	Equal Sign	Bus address 29
62	\$3E	>	Greater Than Sign	Bus address 30
63	\$3F	?	Question Mark	Unlisten (UNL)

PERSONAL 488 ISA CARD, PERSONAL 488 PCI CARD*Decimal Values 64 to 95—TAG*

Dec	Hex	Character	Name	Bus Message
64	\$40	@	At sign	Bus address 00
65	\$41	A	Capital A	Bus address 01
66	\$42	B	Capital B	Bus address 02
67	\$43	C	Capital C	Bus address 03
68	\$44	D	Capital D	Bus address 04
69	\$45	E	Capital E	Bus address 05
70	\$46	F	Capital F	Bus address 06
71	\$47	G	Capital G	Bus address 07
72	\$48	H	Capital H	Bus address 08
73	\$49	I	Capital I	Bus address 09
74	\$4A	J	Capital J	Bus address 10
75	\$4B	K	Capital K	Bus address 11
76	\$4C	L	Capital L	Bus address 12
77	\$4D	M	Capital M	Bus address 13
78	\$4E	N	Capital N	Bus address 14
79	\$4F	O	Capital O	Bus address 15
80	\$50	P	Capital P	Bus address 16
81	\$51	Q	Capital Q	Bus address 17
82	\$52	R	Capital R	Bus address 18
83	\$53	S	Capital S	Bus address 19
84	\$54	T	Capital T	Bus address 20
85	\$55	U	Capital U	Bus address 21
86	\$56	V	Capital V	Bus address 22
87	\$57	W	Capital W	Bus address 23
88	\$58	X	Capital X	Bus address 24
89	\$59	Y	Capital Y	Bus address 25
90	\$5A	Z	Capital Z	Bus address 26
91	\$5B	[Opening Bracket	Bus address 27
92	\$5C	\	Backward Slash	Bus address 28
93	\$5D]	Closing Bracket	Bus address 29
94	\$5E	^	Caret	Bus address 30
95	\$5F	_	Underscore	Untalk (UNT)

Decimal Values 96 to 127—SCG

Dec	Hex	Character	Name	Bus Message
Secondary Command Group (SCG)				
96	\$60	'	Grave	Command 00
97	\$61	a	Lowercase A	Command 01
98	\$62	b	Lowercase B	Command 02
99	\$63	c	Lowercase C	Command 03
100	\$64	d	Lowercase D	Command 04
101	\$65	e	Lowercase E	Command 05
102	\$66	f	Lowercase F	Command 06
103	\$67	g	Lowercase G	Command 07
104	\$68	h	Lowercase H	Command 08
105	\$69	i	Lowercase I	Command 09
106	\$6A	j	Lowercase J	Command 10
107	\$6B	k	Lowercase K	Command 11
108	\$6C	l	Lowercase L	Command 12
109	\$6D	m	Lowercase M	Command 13
110	\$6E	n	Lowercase N	Command 14
111	\$6F	o	Lowercase O	Command 15
112	\$70	p	Lowercase P	Command 16
113	\$71	q	Lowercase Q	Command 17
114	\$72	r	Lowercase R	Command 18
115	\$73	s	Lowercase S	Command 19
116	\$74	t	Lowercase T	Command 20
117	\$75	u	Lowercase U	Command 21
118	\$76	v	Lowercase V	Command 22
119	\$77	w	Lowercase W	Command 23
120	\$78	x	Lowercase X	Command 24
121	\$79	y	Lowercase Y	Command 25
122	\$7A	z	Lowercase Z	Command 26
123	\$7B	{	Opening Brace	Command 27
124	\$7C		Vertical Line	Command 28
125	\$7D	}	Closing Brace	Command 29
126	\$7E	~	Tilde	Command 30
127	\$7F	DEL	Delete	Command 31

Abbreviations

***CA:** Not Controller Active mode.

***SC:** Not System Controller mode.

A/D: Analog-to-Digital.

ACG: Addressed Command Group.

ACK: Acknowledgement (ASCII code).

