

# NI-488.2™

---

## NI-488.2 User Manual for Windows

## **Worldwide Technical Support and Product Information**

<http://www.natinst.com>

### **National Instruments Corporate Headquarters**

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

### **Worldwide Offices**

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,  
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,  
France 0 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, India 91805275406,  
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,  
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,  
Spain (Madrid) 91 640 0085, Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,  
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix of this manual.

© Copyright 1998, 1999 National Instruments Corporation. All rights reserved.

# Important Information

---

## Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THEREFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

## Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

## Trademarks

HS488™, natinst.com™, NI-488™, NI-488.2™, and TNT4882™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

## WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human. Applications of National Instruments products involving medical or clinical treatment can create a potential for death or bodily injury caused by product failure, or by errors on the part of the user or application designer. Because each end-user system is customized and differs from National Instruments testing platforms and because a user or application designer may use National Instruments products in combination with other products in a manner not evaluated or contemplated by National Instruments, the user or application designer is ultimately responsible for verifying and validating the suitability of National Instruments products whenever National Instruments products are incorporated in a system or application, including, without limitation, the appropriate design, process and safety level of such system or application.

# Contents

---

## About This Manual

Overview of the NI-488.2 Documentation .....	xi
Conventions .....	xii
Related Documentation.....	xii

## Chapter 1

### Introduction

Setting up and Configuring Your System.....	1-1
Controlling More Than One Interface.....	1-2
Configuration Requirements .....	1-2

## Chapter 2

### Measurement & Automation Explorer

Overview.....	2-1
Starting Measurement & Automation Explorer.....	2-2
Getting Started with NI-488.2 .....	2-3
Troubleshoot NI-488.2 Problems .....	2-4
Scan for GPIB Instruments .....	2-6
Communicate with a GPIB Instrument.....	2-7
Query/Read/Write Communication.....	2-7
Advanced Communication .....	2-9
About Instrument Communication.....	2-10
Adding a New GPIB Instrument.....	2-10
View NI-488.2 Software Version .....	2-12
Monitor, Record, and Display NI-488.2 Calls.....	2-13
View or Change GPIB Interface Settings .....	2-14
Under Windows 98/95.....	2-14
Under Windows NT .....	2-15
View GPIB Instrument Information .....	2-16
Change GPIB Device Templates .....	2-17
Under Windows 98/95.....	2-18
Under Windows NT .....	2-18
Enable/Disable NI-488.2 DOS Support.....	2-18
Under Windows 98/95.....	2-18
Under Windows NT .....	2-19
Access Additional Help and Resources for NI-488.2 and GPIB .....	2-19
NI-488.2 Online Help.....	2-19
National Instruments GPIB Web Site.....	2-19

GPIB-ENET Network Settings (Windows 98/95 Only) .....	2-20
Assign IP Address .....	2-20
Configure Advanced IP Settings .....	2-20
Update GPIB-ENET Firmware .....	2-20

## Chapter 3

### Developing Your NI-488.2 Application

Simple Instrument Control .....	3-1
Interactive Instrument Control .....	3-3
Choosing Your Programming Methodology .....	3-3
Choosing a Method to Access the NI-488.2 Driver .....	3-3
NI-488.2 Language Interfaces .....	3-4
Direct Entry Access .....	3-4
Choosing How to Use the NI-488.2 API .....	3-4
Communicating with a Single GPIB Device .....	3-4
Using Multiple Interfaces and/or Multiple Devices .....	3-5
Checking Status with Global Variables .....	3-6
Status Word (ibsta) .....	3-6
Error Variable (iberr) .....	3-7
Count Variables (ibcnt and ibcntl) .....	3-8
Using Interactive Control to Communicate with Devices .....	3-8
Programming Models .....	3-8
Applications That Communicate with a Single GPIB Device .....	3-8
Items to Include .....	3-8
General Program Steps and Examples .....	3-9
Applications That Use Multiple Interfaces or Communicate with	
Multiple GPIB Devices .....	3-10
Items to Include .....	3-10
General Program Steps and Examples .....	3-11
Language-Specific Programming Instructions .....	3-13
Microsoft Visual C/C++ (Version 2.0 or Later) .....	3-13
Borland C/C++ (Version 4.0 or Later) .....	3-13
Visual Basic (Version 4.0 or Later) .....	3-13
Direct Entry with C .....	3-14
gpib-32.dll Exports .....	3-14
Directly Accessing the gpib-32.dll Exports .....	3-15
Running Existing NI-488.2 Applications .....	3-17
Running Existing Win32 and Win16 NI-488.2 Applications .....	3-17
Running Existing DOS NI-488.2 Applications Under Windows 98/95 .....	3-17
Running Existing DOS NI-488.2 Applications under Windows NT .....	3-18

## Chapter 4

### Debugging Your Application

NI Spy .....	4-1
Global Status Variables .....	4-2
Existing Applications.....	4-3
NI-488.2 Error Codes .....	4-3
Configuration Errors .....	4-4
Timing Errors.....	4-4
Communication Errors.....	4-5
Repeat Addressing.....	4-5
Termination Method.....	4-5
Other Errors .....	4-6

## Chapter 5

### NI Spy Utility

Overview.....	5-1
Starting NI Spy .....	5-1
Using the NI Spy Online Help .....	5-3
Locating Errors with NI Spy.....	5-3
Viewing Properties for Recorded Calls .....	5-3
Exiting NI Spy .....	5-4
Performance Considerations .....	5-4

## Chapter 6

### Interactive Control Utility

Overview.....	6-1
Getting Started with Interactive Control.....	6-1
Interactive Control Syntax .....	6-5
Number Syntax .....	6-5
String Syntax .....	6-6
Address Syntax.....	6-6
Interactive Control Commands .....	6-6
Status Word.....	6-11
Error Information.....	6-12
Count Information.....	6-12

## Chapter 7

### NI-488.2 Programming Techniques

Termination of Data Transfers .....	7-1
High-Speed Data Transfers (HS488).....	7-2
Enabling HS488 .....	7-2
System Configuration Effects on HS488.....	7-3
Waiting for GPIB Conditions .....	7-4
Asynchronous Event Notification in Win32 NI-488.2 Applications .....	7-4
Calling the ibnotify Function .....	7-4
ibnotify Programming Example.....	7-5
Writing Multithreaded Win32 NI-488.2 Applications .....	7-9
Device-Level Calls and Bus Management .....	7-11
Talker/Listener Applications .....	7-11
Serial Polling .....	7-12
Service Requests from IEEE 488 Devices .....	7-12
Service Requests from IEEE 488.2 Devices.....	7-12
Automatic Serial Polling.....	7-13
Stuck SRQ State .....	7-13
Autopolling and Interrupts.....	7-13
SRQ and Serial Polling with Device-Level Traditional NI-488.2 Calls.....	7-14
SRQ and Serial Polling with Multi-Device NI-488.2 Calls.....	7-15
Example 1: Using FindRQS .....	7-15
Example 2: Using AllSpoll.....	7-16
Parallel Polling .....	7-17
Implementing a Parallel Poll.....	7-17
Parallel Polling with Traditional NI-488.2 Calls.....	7-17
Parallel Polling with Multi-Device NI-488.2 Calls .....	7-19

## Appendix A

### GPIB Basics

## Appendix B

### Status Word Conditions

## Appendix C

### Error Codes and Solutions

## Appendix D

### Windows 98/95: Troubleshooting and Common Questions

## Appendix E

### Windows NT: Troubleshooting and Common Questions

## Appendix F

### Technical Support Resources

## Glossary

## Index

## Figures

Figure 1-1.	Linear and Star System Configuration .....	1-1
Figure 1-2.	Example of Multiboard System Configuration .....	1-2
Figure 2-1.	Measurement & Automation Explorer .....	2-2
Figure 2-2.	Select View Documentation .....	2-3
Figure 2-3.	Select Getting Started Wizard .....	2-4
Figure 2-4.	Select NI-488.2 Troubleshooting Wizard .....	2-5
Figure 2-5.	NI-488.2 Troubleshooting Wizard .....	2-5
Figure 2-6.	Select Scan for Instruments .....	2-6
Figure 2-7.	Select Communicate with Instrument .....	2-8
Figure 2-8.	NI-488.2 Communicator .....	2-8
Figure 2-9.	Select Interactive Control Utility .....	2-9
Figure 2-10.	Scan for New Instrument After Scanning .....	2-11
Figure 2-11.	Select About Measurement & Automation Explorer .....	2-12
Figure 2-12.	Select NI Spy .....	2-13
Figure 2-13.	Select Properties .....	2-14
Figure 2-14.	View Interface Information under Windows 98/95 .....	2-15
Figure 2-15.	View Interface Information under Windows NT .....	2-16
Figure 2-16.	GPIB Instrument Information .....	2-17
Figure 3-1.	Select Communicate with Instrument .....	3-2
Figure 3-2.	NI-488.2 Communicator .....	3-2
Figure 4-1.	Select NI Spy .....	4-2
Figure 5-1.	Select NI Spy .....	5-2
Figure 5-2.	NI Spy Application .....	5-3



Figure 6-1. Select Interactive Control Utility ..... 6-2

Figure A-1. GPIB Address Bits ..... A-2

## Tables

Table 3-1. Status Word Layout ..... 3-6

Table 6-1. Syntax for Device-Level Traditional NI-488.2 Calls in  
Interactive Control ..... 6-7

Table 6-2. Syntax for Board-Level Traditional NI-488.2 Calls in  
Interactive Control ..... 6-8

Table 6-3. Syntax for Multi-Device NI-488.2 Calls in Interactive Control ..... 6-9

Table 6-4. Auxiliary Functions in Interactive Control ..... 6-11

Table A-1. GPIB Handshake Lines ..... A-3

Table A-2. GPIB Interface Management Lines ..... A-3

# About This Manual

---

This manual describes the features and functions of the NI-488.2 software for Windows.

## Overview of the NI-488.2 Documentation

---

All of the NI-488.2 documentation is available on the *NI-488.2 for Windows* CD. The following documents can help you learn about and use the NI-488.2 software and GPIB hardware.

- The *Getting Started* card briefly describes how to install your NI-488.2 software and GPIB hardware.
- The *NI-488.2 User Manual for Windows* describes the features and functions of the NI-488.2 software for Windows.
- The *NI-488.2 Function Reference Manual for Windows* describes the NI-488.2 API.
- The *GPIB Hardware Guide* contains detailed instructions about how to configure and install your GPIB hardware. The hardware guide also includes hardware and software specifications and compliance information.

You can access each of these documents online by inserting your *NI-488.2 for Windows* CD and selecting the **View Documentation** option from the autorun menu. You can also view documentation at the National Instruments web site, [www.natinst.com/manuals/](http://www.natinst.com/manuals/).

- The online *NI-488.2 Help* addresses questions you might have about NI-488.2 and includes troubleshooting information and descriptions of the NI-488.2 API.

To access the NI-488.2 online help, first open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, expand the **Devices and Interfaces** item, select a GPIB interface, right-click, and select **NI-488.2 Help**.

## Conventions

---

The following conventions appear in this manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.



This icon denotes a note, which alerts you to important information.

**bold** Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

IEEE 488 and IEEE 488.2 *IEEE 488* and *IEEE 488.2* refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.

*italic* Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

monospace Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

**monospace bold** Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

## Related Documentation

---

The following documents contain information that you may find helpful as you read this manual:

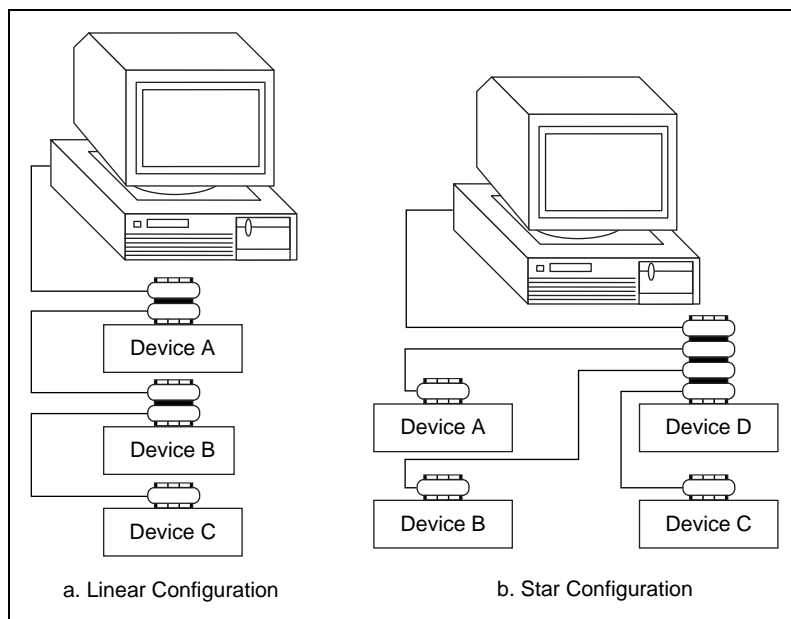
- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*

# Introduction

This chapter explains how to set up your GPIB system.