ADC: Analog-to-Digital Converter.

API: Application Program Interface.

ASCII: American Standard Code for Information Interchange.

ATN: Attention line.

BEL: Bell (ASCII code).

BS: Backspace (ASCII code).

CA: Controller Active mode.

CAN: Cancel (ASCII code).

CCL: Character Command Language.

CJC: Cold Junction Compensation.

CMD: Bus Command interpretation.

CR: Carriage Return (ASCII code).

CSR: Calibration Status Register.

CTS: Clear To Send line.

DAV: Data Valid line.

DC1: Device Control 1 (ASCII code).

DC2: Device Control 2 (ASCII code).

DC3: Device Control 3 (ASCII code).

DC4: Device Control 4 (ASCII code).

DCD: Data Carrier Detect line.

DCL: Device Clear bus command.

DDE: Dynamic Data Exchange.

DEL: Delete (ASCII code).

DIO: Data Transfer (I/O line).

DLE: Data Link Escape (ASCII code).

DLL: Dynamic Link Library.

DMA: Direct Memory Access.

DMM: Digital Multimeter.

DSR: Data Set Ready line.

DTR: Data Terminal Ready line.

EEPROM: Electronically Erasable Programmable Read-Only Memory.

EM: End of Medium (ASCII code).

ENQ: Inquiry (ASCII code).

EOI: End-Or-Identify line.

EOL: End-Of-Line character.

EOT: End of Transmission (ASCII code).

EPROM: Erasable Programmable Read-Only Memory.

ESB: Event Status Register bit.

ETX: End of Text (ASCII code).

FCC: Federal Communications Commission.

FF: Form Feed (ASCII code).

FS: File Separator (ASCII code).

GET: Group Execute Trigger bus command.

GPIB: General Purpose Interface Bus.

GS: Group Separator (ASCII code).

GTL: Go To Local bus command.

GUI: Graphical User Interface.

H/W: Hardware.

HT: Horizontal Tab (ASCII code).

IDDC: Invalid Device Dependent Command.

IDDCO: Invalid Device Dependent Command Option.

IEEE: Institute of Electrical and Electronic Engineers.

IFC: Interface Clear line.

IOCTL: Input/Output Control.

ISA: Industry Standard Architecture bus.

ISR: Interrupt Service Routine.

IST: Bus Device Individual Status.

LAG: Listen Address Group bus command.

LED: Light-Emitting Diode.

LF: Line Feed (ASCII code).

LLO: Local Lock Out bus command.

LSB: Least Significant Bit.

MAV: Message Available bit.

MLA: My Listen Address.

MSB: Most Significant Bit.

MSS: Master Summary Status bit.

MTA: My Talk Address.

N/U: Not used.

NAK: Negative Acknowledgement (ASCII code).

NDAC: Not Data Accepted line.

NRFD: Not Ready For Data line.

NUL: Null (ASCII code).

NV-RAM: Non-Volatile Random-Access Memory.

PCI: Peripheral Component Interconnect bus.

PPC: Parallel Poll Configure bus command.

PPD: Parallel Poll Disable bus command.

PPU: Parallel Poll Unconfig bus command.

RAM: Random-Access Memory.

REN: Remote Enable line.

RI: Ring Indicator line.

RMS: Root Mean Square.

RQS: Request for Service bit.

RTD: Resistance Temperature Device.

RTS: Request To Send line.

SC: System Controller mode.

SCG: Secondary Command Group.

SCFI: Standard Commands for Programmable Instruments.

SCSI: Small Computer System Interface bus.

SDC: Selected Device Clear bus command.

SI: Shift In (ASCII code).

SO: Shift Out (ASCII code).

SOH: Start of Header (ASCII code).

SPD: Serial Poll Disable bus command.

SPE: Serial Poll Enable bus command.

SRE: Service Request Enable Register.

SRQ: Service Request Line.

STB: Status Byte Register.

STX: Start of Text (ASCII code).

SUB: Substitute (ASCII code).

SYN: Synchronous Idle (ASCII code).