## Setting up and Configuring Your System

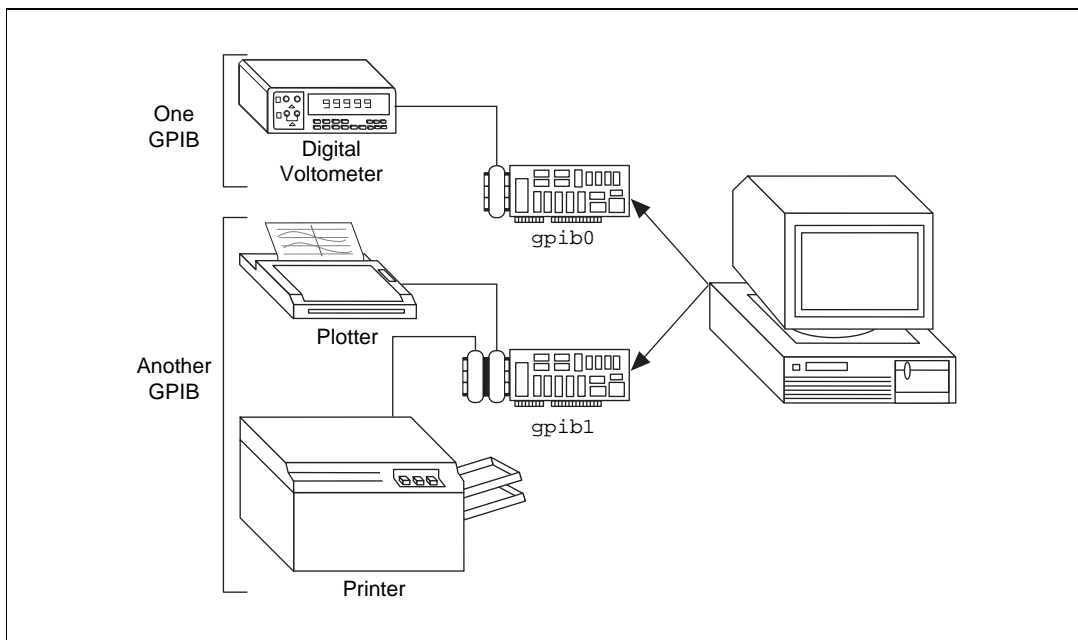
Devices are usually connected with a cable assembly consisting of a shielded 24-conductor cable with both a plug and receptacle connector at each end. With this design, you can link devices in a linear configuration, a star configuration, or a combination of the two configurations. Figure 1-1 shows the linear and star configurations.



**Figure 1-1.** Linear and Star System Configuration

## Controlling More Than One Interface

Figure 1-2 shows an example of a multiboard system configuration. `gpib0` is the access interface for the voltmeter, and `gpib1` is the access interface for the plotter and printer. The control functions of the devices automatically access their respective interfaces.



**Figure 1-2.** Example of Multiboard System Configuration

## Configuration Requirements

To achieve the high data transfer rate that the GPIB was designed for, you must limit the number of devices on the bus and the physical distance between devices. The following restrictions are typical:

- A maximum separation of 4 m between any two devices and an average separation of 2 m over the entire bus.
- A maximum total cable length of 20 m.
- A maximum of 15 devices connected to each bus, with at least two-thirds powered on.

For high-speed operation, the following restrictions apply:

- All devices in the system must be powered on.
- Cable lengths must be as short as possible with up to a maximum of 15 m of cable for each system.
- There must be at least one equivalent device load per meter of cable.

If you want to exceed these limitations, you can use a bus extender to increase the cable length or a bus expander to increase the number of device loads. You can order bus extenders and expanders from National Instruments.

---

# Measurement & Automation Explorer

This chapter describes Measurement & Automation Explorer, an interactive utility you can use with the NI-488.2 software.

To open Measurement & Automation Explorer, select **Start» Programs» National Instruments NI-488.2» Explore GPIB**.

## Overview

---

You can perform the following GPIB-related tasks in Measurement & Automation Explorer:

- Establish basic communication with your GPIB instruments.
- Scan for instruments connected to your GPIB interface.
- Launch the NI-488.2 Getting Started Wizard to get started with GPIB instrument communication.
- Launch the NI-488.2 Troubleshooting Wizard to troubleshoot GPIB and NI-488.2 problems.
- Launch NI Spy to monitor NI-488.2 or VISA API calls to GPIB interfaces.
- View information about your GPIB hardware and NI-488.2 software.
- Reconfigure the GPIB interface settings.
- Locate additional help resources for GPIB and NI-488.2.

# Starting Measurement & Automation Explorer

To start Measurement & Automation Explorer, select **Start» Programs»National Instruments NI-488.2»Explore GPIB**. Figure 2-1 shows Measurement & Automation Explorer.

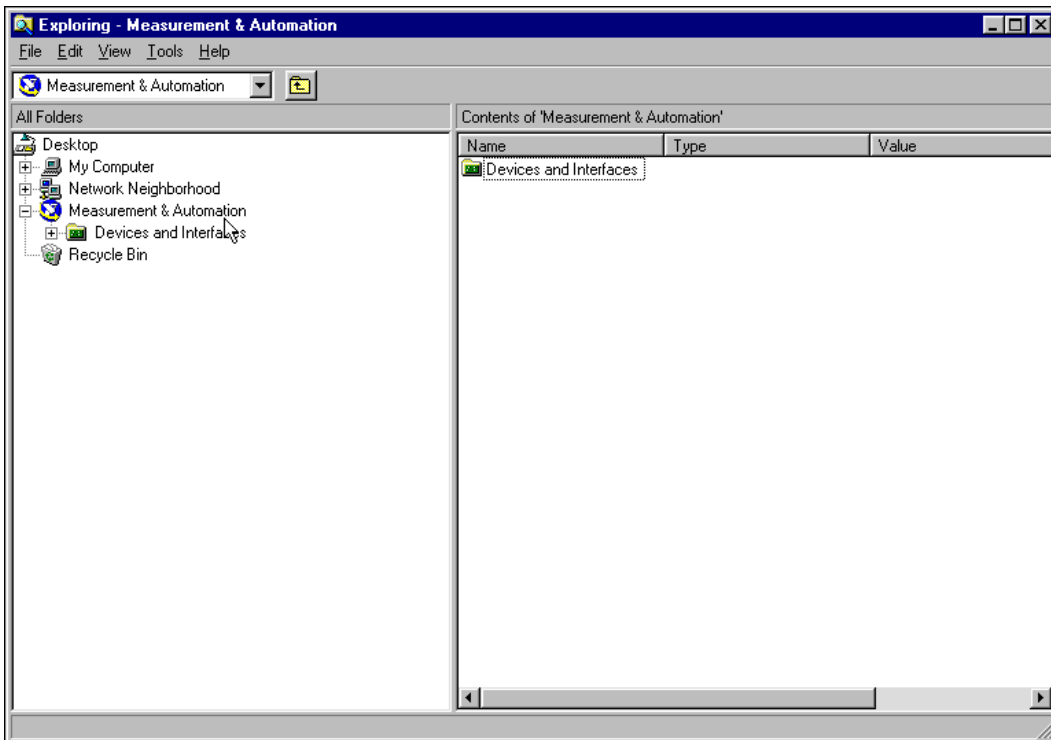


Figure 2-1. Measurement & Automation Explorer



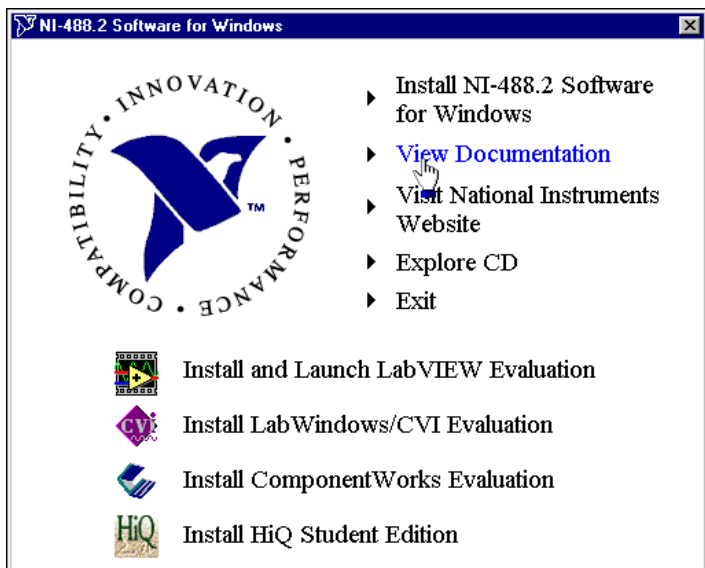
## Getting Started with NI-488.2

The NI-488.2 Getting Started Wizard helps you get started with GPIB instrument communication using Measurement & Automation Explorer.

The general steps to get started with NI-488.2 are as follows:

1. Install the NI-488.2 software and GPIB hardware by following the instructions on your *Getting Started* card.

If you do not have a *Getting Started* card, select the **View Documentation** option from the *NI-488.2 for Windows* CD autorun screen, as shown in Figure 2-2. Then, select your operating system and GPIB interface.



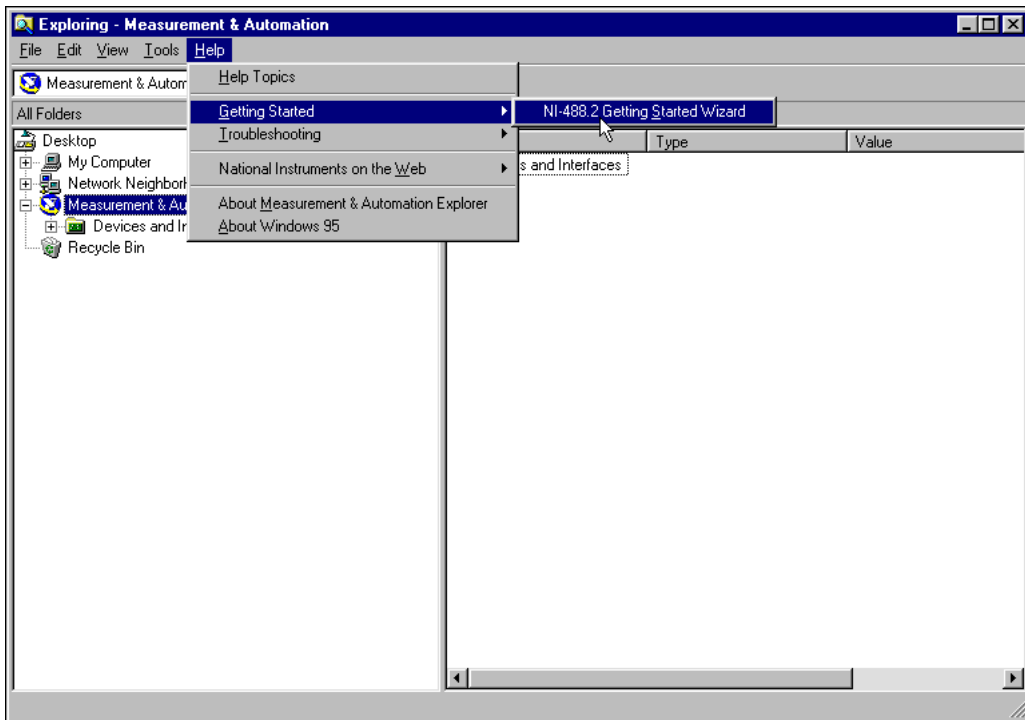
**Figure 2-2.** Select View Documentation

2. Verify the installation and establish basic communication with your GPIB instruments using the NI-488.2 Getting Started Wizard.

To run the Getting Started Wizard, select **Measurement & Automation** in the left window frame of Measurement & Automation Explorer, then choose **Help»Getting Started»NI-488.2 Getting Started Wizard**, as shown in Figure 2-3.



**Note** The NI-488.2 Getting Started Wizard runs automatically after you install the *NI-488.2 for Windows* software and restart your system.



**Figure 2-3.** Select Getting Started Wizard

The Getting Started Wizard guides you through the process of verifying installation and establishing communication with your instrument. Follow the steps outlined by the Getting Started Wizard by clicking on the items in the list as they are highlighted.

3. Start to use NI-488.2:
  - Run an existing NI-488.2 application, or
  - Develop a new NI-488.2 application.

## Troubleshoot NI-488.2 Problems

To troubleshoot NI-488.2 problems, run the NI-488.2 Troubleshooting Wizard. Select **Measurement & Automation** in the left window frame of Measurement & Automation Explorer, then choose **Help» Troubleshooting»NI-488.2 Troubleshooting Wizard**, as shown in Figure 2-4.

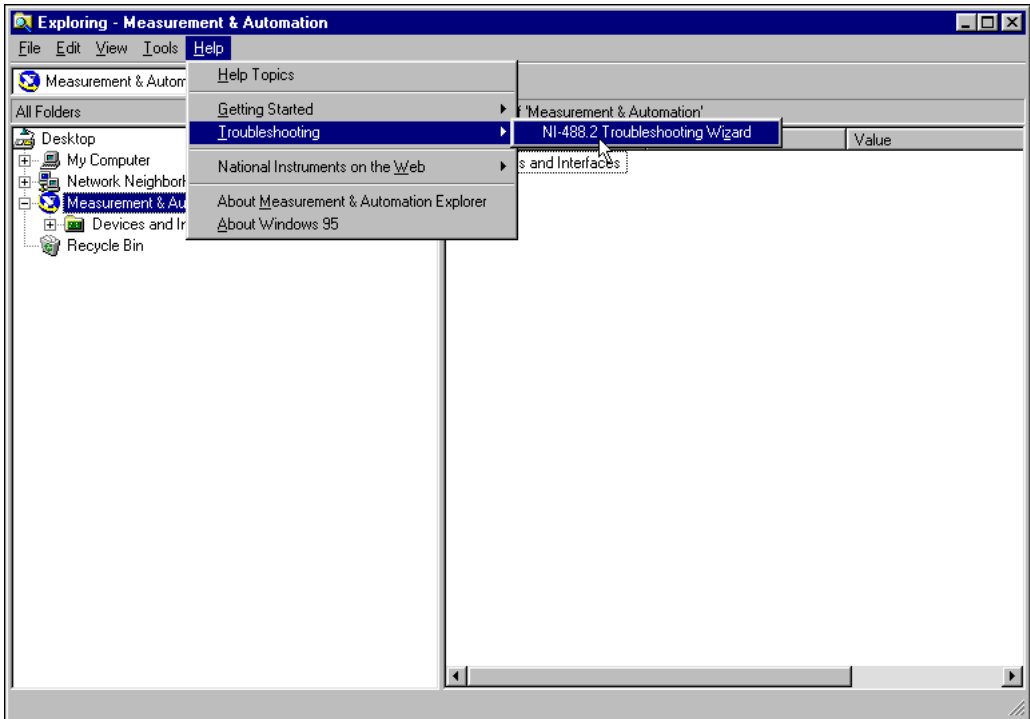


Figure 2-4. Select NI-488.2 Troubleshooting Wizard

Figure 2-5 show the Troubleshooting Wizard after it has tested a GPIB interface. You can access online help for the Troubleshooting Wizard by clicking on the **Help** button.



Figure 2-5. NI-488.2 Troubleshooting Wizard

## Scan for GPIB Instruments

To scan for connected GPIB instruments, follow these steps:

1. Make sure that your instrument is connected to the GPIB interface and powered on.
2. Select the GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Scan for Instruments**, as shown in Figure 2-6.

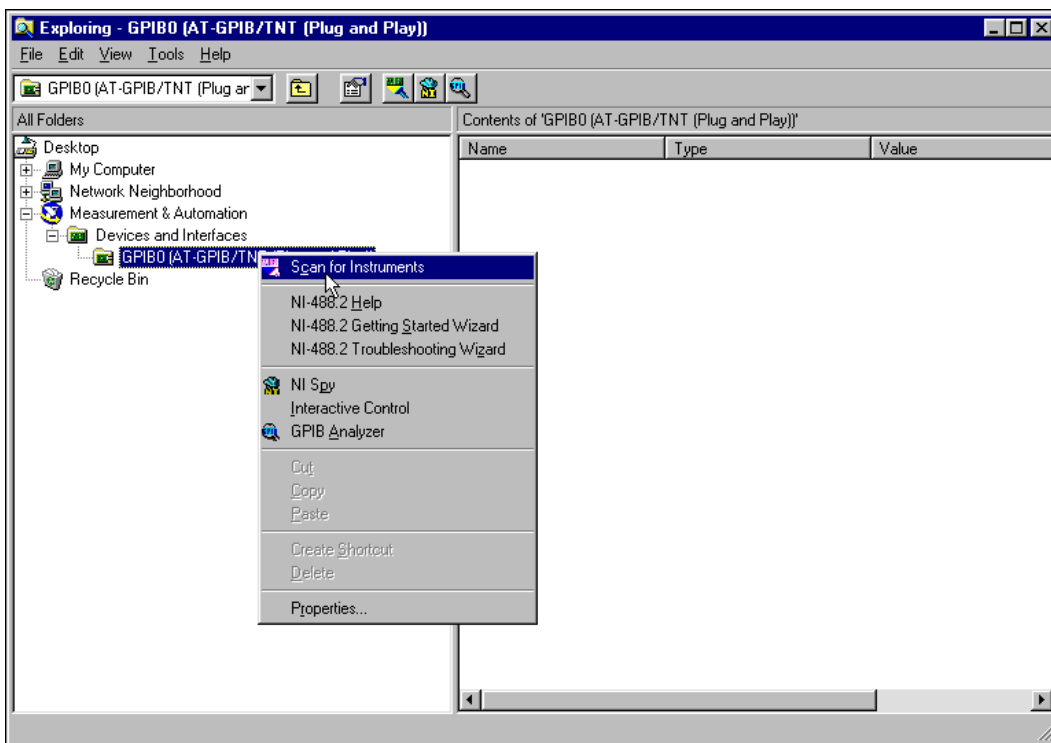


Figure 2-6. Select Scan for Instruments

Each instrument that is found is listed in the right window frame.



**Note** If the message **Instruments not Found** appears in the right window frame, **Scan for Instruments** failed because it did not find any instruments. Make sure that your GPIB instruments are properly connected to the GPIB interface with a GPIB cable and that they are powered on, then repeat the **Scan for Instruments**.



**Note** If the message **Instruments Enumeration Failed** appears in the right window frame, **Scan for Instruments** found too many Listeners on the GPIB. You might encounter this message if you have a running GPIB Analyzer with the GPIB handshake option enabled. To fix this problem, disable the GPIB handshake option in the GPIB Analyzer. You might also encounter this message if you have a GPIB extender in your system. In this case, **Scan for Instruments** cannot detect any instruments connected to your GPIB interface. You can verify communication with the instruments using the Interactive Control utility. To view the Interactive Control utility help on verifying instrument communication, select the GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Interactive Control**. Then, type help "Interactive Control:getting started".

## Communicate with a GPIB Instrument

---

### Query/Read/Write Communication

To establish basic communication with a GPIB instrument using the NI-488.2 Communicator, follow these steps:

1. Make sure that you have already scanned for instruments as described in the previous section, [Scan for GPIB Instruments](#).
2. Start the NI-488.2 Communicator by selecting the GPIB instrument in the right window frame of Measurement & Automation Explorer, right-clicking, and choosing **Communicate with Instrument**, as shown in Figure 2-7.

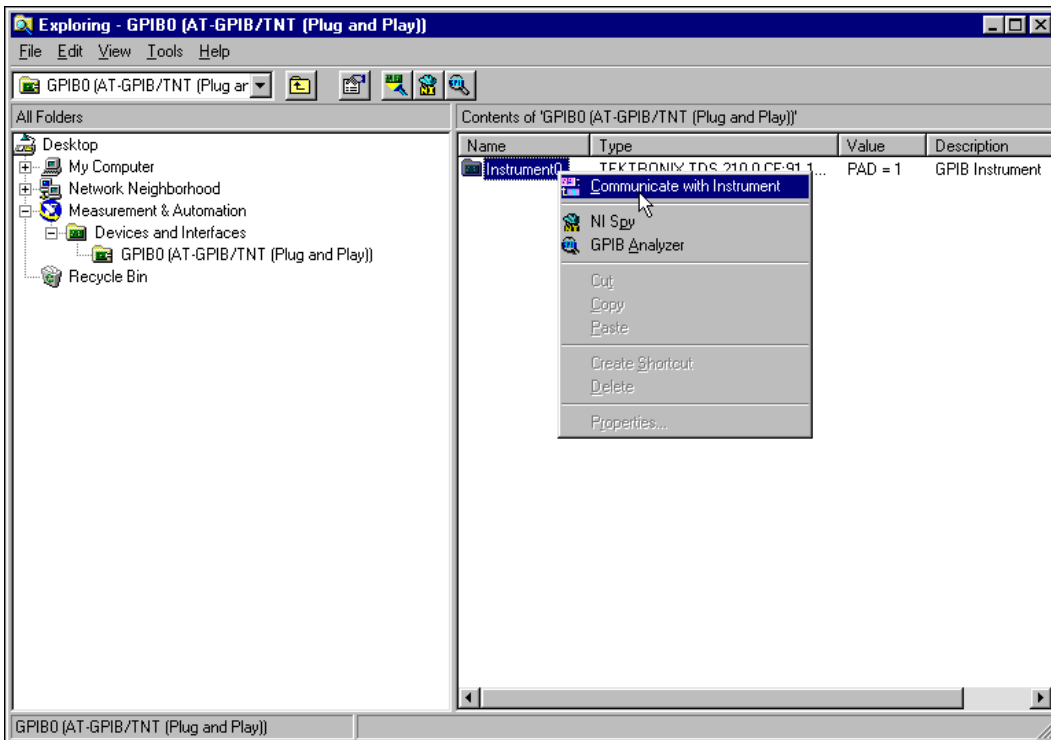


Figure 2-7. Select Communicate with Instrument

Figure 2-8 shows the NI-488.2 Communicator.

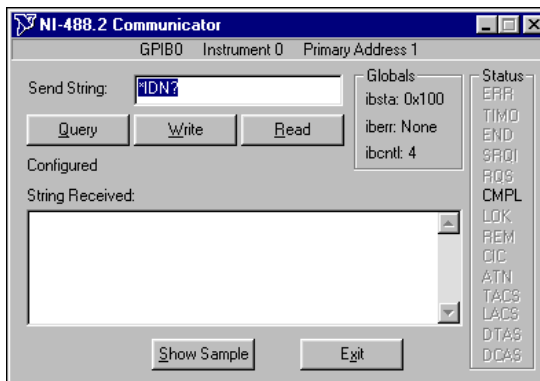


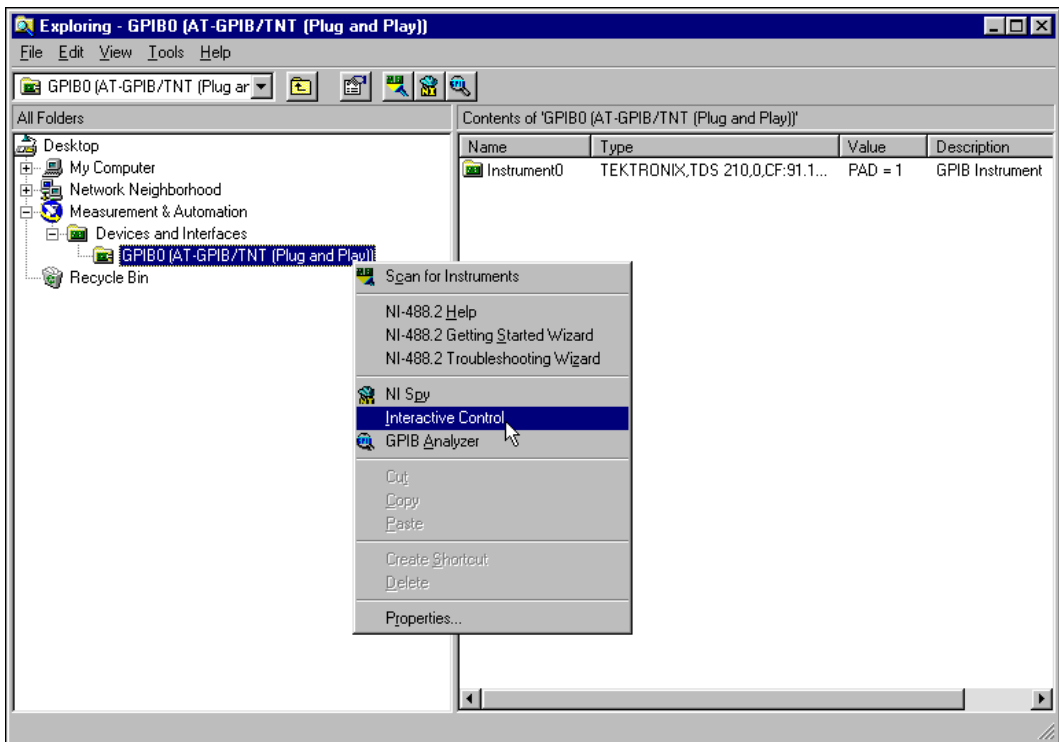
Figure 2-8. NI-488.2 Communicator

3. Choose **Query** to write a command to the instrument then read a response back. Choose **Write** to write a command to the instrument. Or, choose **Read** to read a response from the instrument.

## Advanced Communication

For more advanced interactive communication with GPIB instruments, run the Interactive Control utility by following these steps:

1. Select the GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Interactive Control**, as shown in Figure 2-9.



**Figure 2-9.** Select Interactive Control Utility

2. Type in NI-488.2 API calls interactively to communicate with the GPIB instrument in the Interactive Control utility. For example, you might use `ibdev`, `ibclr`, `ibwrt`, `ibrdr`, and `ibonl`. Type in `help` to view the online help for Interactive Control.

## About Instrument Communication

Refer to the documentation that came with your GPIB instrument for a description of the commands that your instrument understands. Most instruments respond to the \*IDN? command by returning an identification string.

## Adding a New GPIB Instrument

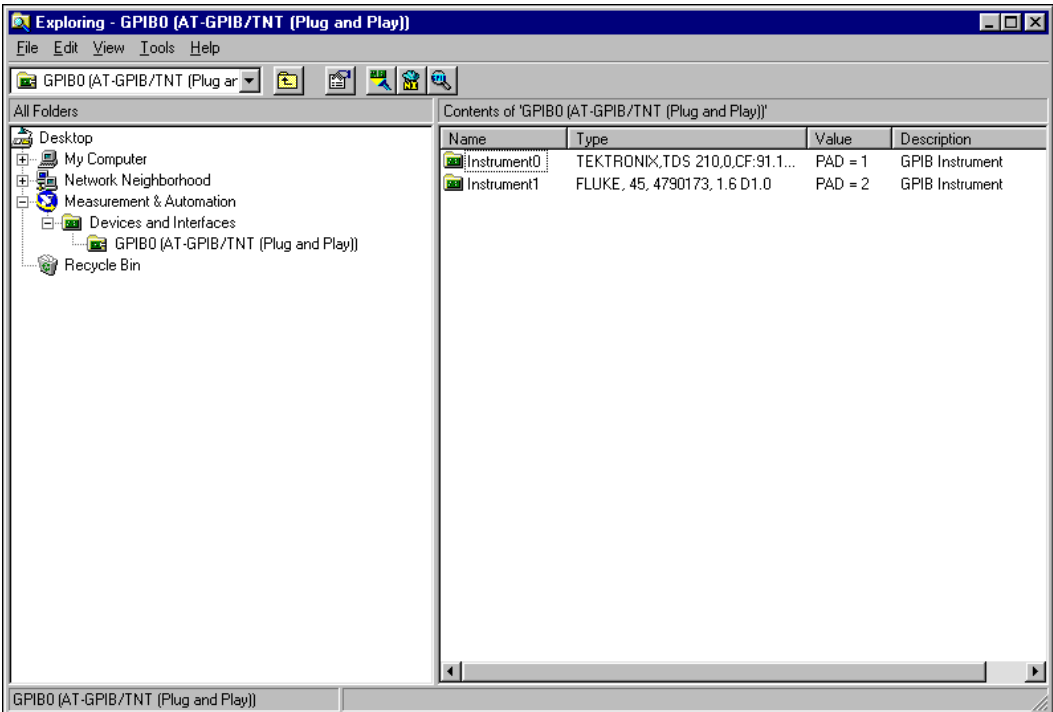
---

To add a new GPIB instrument in Measurement & Automation Explorer, follow these steps:

1. Make sure that your instrument is connected to the GPIB interface and powered on.
2. Scan for connected GPIB instruments. For help, refer to the section [Scan for GPIB Instruments](#).