T/C: Thermocouple.

TAG: Talk Address Group bus command.

TCT: Take Control bus command.

TTL: Transistor-Transistor Logic.

UCG: Universal Command Group.

UNL: Unlisten bus command.

UNT: Untalk bus command.

US: Unit Separator (ASCII code).

VDM: Virtual DOS Machine.

VT: Vertical Tab (ASCII code).

Index

16-bit slot	18, 19
32-bit expansion slot	21
8-bit slot	19

A

AC power	12
Add New Hardware	22, 29
Add New Hardware Wizard	22, 23
Add/Remove Programs	35
ASCIICodes	115
ASCIIControl Codes	116
AT-class machine	18, 19
AT-style interrupt-sharing	18
auxiliary registers	16

B

BIOS	30, 31, 32, 36, 37
------	--------------------

C

card-edge receptacle	19
Change Setting button	27
Change SystemResources dialog box	34
COM	14
Control Panel	22, 26

D

DACK	15, 18
Data Transfer→Output Commands→Enter	31
Data Transfer→Output Commands→Output	31, 36
Desktop Start button	26
Device Manager	25, 26, 27
Device Manager tab	14
Device→MakeNew Device	31
Device→MakeNewDevice	36
DIP-switch settings	13
DIPswitches	35
Direct Memory Access (DMA)	14, 15, 18, 111
DMA	15, 18, 31, 32, 37
DMA channel	16, 19
driver disk	24
Driver488	111

Driver488 software	10, 13
Driver488/W95	7
Driver488/W95 Driver Disk	21
Driver488/W95 Software Settings	111
Driver488/WNT	7
Driver488/WNT Software Settings	112
DRQ	15, 18

F

factory default	16
floppy controller	19

H

hardware installation	12, 20
Hardware-Software Conflicts	111

I

I/O address	14, 15, 16, 18
I/O base address	15, 16
I/O Base Address	111
IEEE 488	10, 12, 13, 14, 19, 111
IEEE 488 bus cables	110
IEEE 488 Bus Errors	110
IEEE 488 controller chip	16
IEEE 488 port connector	12, 20
IEEE 488.2	7, 9
IEEE488 bus	8
IEEE488 Bus and Serial Bus	113
IEEE488 Bus Commands	114
IEEE488 Software Development Disk	30
IEEE488-related .inf file	28
IEEE488.2	27
IEEEController hardware	29
Input/Output (I/O) address	15, 16
Interrupt (IRQ)	15, 17, 18
interrupt level	16
Interrupt Setting	111
IRQ	14, 15, 17
ISA	13, 15, 17, 18, 19, 21, 28, 35
ISA expansion slots	11, 19
ISA-bus expansion slot	20
ISAcad	21

J

jumpers 35

L

local bus 7

LPT 14

M

Manufacturer/Models dialog box 24

multi-tasking 9

My Computer 30

P

PC-bus compatible 11, 19

PC/XTbus 1 8

PCI 10, 21

PCI expansion slots 11, 19

PCI slot 12

PCCard 21

peripheral settings 31

Personal 488/PCI 27

Personal488/AT 21

Personal488/AT (the Personal 488 ISA Card) 27

Personal488/NT disk 1 32

Personal488/PCI 21

Plug and Play and “Legacy” Devices 32

plug-and-play 7, 10

precautions for static-electricity discharge 19

Q

Query→CheckListener 31, 36

R

Resources tab 27

S

SW1 15

SW2 15, 17

System Resources 34

T

terminators 31

W

Windows 95 or 98 7, 9, 10, 13, 21, 22, 28
Windows 95 or 98 software development kit 10, 13
Windows 95/98 Driver Installation/Removal 28
Windows Explorer 30
Windows NT 7, 9, 10, 13, 21, 32, 34
Windows NTService Packet 3 (SP3)
 Driver Installation/Removal 34



© Copyright 1999. Black Box Corporation. All rights reserved.

1000 Park Drive • Lawrence, PA 15055-1018 • 724-746-5500 • Fax 724-746-0746