Each instrument that is powered on and connected to the selected interface is listed in the right window frame. Figure 2-10 shows a successful scan where two instruments were detected.





**Figure 2-10.** Scan for New Instrument After Scanning



**Note** If the message **Instruments not Found** appears in the right window frame, **Scan for Instruments** failed because it did not find any instruments. Make sure that your GPIB instruments are properly connected to the GPIB interface with a GPIB cable and that they are powered on, then repeat the **Scan for Instruments**.

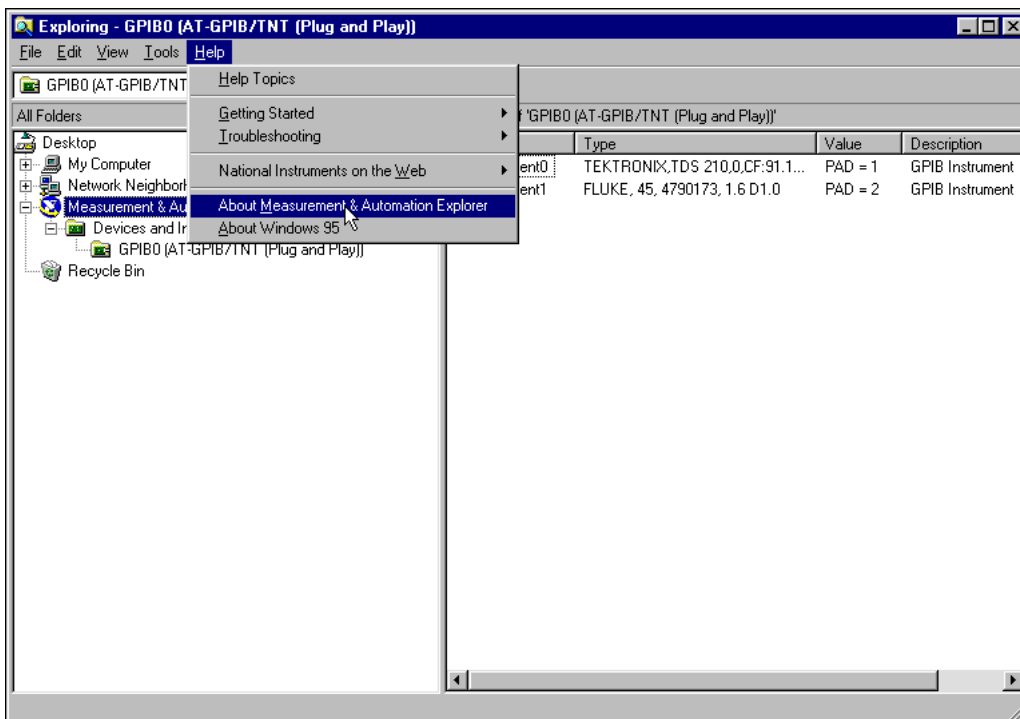


**Note** If the message **Instruments Enumeration Failed** appears in the right window frame, **Scan for Instruments** found too many Listeners on the GPIB. You might encounter this message if you have a running GPIB Analyzer with the GPIB handshake option enabled. To fix this problem, disable the GPIB handshake option in the GPIB Analyzer. You might also encounter this message if you have a GPIB extender in your system. In this case, **Scan for Instruments** cannot detect any instruments connected to your GPIB interface. You can verify communication with the instruments using the Interactive Control utility. To view the Interactive Control utility help on verifying instrument communication, select the GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Interactive Control**. Then, type help "Interactive Control:getting started".

## View NI-488.2 Software Version

To view the NI-488.2 software version, follow these steps:

1. Select **Measurement & Automation** in the left window frame of Measurement & Automation Explorer, then choose **Help»About Measurement & Automation Explorer**, as shown in Figure 2-11.



**Figure 2-11.** Select About Measurement & Automation Explorer

2. In the **About Measurement & Automation Explorer** dialog box, click on the **System Info** button.
3. Select the **Software** tab.
  - The **Name** column displays the name of the software.
  - The **Type** column displays operating system information.
  - The **Value** column displays the version number of the software.
  - The **Description** column displays additional information.

# Monitor, Record, and Display NI-488.2 Calls

To monitor NI-488.2 calls, run NI Spy by following these steps:

1. Select a GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **NI Spy**, as shown in Figure 2-12.

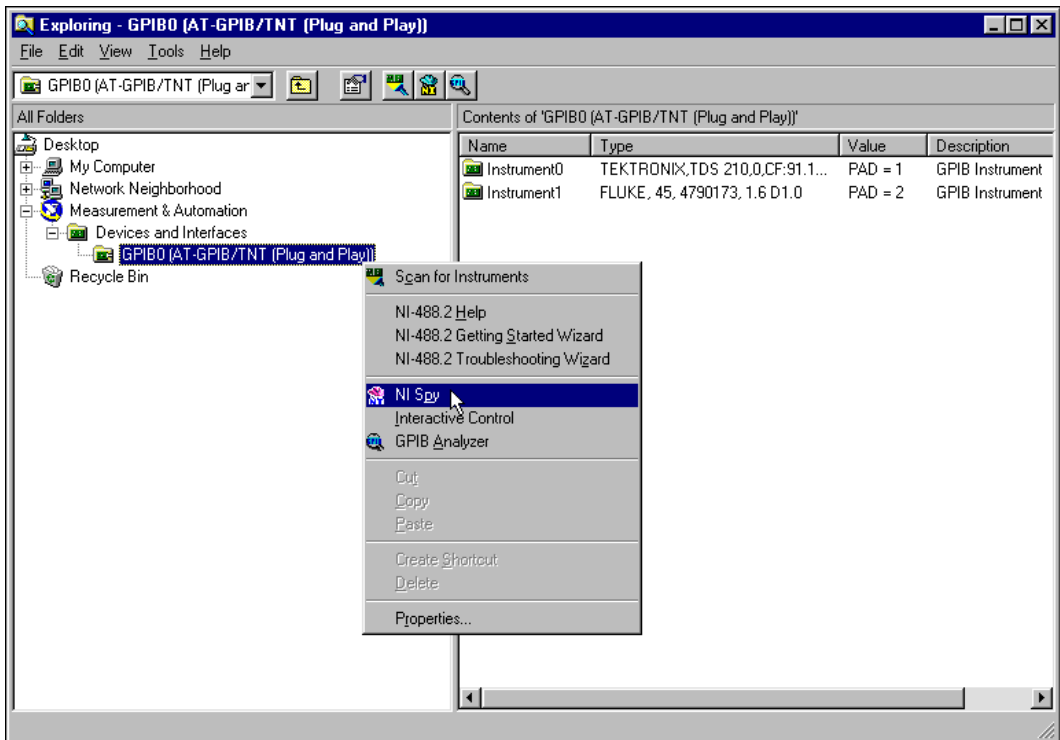


Figure 2-12. Select NI Spy

2. Start a capture by clicking on the blue arrow on the toolbar of NI Spy.
3. Refer to the NI Spy online help for more information. To access the NI Spy online help, select **Help** from the **NI Spy** menu.

# View or Change GPIB Interface Settings

## Under Windows 98/95

To view or change GPIB interface information, follow these steps:

1. To open the Configuration utility, select a GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Properties**, as shown in Figure 2-13.

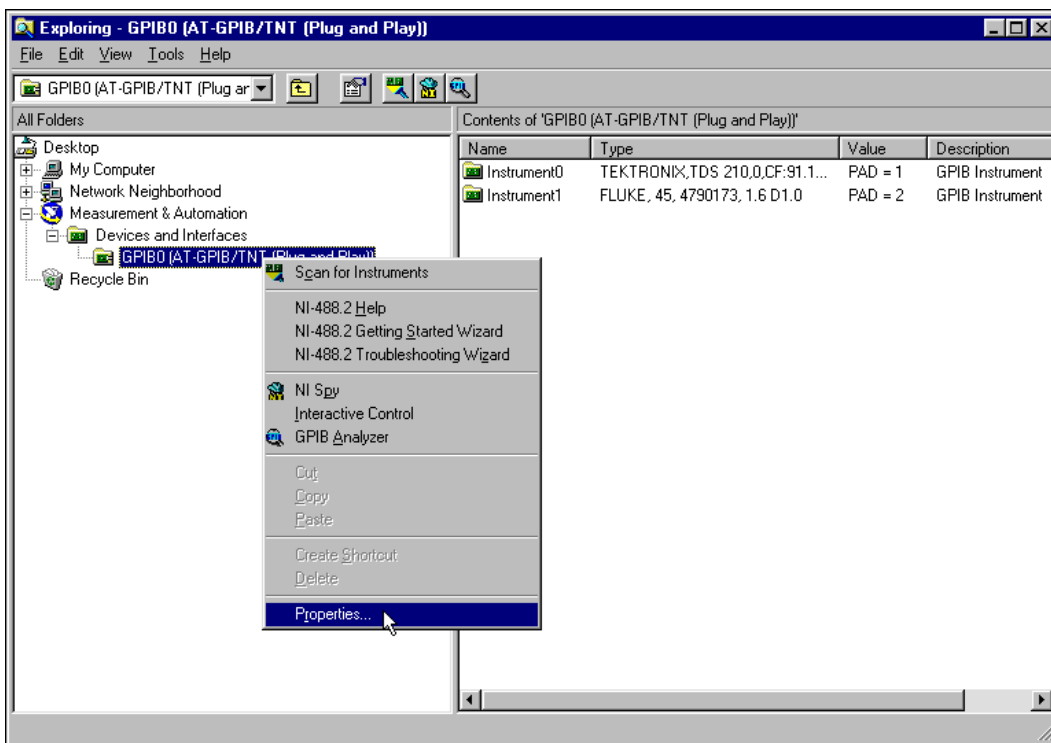
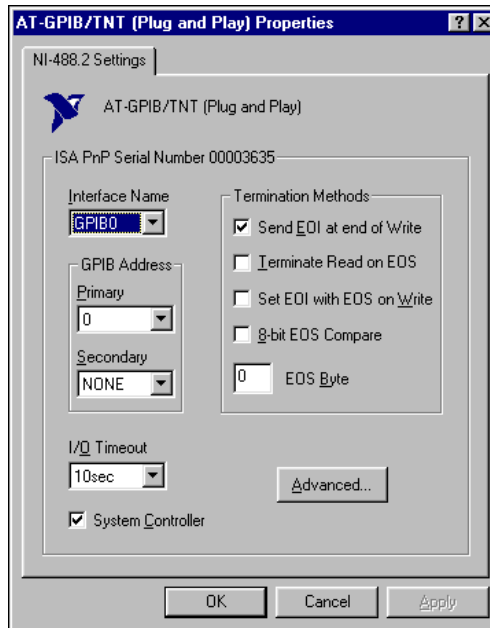


Figure 2-13. Select Properties

Figure 2-14 shows the NI-488.2 Configuration utility **Properties** page for an AT-GPIB/TNT (Plug and Play) interface.



**Figure 2-14.** View Interface Information under Windows 98/95

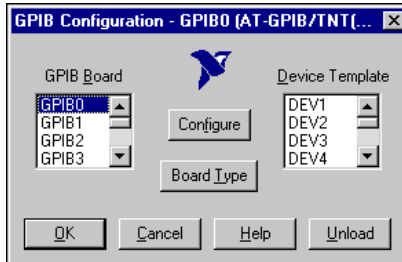
You can use the NI-488.2 Configuration utility to change the settings for your interface. To view the online help, click on the ? button in the upper-right corner of the dialog box, then click on the item you need help with.

## Under Windows NT

To view or change GPIB interface information, follow these steps:

1. Select a GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Properties**, as shown in Figure 2-13.

Figure 2-15 shows the NI-488.2 Configuration utility.



**Figure 2-15.** View Interface Information under Windows NT

2. Select the correct GPIB interface and click on **Configure**.

You can use the NI-488.2 Configuration utility to change the settings for your interface. To view the online help, click on the **Help** button.

## View GPIB Instrument Information

---

To view GPIB instrument information, follow these steps:

1. Scan for connected GPIB instruments, if you have not already done so. For help, refer to the section [Scan for GPIB Instruments](#).
2. Select the GPIB interface listed under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer.
3. View the instrument information in the right window frame of Measurement & Automation Explorer, as shown in Figure 2-16.

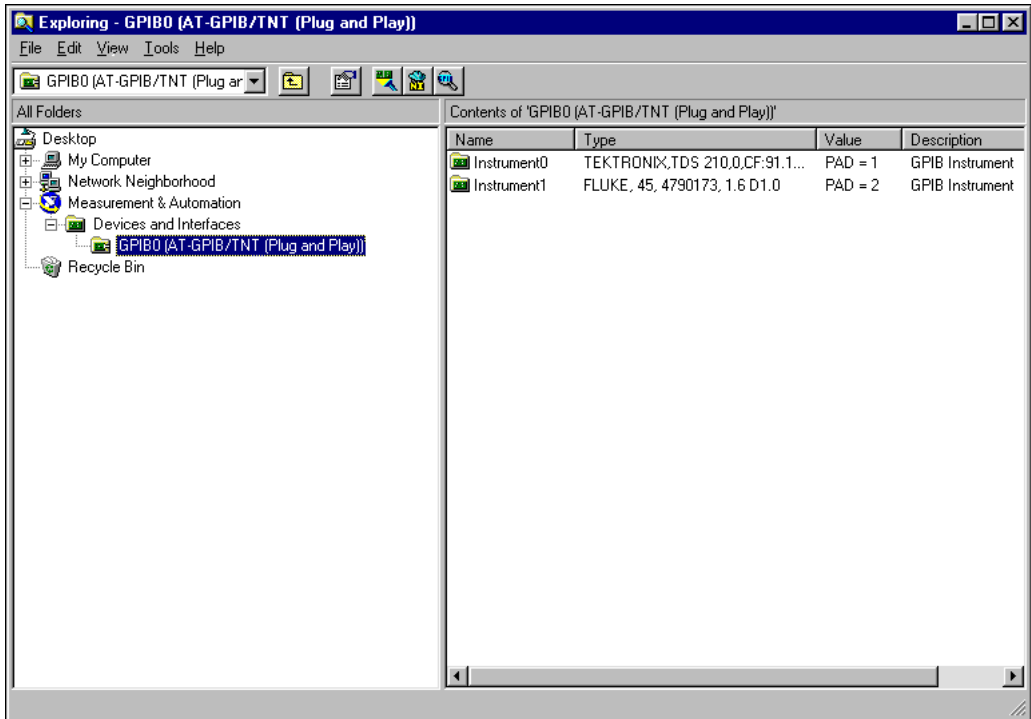


Figure 2-16. GPIB Instrument Information

- The **Name** column displays the logical instrument name assigned by Measurement & Automation Explorer.
- The **Type** column displays the instrument's response to the identification query (\*IDN?).
- The **Value** column displays the primary (PAD) and secondary (SAD) addresses of the instrument.
- The **Description** column identifies the instrument as a GPIB instrument.

## Change GPIB Device Templates

For older NI-488.2 applications, you might need to modify one of the device templates to find a given GPIB instrument by name, for example, `ibfind("fluke45")`. These older applications still use `ibfind` instead of the preferred `ibdev` NI-488.2 call to get a device handle. New applications should avoid using `ibfind` to obtain device handles and use `ibdev` instead. `ibdev` allows you to dynamically configure your GPIB

device handle and frees the application from unnecessary device name requirements.

If you must modify a device template, follow these steps.

## Under Windows 98/95

To reconfigure GPIB device templates under Windows 98/95, follow these steps:

1. Launch the Windows 98/95 Device Manager.
  - a. Select **Start»Settings»Control Panel**.
  - b. Double-click on the **System** icon.
  - c. Select the **Device Manager** tab.
2. Select **National Instruments GPIB Interfaces**.
3. Click on the **Properties** button.
4. Select the **Device Templates** tab.
5. Use the context-sensitive help for more information.

## Under Windows NT

To reconfigure GPIB device templates under Windows NT, follow these steps:

1. Select a GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Properties** to launch the NI-488.2 Configuration utility.
2. Change the GPIB device templates. Use the online help, available in the NI-488.2 Configuration utility, for more information.

## Enable/Disable NI-488.2 DOS Support

---

### Under Windows 98/95

To enable NI-488.2 DOS support under Windows 98/95, follow these steps:

1. Select a GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer.
2. Select **Tools»Settings»NI-488.2**.
3. Enable or disable DOS support in the **NI-488.2 Settings** dialog box.



## Under Windows NT

To enable NI-488.2 DOS support under Windows NT, follow these steps:

1. Open your `config.nt` file, located in the Windows NT `system32` directory (for example, `c:\windows\system32`).

2. Find the following lines of code:

```
REM ***To run DOS GPIB applications, uncomment the
REM ***following line
REM device=<path>\doswin16\gplib-nt.com
```

where `<path>` is the directory in which you installed the NI-488.2 software.

3. Remove `REM` from the last line so that it reads

```
device=<path>\doswin16\gplib-nt.com
```

To disable DOS support, add `REM` back to the line of code where it was removed.

## Access Additional Help and Resources for NI-488.2 and GPIB

---

### NI-488.2 Online Help

The NI-488.2 online help includes information on getting started, troubleshooting, application development, and answers to frequently asked questions, as well as a complete NI-488.2 API function reference. To view the NI-488.2 online help, from **Measurement & Automation Explorer**, select a GPIB interface in the left window frame under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help**.

### National Instruments GPIB Web Site

To access the National Instruments web site for GPIB, from **Measurement & Automation Explorer**, select the **Measurement & Automation** icon, then select **Help»National Instruments on the Web»GPIB Home Page**.

## GPIB-ENET Network Settings (Windows 98/95 Only)

---

### Assign IP Address



**Note** The Assign IP Address utility requires that you have network administrator knowledge. Please contact your network administrator before running this utility.

1. In Measurement & Automation Explorer, select a GPIB-ENET interface under **Devices and Interfaces** in the left window frame, right-click, and choose **Assign IP Address**.
2. Use the Assign IP Address utility to assign the IP address.  
Use the online help, available in the Assign IP Address utility, for more information.

### Configure Advanced IP Settings



**Note** The Advanced IP Settings utility requires that you have network administrator knowledge. Please contact your network administrator before running this utility.

1. In Measurement & Automation Explorer, select a GPIB-ENET interface under **Devices and Interfaces** in the left window frame, right-click, and choose **Advanced IP Settings**.
2. Use the Advanced IP Settings utility to assign subnet information.  
Use the online help, available in the Advanced IP Settings utility, for more information.

### Update GPIB-ENET Firmware

1. In Measurement & Automation Explorer, select a GPIB-ENET interface under **Devices and Interfaces** in the left window frame, right-click, and choose **Update Firmware**.
2. Use the Update Firmware utility to update firmware.  
Use the online help, available in the Update Firmware utility, for more information.

---

# Developing Your NI-488.2 Application

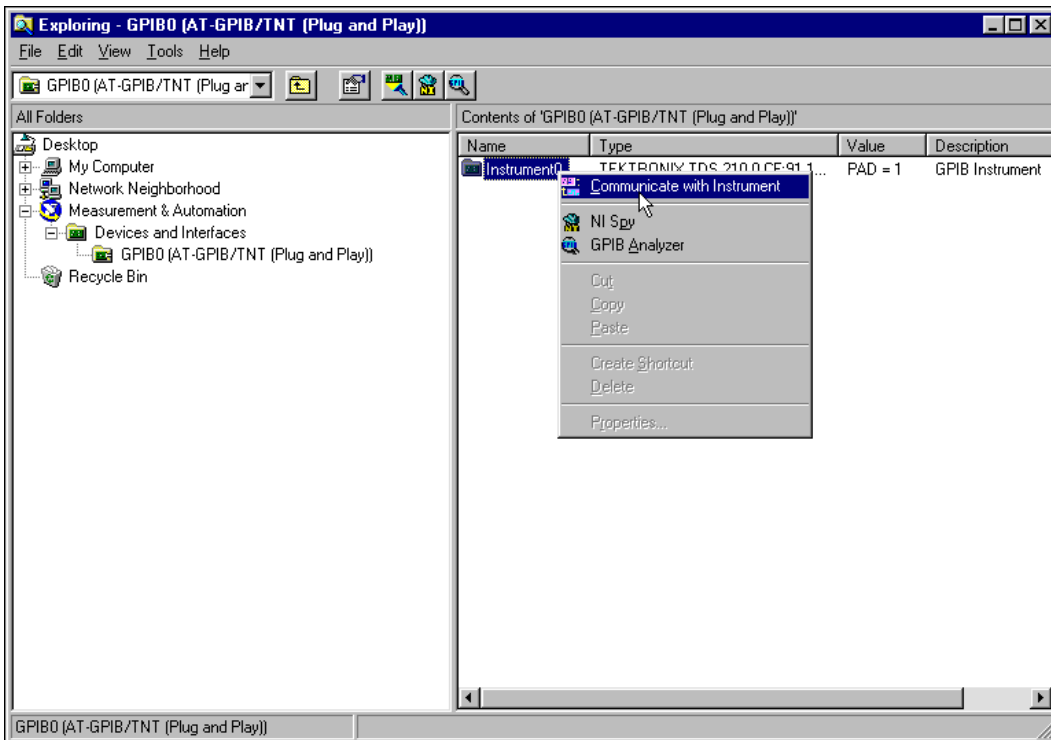
This chapter explains how to develop an NI-488.2 application using the NI-488.2 API.

## Simple Instrument Control

---

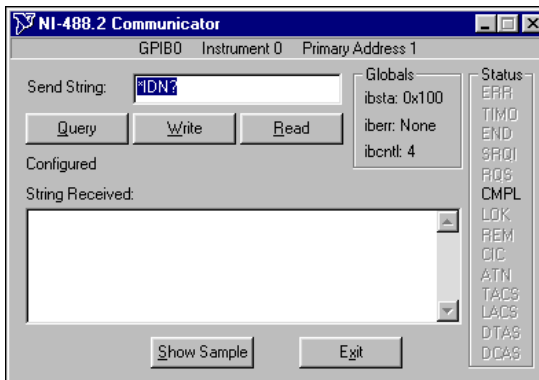
To establish basic communication with a GPIB instrument using the NI-488.2 Communicator, follow these steps:

1. Make sure that you have already scanned for instruments, as described in the section *Scan for GPIB Instruments* in Chapter 2, *Measurement & Automation Explorer*.
2. To start the NI-488.2 Communicator, select the GPIB instrument in the right window frame of Measurement & Automation Explorer, right-click, and choose **Communicate with Instrument**, as shown in Figure 3-1.



**Figure 3-1.** Select Communicate with Instrument

Figure 3-2 shows the NI-488.2 Communicator application.



**Figure 3-2.** NI-488.2 Communicator

- To use the NI-488.2 Communicator, choose **Query** to write a command to the instrument then automatically read a response back, choose **Write** to write a command to the instrument, or choose **Read** to read a response from the instrument.

Click on the **Show Sample** button to view sample C/C++ code that performs a simple query of a GPIB instrument.

## Interactive Instrument Control

---

Before you begin writing your application, you might want to use the Interactive Control utility to communicate with your instruments interactively by typing in commands from the keyboard rather than issuing them from an application. You can use the Interactive Control utility to learn to communicate with your instruments using the NI-488.2 API. For specific device communication instructions, refer to the user manual that came with your instrument. For information about using the Interactive Control utility and detailed examples, refer to Chapter 6, *Interactive Control Utility*.

To start Interactive Control, launch Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select the GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **Interactive Control**

To establish simple communication, use the calls modeled in the sample provided by the NI-488.2 Communicator. The first NI-488.2 API call to make is `ibdev`. After that, you can use `ibwrt` and `ibrd` to send commands to and read responses from the GPIB instrument. Finally, use `ibonl` to put the device handle offline when you are finished. Refer to Chapter 6, *Interactive Control Utility*, for more information.

## Choosing Your Programming Methodology

---

Based on your development environment, you can select a method for accessing the driver, and based on your NI-488.2 programming needs, you can choose how to use the NI-488.2 API.

### Choosing a Method to Access the NI-488.2 Driver

Applications can access the NI-488.2 dynamic link library (DLL), `gpiib-32.dll`, either by using an NI-488.2 language interface or by direct access.

## NI-488.2 Language Interfaces

You can use a language interface if your program is written in Microsoft Visual C/C++ (2.0 or later), Borland C/C++ (4.0 or later), or Microsoft Visual Basic (4.0 or later). Otherwise, you must access `gpib-32.dll` directly.

## Direct Entry Access

You can access the DLL directly from any programming environment that allows you to request addresses of variables and functions that a DLL exports. `gpib-32.dll` exports pointers to each of the global variables and all the NI-488.2 calls.

## Choosing How to Use the NI-488.2 API

The NI-488.2 API has two subsets of calls to meet your application needs. Both of these sets, the traditional calls and the multi-device calls, are compatible across computer platforms and operating systems, so you can port programs to other platforms with little or no source code modification. For most applications, the traditional NI-488.2 calls are sufficient. If you have a complex configuration with one or more interfaces and multiple devices, use the multi-device NI-488.2 calls. Whichever option you choose, bus management operations necessary for device communication are performed automatically.

The following sections describe some differences between the traditional NI-488.2 calls and the multi-device NI-488.2 calls.

## Communicating with a Single GPIB Device

If your system has only one device attached to each interface, the traditional NI-488.2 calls are probably sufficient for your programming needs. A typical NI-488.2 application with a single device has three phases:

- Initialization: use `ibdev` to get a handle and use `ibclr` to clear the device.
- Device Communication: use `ibwrt`, `ibrd`, `ibtrg`, `ibrsp`, and `ibwait` to communicate with the device.
- Cleanup: use `ibonl` to put the handle offline.

Refer to the sample applications that are installed with the NI-488.2 software to see detailed examples for different GPIB device types.

For NI-488.2 applications that need to control the GPIB in non-typical ways, for example, to communicate with non-compliant GPIB devices, there are a set of low-level functions that perform rudimentary GPIB applications. If you use these functions, you need to understand GPIB management details like how to address talkers and listeners. Refer to Appendix A, *GPIB Basics*, for some details on GPIB management.

The set of low-level functions are called board-level functions. They access the interface directly and require you to handle the addressing and bus management protocol. These functions give you the flexibility and control to handle situations such as the following:

- Communicating with non-compliant (non-IEEE 488.2) devices.
- Altering various low-level interface configurations.
- Managing the bus in non-typical ways.

Board-level functions that an NI-488.2 application might use include the following: `ibcmd`, `ibrd`, `ibwrt`, and `ibconfig`. For a detailed list, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help** to view the online help.

## Using Multiple Interfaces and/or Multiple Devices

When your system includes an interface that must access multiple devices, use the multi-device NI-488.2 calls, which can perform the following tasks with a single call:

- Find the Listeners on the bus using `FindLstn`.
- Find a device requesting service using `FindRQS`.
- Determine the state of the SRQ line, or wait for SRQ to be asserted using `TestSRQ` or `WaitSRQ`.
- Address multiple devices to receive a command using `SendList`.

You can mix board-level traditional NI-488.2 calls with the multi-device NI-488.2 calls to have access to all the NI-488.2 functionality.

## Checking Status with Global Variables

Each NI-488.2 API call updates four global variables to reflect the status of the device or interface that you are using. These global status variables are the status word (`ibsta`), the error variable (`iberr`), and the count variables (`ibcnt` and `ibcnt1`). They contain useful information about the performance of your application. Your application should check these variables after each NI-488.2 call. The following sections describe each of these global variables and how you can use them in your application.



**Note** If your application is a multithreaded application, refer to the section [Writing Multithreaded Win32 NI-488.2 Applications](#) in Chapter 7, [NI-488.2 Programming Techniques](#).

### Status Word (`ibsta`)

All NI-488.2 calls update a global status word, `ibsta`, which contains information about the state of the GPIB and the GPIB hardware. The value stored in `ibsta` is the return value of all the traditional NI-488.2 calls, except `ibfind` and `ibdev`. You can examine various status bits in `ibsta` and use that information to make decisions about continued processing. If you check for possible errors after each call using the `ibsta` ERR bit, debugging your application is much easier.

`ibsta` is a 16-bit value. A bit value of one (1) indicates that a certain condition is in effect. A bit value of zero (0) indicates that the condition is not in effect. Each bit in `ibsta` can be set for device-level traditional NI-488.2 calls (`dev`), board-level traditional NI-488.2 calls and multi-device NI-488.2 calls (`brd`), or all (`dev, brd`).

Table 3-1 shows the condition that each bit position represents, the bit mnemonics, and the type of calls for which the bit can be set. For a detailed explanation of each status condition, refer to Appendix B, [Status Word Conditions](#).

**Table 3-1.** Status Word Layout

Mnemonic	Bit Pos	Hex Value	Type	Description
ERR	15	8000	dev, brd	NI-488.2 error
TIMO	14	4000	dev, brd	Time limit exceeded
END	13	2000	dev, brd	END or EOS detected



**Table 3-1.** Status Word Layout (Continued)

Mnemonic	Bit Pos	Hex Value	Type	Description
SRQI	12	1000	brd	SRQ interrupt received
RQS	11	800	dev	Device requesting service
CMPL	8	100	dev, brd	I/O completed
LOK	7	80	brd	Lockout State
REM	6	40	brd	Remote State
CIC	5	20	brd	Controller-In-Charge
ATN	4	10	brd	Attention is asserted
TACS	3	8	brd	Talker
LACS	2	4	brd	Listener
DTAS	1	2	brd	Device Trigger State
DCAS	0	1	brd	Device Clear State

The language header file defines each of the `ibsta` status bits. You can test for an `ibsta` status bit being set using the bitwise and operator (`&` in C/C++). For example, the `ibsta` ERR bit is bit 15 of `ibsta`.

To check for an NI-488.2 error, use the following statement after each NI-488.2 call:

```
if (ibsta & ERR)
    printf("NI-488.2 error encountered");
```

## Error Variable (`iberr`)

If the ERR bit is set in `ibsta`, an NI-488.2 error has occurred. When an error occurs, the error type is specified by `iberr`. To check for an NI-488.2 error, use the following statement after each NI-488.2 call:

```
if (ibsta & ERR)
    printf("NI-488.2 error %d encountered", iberr);
```



**Note** The value in `iberr` is meaningful as an error type only when the ERR bit is set in `ibsta`, indicating that an error has occurred.

For more information about error codes and solutions, refer to Chapter 4, *Debugging Your Application*, or Appendix C, *Error Codes and Solutions*.

## Count Variables (`ibcnt` and `ibcntl`)

The count variables are updated after each read, write, or command function. In Win32 applications, `ibcnt` and `ibcntl` are 32-bit integers. On some systems, like MS-DOS, `ibcnt` is a 16-bit integer, and `ibcntl` is a 32-bit integer. For cross-platform compatibility, all applications should use `ibcntl`. If you are reading data, the count variables indicate the number of bytes read. If you are sending data or commands, the count variables reflect the number of bytes sent.

## Using Interactive Control to Communicate with Devices

---

Before you begin writing your application, you might want to use the Interactive Control utility to communicate with your instruments interactively by typing in commands from the keyboard rather than from an application. You can use the Interactive Control utility to learn to communicate with your instruments using the NI-488.2 API. For specific device communication instructions, refer to the user manual that came with your instrument. For information about using the Interactive Control utility and detailed examples, refer to Chapter 6, *Interactive Control Utility*.

## Programming Models

---

### Applications That Communicate with a Single GPIB Device

This section describes items you should include in your application and provides general program steps with an NI-488.2 example.

#### Items to Include

Include the following items in your application:

- Header files—In a C application, include the header files `windows.h` and `decl-32.h`. The standard Windows header file, `windows.h`, contains definitions used by `decl-32.h`, and `decl-32.h` contains prototypes for the NI-488.2 calls and constants that you can use in your application.
- Error checking—Check for errors after each NI-488.2 call.

- Error handling—Declare and define a function to handle NI-488.2 errors. This function takes the device offline and closes the application. If the function is declared as:

```
void gpiberr (char * msg); /*function prototype*/
```

Then, your application invokes it as follows:

```
if (ibsta & ERR) {
    gpiberr("NI-488.2 error");
}
```

## General Program Steps and Examples

The following steps show you how to use the device-level traditional NI-488.2 calls in your application. The NI-488.2 software includes the source code for an example written in C, `devquery.c`, and the source code for the example written to use direct entry to access `gpib-32.dll`, `dlldevquery.c`. The NI-488.2 software also includes a sample program written in Visual Basic, `devquery.frm`.

### Initialization

#### Step 1. Open a Device

Use `ibdev` to open a device handle. The `ibdev` function requires the following parameters:

- Connect board index (typically 0, for GPIB0).
- Primary address for the GPIB instrument (refer to the instrument user manual or use the `FindLstn` function to dynamically determine the GPIB address of your GPIB device, as described in [Step 2. Determine the GPIB Address of Your Device](#) in the section [Applications That Use Multiple Interfaces or Communicate with Multiple GPIB Devices](#) later in this chapter).
- Secondary address for the GPIB instrument (0 if the GPIB instrument does not use secondary addressing).
- Timeout period (typically set to T10s, which is 10 seconds).
- End-of-transfer mode (typically set to 1 so that EOI is asserted with the last byte of writes).
- EOS detection mode (typically 0 if the GPIB instrument does not use EOS characters).

A successful `ibdev` call returns a device handle, `ud`, that is used for all device-level traditional NI-488.2 calls that communicate with the GPIB instrument.

### Step 2. Clear the Device

Use `ibclr` to clear the device. This resets the device's internal functions to the default state.

## Device Communication

### Step 3. Communicate with the Device

Communicate with the device by sending it the "`*IDN?`" query and then reading back the response. Many devices respond to this query by returning a description of the device. Refer to the documentation that came with your GPIB device to see specific instructions on the proper way to communicate with it.

#### Step 3a.

Use `ibwrt` to send the "`*IDN?`" query command to the device.

#### Step 3b.

Use `ibrdr` to read the response from the device.

Continue communicating with the GPIB device until you are finished.

## Cleanup

### Step 4. Place the Device Offline before Exiting Your Application

Use `ibonl` to put the device handle offline before you exit the application.

## Applications That Use Multiple Interfaces or Communicate with Multiple GPIB Devices

This section describes items you should include in your application and provides general program steps with an NI-488.2 example.

## Items to Include

Include the following items in your application:

- Header files—In a C application, include the header files `windows.h` and `decl-32.h`. The standard Windows header file, `windows.h`, contains definitions used by `decl-32.h`, and `decl-32.h` contains prototypes for the NI-488.2 calls and constants that you can use in your application.
- Error checking—Check for errors after each NI-488.2 call.

- Error handling—Declare and define a function to handle NI-488.2 errors. This function takes the device offline and closes the application. If the function is declared as:

```
void gpiberr (char * msg); /*function prototype*/
```

Then your application invokes it as follows:

```
if (ibsta & ERR) {
    gpiberr("NI-488.2 error");
}
```

## General Program Steps and Examples

The following steps show you how to use the multi-device NI-488.2 calls in your application. The NI-488.2 software includes the source code for an example written in C, `4882query.c`, and the source code for the example written to use direct entry to access the `gpib-32.dll`, `dll4882query.c`. The NI-488.2 software also includes a sample program written in Visual Basic, `query4882.frm`.

### Initialization

#### Step 1. Become Controller-In-Charge (CIC)

Use `SendIFC` to initialize the bus and the GPIB interface so that the GPIB interface is Controller-In-Charge (CIC). The only argument of `SendIFC` is the GPIB interface number, typically 0 for `GPIB0`.

#### Step 2. Determine the GPIB Address of Your Device

Use `FindLstn` to find all the devices attached to the GPIB. The `FindLstn` function requires the following parameters:

- Interface number (typically 0, for `GPIB0`).
- A list of primary addresses, terminated with the `NOADDR` constant.
- A list for reported GPIB addresses of devices found listening on the GPIB.
- Limit, which is the number of the GPIB addresses to report.

Use `FindLstn` to test for the presence of all of the primary addresses that are passed to it. If a device is present at a particular primary address, then the primary address is stored in the GPIB addresses list. Otherwise, all secondary addresses of the given primary address are tested, and the GPIB address of any devices found are stored in the GPIB addresses list. When you have the list of GPIB addresses, you can determine which one corresponds to your instrument and use it for subsequent calls.

Alternately, if you already know your GPIB device's primary and secondary address, you can create an appropriate GPIB address to use in subsequent NI-488.2 calls, as follows: a GPIB address is a 16-bit value that contains the primary address in the low byte and the secondary address in the high byte. If you are not using secondary addressing, the secondary address is 0. For example, if the primary address is 1, then the 16-bit value is 0x01; otherwise, if the primary address is 1 and the secondary address is 67, then the 16-bit value is 0x6701.

### **Step 3. Initialize the Devices**

Use `DevClearList` to clear the devices on the GPIB. The first argument is the GPIB interface number. The second argument is the list of GPIB addresses that were found to be listening as determined in Step 2.

## **Device Communication**

### **Step 4. Communicate with the Devices**

Communicate with the devices by sending them the "`*IDN?`" query and then reading back the responses. Many devices respond to this query by returning a description of the device. Refer to the documentation that came with your GPIB devices to see specific instruction on the proper way to communicate with them.

#### **Step 4a.**

Use `SendList` to send the "`*IDN?`" query command to multiple GPIB devices. The address is the list of GPIB devices to be queried. The buffer that you pass to `SendList` is the command message to the device.

#### **Step 4b.**

Use `Receive` for each device to read the responses from each device.

Continue communicating with the GPIB devices until you are finished.

## **Cleanup**

### **Step 5. Place the Interface Offline before Exiting Your Application**

Use `ibonl` to put the interface offline before you exit the application.

# Language-Specific Programming Instructions

---

The following sections describe how to develop, compile, and link your Win32 NI-488.2 applications using various programming languages.

## Microsoft Visual C/C++ (Version 2.0 or Later)

Before you compile your Win32 C application, make sure that the following lines are included at the beginning of your program:

```
#include <windows.h>
#include "decl-32.h"
```

To compile and link a Win32 console application named `cprog` in a DOS shell, type the following on the command line:

```
cl cprog.c gpib-32.obj
```

## Borland C/C++ (Version 4.0 or Later)

Before you compile your Win32 C application, make sure that the following lines are included at the beginning of your program:

```
#include <windows.h>
#include "decl-32.h"
```

To compile and link a Win32 console application named `cprog` in a DOS shell, type the following on the command line:

```
bcc32 -w32 cprog.c borlandc_gpib-32.obj
```

## Visual Basic (Version 4.0 or Later)

With Visual Basic, you can access the traditional NI-488.2 calls as subroutines, using the BASIC keyword `CALL` followed by the traditional NI-488.2 call name, or you can access them using the `il` set of functions. With some of the NI-488.2 calls (for example `ibrd` and `Receive`), the length of the string buffer is automatically calculated within the actual function or subroutine, which eliminates the need to pass in the length as an extra parameter. For more information about function syntax for Visual Basic, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help** to view the online help.

Before you run your Visual Basic application, include the `niglobal.bas` and `vbib-32.bas` files in your application project file.

## Direct Entry with C

The following sections describe how to use direct entry with C.

### gpib-32.dll Exports

`gpib-32.dll` exports pointers to the global variables and all of the NI-488.2 calls. Pointers to the global variables (`ibsta`, `iberr`, `ibcnt`, and `ibcnt1`) are accessible through these exported variables:

```
int *user_ibsta;
int *user_iberr;
int *user_ibcnt;
long *user_ibcnt1;
```

Except for the functions `ibbna`, `ibfind`, `ibrdf`, and `ibwrtf`, all the NI-488.2 call names are exported from `gpib-32.dll`. Thus, to use direct entry to access a particular function and to get a pointer to the exported function, you just need to call `GetProcAddress` passing the name of the function as a parameter. For more information about the parameters to use when you invoke the function, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help** to view the online help.

The functions `ibbna`, `ibfind`, `ibrdf`, and `ibwrtf` all require an argument that is a name. `ibbna` requires an interface name, `ibfind` requires an interface or device name, and `ibrdf` and `ibwrtf` require a file name. Because Windows NT supports both normal (8-bit) and Unicode (16-bit) characters, `gpib-32.dll` exports both normal and Unicode versions of these functions. Because Windows 98/95 does not support 16-bit wide characters, use only the 8-bit ASCII versions, named `ibbnaA`, `ibfindA`, `ibrdfA`, and `ibwrtfA`. The Unicode versions are named `ibbnaW`, `ibfindW`, `ibrdfW`, and `ibwrtfW`. You can use either the Unicode or ASCII versions of these functions with Windows NT, but only the ASCII versions with Windows 98/95.

In addition to pointers to the status variables and a handle to the loaded `gpib-32.dll`, you must define the direct entry prototypes for the functions you use in your application. The prototypes for each function



exported by `gpib-32.dll` are described in the NI-488.2 online help. The direct entry sample programs illustrate how to use direct entry to access `gpib-32.dll`. For more information about direct entry, refer to the online help that is built into your development environment.

## Directly Accessing the `gpib-32.dll` Exports

Make sure that the following lines are included at the beginning of your application:

```
#ifdef __cplusplus
extern "C" {
#endif

#include <windows.h>
#include "decl-32.h"

#ifdef __cplusplus
}
#endif
```

In your Win32 application, you need to load `gpib-32.dll` before accessing the `gpib-32.dll` exports. The following code fragment shows you how to call the `LoadLibrary` function to load `gpib-32.dll` and check for an error:

```
HINSTANCE Gpib32Lib = NULL;
Gpib32Lib=LoadLibrary("GPIB-32.DLL");
if (Gpib32Lib == NULL) {
    return FALSE;
}
```

The prototypes for each function can be found in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help** to view the online help. For functions that return an integer value, like `ibdev` or `ibwrt`, the pointer to the function needs to be cast as follows:

```
int (_stdcall *Pname)
```

where `*Pname` is the name of the pointer to the function. For functions that do not return a value, like `FindLstn` or `SendList`, the pointer to the function needs to be cast as follows:

```
void (_stdcall *Pname)
```

where \*Pname is the name of the pointer to the function. They are followed by the function's list of parameters as described in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help** to view the online help. Below is an example of how to cast the function pointer and how the parameter list is set up for `ibdev` and `ibonl` functions:

```
int (__stdcall *Pibdev)(int ud, int pad, int sad, int tmo,
int eot, int eos);

int (__stdcall *Pibonl)(int ud, int v);
```

Next, your Win32 application needs to use `GetProcAddress` to get the addresses of the global status variables and functions your application needs. The following code fragment shows you how to get the addresses of the pointers to the status variables and any functions your application needs:

```
/* Pointers to NI-488.2 global status variables */
int *Pibsta;
int *Piberr;
long *Pibcntl;
static int(__stdcall *Pibdev)
    (int ud, int pad, int sad, int tmo, int eot,
    int eos);

static int(__stdcall *Pibonl)
    (int ud, int v);

Pibsta = (int *) GetProcAddress(Gpib32Lib,
    (LPCSTR)"user_ibsta");
Piberr = (int *) GetProcAddress(Gpib32Lib,
    (LPCSTR)"user_iberr");
Pibcntl = (long *) GetProcAddress(Gpib32Lib,
    (LPCSTR)"user_ibcnt");

Pibdev = (int (__stdcall *)
    (int, int, int, int, int, int))
    GetProcAddress(Gpib32Lib, (LPCSTR)"ibdev");
Pibonl = (int (__stdcall *) (int, int))
    GetProcAddress(Gpib32Lib, (LPCSTR)"ibonl");
```

If `GetProcAddress` fails, it returns a NULL pointer. The following code fragment shows you how to verify that none of the calls to `GetProcAddress` failed:

```
if ((Pibsta == NULL) ||
    (Piberr == NULL) ||
```

```

(Pibcntl == NULL) ||
(Pibdev == NULL) ||
(Pibonl == NULL) {

/* Free the GPIB library */
FreeLibrary(Gpib32Lib);
printf("GetProcAddress failed.");
}

```

Your Win32 application needs to dereference the pointer to access either the status variables or function. The following code shows you how to call a function and access the status variable from within your application:

```

dvm = (*Pibdev) (0, 1, 0, T10s, 1, 0);
if (*Pibsta & ERR) {
    printf("Call failed");
}

```

Before exiting your application, you need to free `gpib-32.dll` with the following command:

```
FreeLibrary(Gpib32Lib);
```

For more examples of directly accessing `gpib-32.dll`, refer to the direct entry sample programs `dlldevquery.c` and `dll4882query.c`, installed with the NI-488.2 software. For more information about direct entry, refer to the online help that is built into your development environment.

## Running Existing NI-488.2 Applications

---

### Running Existing Win32 and Win16 NI-488.2 Applications

The NI-488.2 software includes the necessary components to allow existing Win32 and Win16 NI-488.2 applications to run properly.

### Running Existing DOS NI-488.2 Applications Under Windows 98/95

Make sure that no older version of the NI-488.2 DOS device driver is loaded from your `config.sys` file, a file located on the boot drive of your computer. The older NI-488.2 DOS device driver is loaded with the following command line:

```
device=path/gpib.com
```

where *path* is the directory in which you installed the NI-488.2 DOS software (for example, `c:\at-gpib`). Delete this command line to ensure that the older NI-488.2 DOS driver does not load.

To configure the NI-488.2 software to run your existing DOS NI-488.2 applications, complete the following steps after you install the NI-488.2 software and GPIB hardware.

First, start Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. To enable NI-488.2 DOS support under Windows 98/95, follow these steps:

1. Select a GPIB interface under **Devices and Interfaces** in Measurement & Automation Explorer.
2. Select **Tools»Settings»NI-488.2**.
3. Enable or disable DOS support in the **NI-488.2 Software Settings** dialog box.

You can now run your existing DOS NI-488.2 applications.

## Running Existing DOS NI-488.2 Applications under Windows NT

To run DOS NI-488.2 applications, you must enable NI-488.2 DOS support under Windows NT. To enable NI-488.2 DOS support under Windows NT, follow these steps:

1. Open your `config.nt` file, located in the Windows NT system32 directory (for example, `c:\windows\system32`).
2. Find the following lines of code:

```
REM ***To run DOS GPIB applications, uncomment the
REM ***following line
REM device=<path>\doswin16\gpib-nt.com
```

where *<path>* is the directory in which you installed the NI-488.2 software.

3. Remove `REM` from the last line so that it reads  
`device=<path>\doswin16\gpib-nt.com`

To disable DOS support, add `REM` back to the line of code where it was removed.

---

# Debugging Your Application

This chapter describes several ways to debug your application.

## NI Spy

---

The NI Spy utility monitors NI-488.2 API calls made by NI-488.2 applications. It records NI-488.2 API input and output values from all Win32, Win16, and DOS NI-488.2 applications. For more information about NI Spy, refer to its online help by selecting **Help** from the **NI Spy** menu.

To start NI Spy, launch Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **NI Spy** as shown in Figure 4-1.

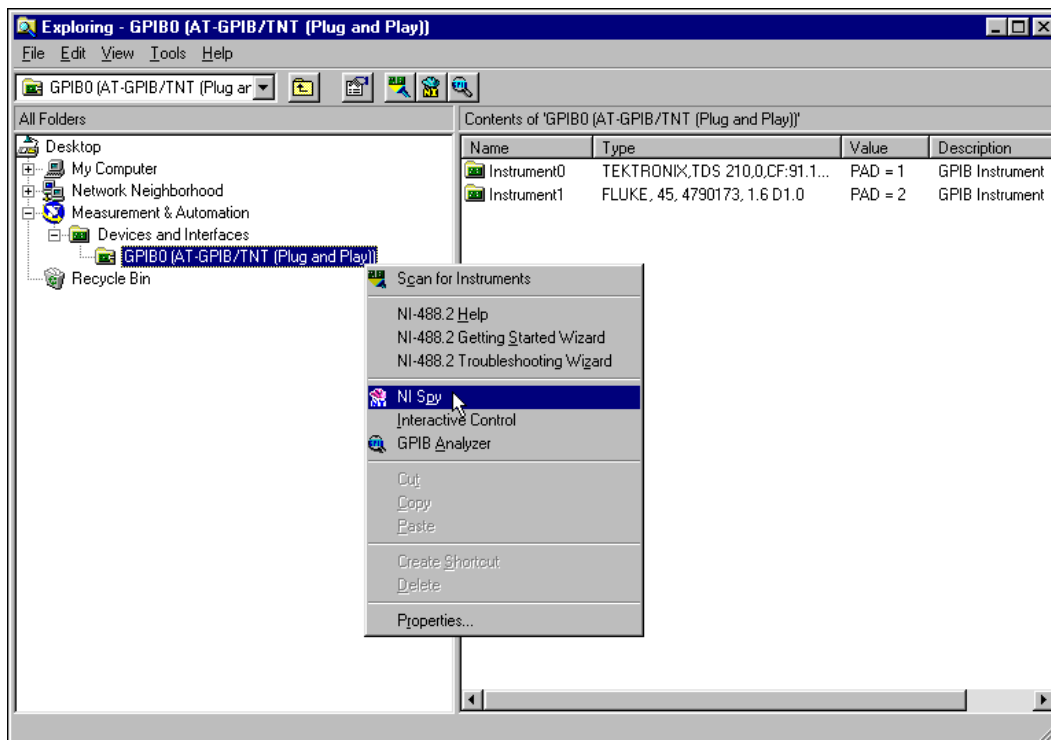


Figure 4-1. Select NI Spy

## Global Status Variables

At the end of each NI-488.2 call, the global status variables (`ibsta`, `iberr`, `ibcnt`, and `ibcnt1`) are updated. If you are developing an NI-488.2 application, you should check for errors after each NI-488.2 call. If a NI-488.2 call failed, the high bit of `ibsta` (the ERR bit) is set. For a failed NI-488.2 call, `iberr` contains a value that defines the error. In some error cases, the value in `ibcnt1` contains even more error information.

You can use NI Spy to determine which NI-488.2 call is failing. Once you know which NI-488.2 call fails, refer to Appendix B, *Status Word Conditions*, and Appendix C, *Error Codes and Solutions*, for help understanding why the NI-488.2 call failed. This information is also available in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**.

Then, select a GPIB interface under **Devices and Interfaces**, right-click, and choose **NI-488.2 Help** to view the online help.

## Existing Applications

---

If the application does not have built-in error detection handling, you can use NI Spy to determine which NI-488.2 call is failing.

To start NI Spy, launch Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **NI Spy**.

After you have an NI Spy capture file, you can use NI Spy to search for failed NI-488.2 calls by searching for calls with the ERR bit set. Once you know which NI-488.2 call fails, refer to Appendix B, *Status Word Conditions*, and Appendix C, *Error Codes and Solutions*, for help understanding why the NI-488.2 call failed. This information is also available in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2» Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **NI-488.2 Help** to view the online help.

## NI-488.2 Error Codes

---

The error variable is meaningful only when the ERR bit in the status variable, `ibsta`, is set. For a detailed description of each error and possible solutions, refer to Appendix C, *Error Codes and Solutions*.

## Configuration Errors

---

Several applications require customized configuration of the NI-488.2 driver. For example, you might want to terminate reads on a special end-of-string character, or you might require secondary addressing. In these cases, you can either reconfigure from your application using the `ibconfig` function or reconfigure using the NI-488.2 Configuration utility.



**Note** National Instruments recommends using `ibconfig` to modify the configuration.

If your application uses `ibconfig`, it works properly regardless of the previous configuration. For more information about using `ibconfig`, refer to the description of `ibconfig` in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **NI-488.2 Help** to view the online help.

## Timing Errors

---

If your application fails, but the same calls issued interactively in the Interactive Control utility are successful, your program might be issuing the NI-488.2 calls too quickly for your device to process and respond to them. This problem can also result in corrupted or incomplete data. This should only be a problem with older, non-standard GPIB devices.

To check if your interactively issued NI-488.2 calls succeed, use the Interactive Control utility. To start the Interactive Control utility, select a GPIB interface under **Devices and Interfaces** in the left window frame of Measurement & Automation Explorer, right-click, and choose **Interactive Control**.

A well-behaved IEEE 488 device does not experience timing errors. If your device is not well-behaved, you can test for and resolve the timing error by single-stepping through your program and inserting finite delays between each NI-488.2 call. One way to do this is to have your device communicate its status whenever possible. Although this method is not possible with many devices, it is usually the best option. Your delays are controlled by the device and your application can adjust itself and work independently on any platform. Other delay mechanisms probably exhibit differing behaviors on different platforms and thus might not eliminate timing errors.



# Communication Errors

---

The following sections describe communication errors you might encounter in your application.

## Repeat Addressing

Devices adhering to the IEEE 488.2 standard should remain in their current state until specific commands are sent across the GPIB to change their state. However, some devices require GPIB addressing before any GPIB activity. Therefore, you might need to configure your NI-488.2 driver to perform repeat addressing if your device does not remain in its currently addressed state. You can either reconfigure from your application using `ibconfig`, or reconfigure using the NI-488.2 Configuration utility.



**Note** National Instruments recommends using `ibconfig` to modify the configuration.

If your application uses `ibconfig`, it works properly regardless of the previous configuration. For more information about `ibconfig`, refer to the description of `ibconfig` in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **NI-488.2 Help** to view the online help.

## Termination Method

You should be aware of the data termination method that your device uses. By default, your NI-488.2 software is configured to send EOI on writes and terminate reads on EOI or a specific byte count. If you send a command string to your device and it does not respond, it might not be recognizing the end of the command. In that case, you need to send a termination message, such as `<CR> <LF>`, after a write command, as follows:

```
ibwrt(dev, "COMMAND\x0A\x0D", 9);
```

## Other Errors

---

If you experience other errors in your application, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. It includes extensive troubleshooting information and the answers to frequently asked questions. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **NI-488.2 Help** to view the online help.

---

# NI Spy Utility

This chapter introduces you to NI Spy, a utility that monitors and records multiple National Instruments APIs (for example, NI-488.2 and VISA).

## Overview

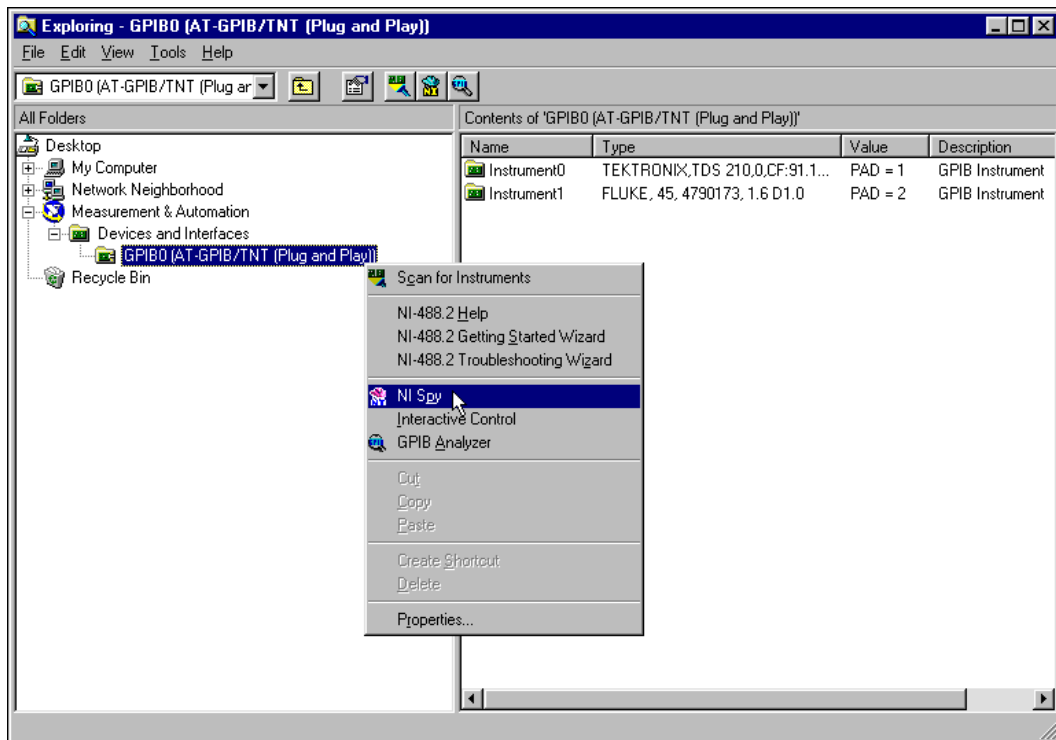
---

NI Spy monitors, records, and displays the NI-488.2 calls made from Win32, Win16, and DOS NI-488.2 applications. It is a useful tool for troubleshooting errors in your application and for verifying that the communication with your GPIB instrument is correct.

## Starting NI Spy

---

To start NI Spy, launch Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI Spy**, as shown in Figure 5-1.



**Figure 5-1.** Select NI Spy

When you launch NI Spy, it displays the main NI Spy window. By default, capture is off. Start capture by clicking on the blue arrow button in the NI Spy toolbar. Then, start the NI-488.2 application that you want to monitor. NI Spy records all NI-488.2 calls made. Figure 5-2 shows the main NI Spy window with several recorded calls.

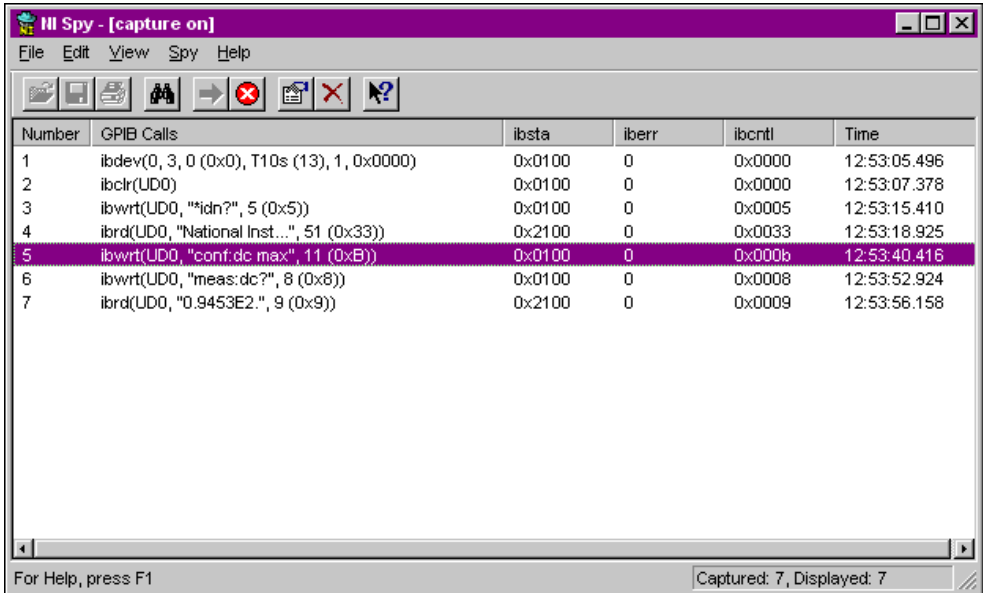


Figure 5-2. NI Spy Application

## Using the NI Spy Online Help

The NI Spy utility has built-in, context-sensitive online help that describes all NI Spy features. To access it, select **Help** from the **NI Spy** menu. You can also access the online help by clicking on the question mark button in the NI Spy toolbar, and then clicking on the area of the screen about which you have a question.

## Locating Errors with NI Spy

All NI-488.2 calls returned with an error are displayed in red within the main NI Spy window.

## Viewing Properties for Recorded Calls

To see the detailed properties of any call recorded in the main NI Spy window, double-click on the call. The **Call Properties** window appears. It contains general, input, output, and buffer information.

## Exiting NI Spy

---

When you exit NI Spy, its current configuration is saved and used to configure NI Spy when you start it again. Unless you save the data captured in NI Spy before you exit, that information is lost.

To save the captured data, click on the red X button on the toolbar and select **File»Save As** to save the data in a `.SPY` file. After you save your data, select **File»Exit** to exit the NI Spy utility.

## Performance Considerations

---

NI Spy can slow down the performance of your NI-488.2 application, and certain configurations of NI Spy have a larger impact on performance than others. For example, configuring NI Spy to record calls to an output file or to use full buffers might have a significant impact on the performance of both your application and your system. For this reason, use NI Spy only while you are debugging your application or in situations where performance is not critical.

---

# Interactive Control Utility

This chapter introduces you to the Interactive Control utility, which lets you communicate with GPIB devices interactively.

## Overview

---

With the Interactive Control utility, you communicate with the GPIB devices through functions you interactively type in at the keyboard. For specific information about communicating with your particular device, refer to the documentation that came with the device. You can use the Interactive Control utility to practice communication with the instrument, troubleshoot problems, and develop your application.

The Interactive Control utility helps you to learn about your instrument and to troubleshoot problems by displaying the following information on your screen after you enter a command:

- Results of the status word (`ibsta`) in hexadecimal notation.
- Mnemonic constant of each bit set in `ibsta`.
- Mnemonic value of the error variable (`iberr`) if an error exists (the ERR bit is set in `ibsta`).
- Count value for each read, write, or command function.
- Data received from your instrument.

## Getting Started with Interactive Control

---

This section shows you how to use the Interactive Control utility to test a sequence of NI-488.2 calls.

To start Interactive Control, launch Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From the Explorer, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and select **Interactive Control**, as shown in Figure 6-1.

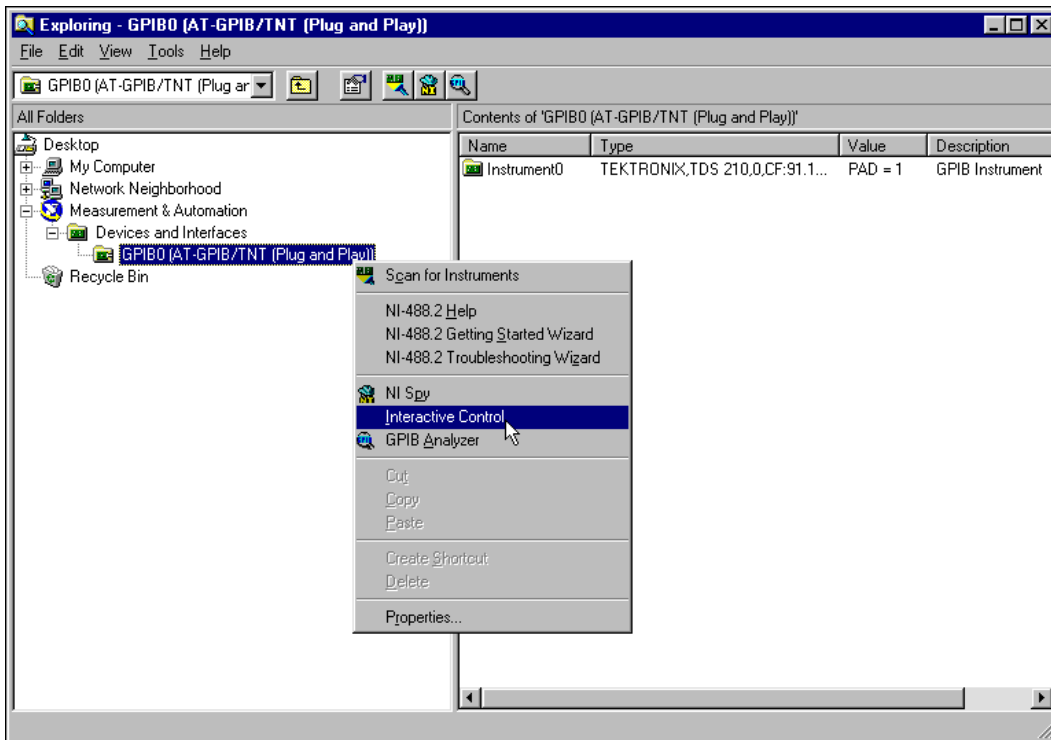


Figure 6-1. Select Interactive Control Utility

When the Interactive Control utility starts, it displays the following banner message:

```

Interactive Control
Copyright 1999 National Instruments Corporation
All rights reserved

Type 'help' for help or 'q' to quit
:

```

First, you must open either an interface handle or device handle to use for further NI-488.2 calls. Use `ibdev` to open a device handle, `ibfind` to open an interface handle, or the `set 488.2` command to switch to a 488.2 prompt. For help on any Interactive Control command, type in `help` followed by the command, for example `help ibdev` or `help set`.



If you want to use device-level calls, open a device handle using `ibdev`. The following example shows you how to use `ibdev` to open a device, assign it to access interface `gpib0`, choose a primary address of 6 with no secondary address, set a timeout of 10 seconds, enable the END message, and disable the EOS mode:

```
:ibdev
  enter board index: 0
  enter primary address: 6
  enter secondary address: 0
  enter timeout: T10s
  enter 'EOI on last byte' flag: 1
  enter end-of-string mode/byte: 0
ud0:
```

If you enter a command and no parameters, you are prompted for the necessary arguments. If you already know the required arguments, you can enter them from the command line, as follows:

```
:ibdev 0 6 0 T10s 1 0
ud0:
```

If you do not know the GPIB primary and secondary address of your GPIB instrument, use the `FindLstn` call as follows:

```
set 488.2 n
```

where *n* is the index of your GPIB interface.

Use `SendIFC` to become controller-in-charge:

```
488.2(0): SendIFC
```

Next, use `FindLstn` to find all the Listeners on the GPIB:

```
488.2(0): FindLstn 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
16,17,18,19,20,21,22,23,24,25,26,27,28,29,30
```

The call returns the number of listeners found in `ibcnt` and then lists the found listeners:

```
count: 1
count is the number of listeners found
Listeners: 5
```

Once you successfully complete `ibdev`, you have a `ud` prompt. The new prompt, `ud0`, represents a device-level handle that you can use for further NI-488.2 calls. To clear the device, use `ibclr`, as follows:

```
ud0: ibclr
[0100] (cml)
```

To write data to the device, use `ibwrt`. Make sure that you refer to the documentation that came with your GPIB instrument for specific command messages.

```
ud0: ibwrt
    enter string: "*IDN?"
[0100] (cml)
count: 5
```

Or, equivalently:

```
ud0: ibwrt "*IDN?"
[0100] (cml)
count: 5
```

To read data from your device, use `ibrd`. The data that is read from the instrument is displayed. For example, to read 29 bytes, enter the following:

```
ud0: ibrd
    enter byte count: 29
[0100] (cml)
count: 29
46 4C 55 4B 45 2C 20 34      FLUKE, 4
35 2C 20 34 37 39 30 31      5, 47901
37 33 2C 20 31 2E 36 20      73, 1.6
44 31 2E 30 0A                D.10.
```

Or, equivalently:

```
ud0: ibrd 29
[0100] (cml)
count: 29
46 4C 55 4B 45 2C 20 34      FLUKE, 4
35 2C 20 34 37 39 30 31      5, 47901
37 33 2C 20 31 2E 36 20      73, 1.6
44 31 2E 30 0A                D.10.
```

When you are finished communicating with the device, make sure you put it offline using the `ibonl` command, as follows:

```
ud0: ibonl 0
[0100] (cml)
:
```

The `ibonl` command properly closes the device handle and the `ud0` prompt is no longer available.

## Interactive Control Syntax

---

The following special rules apply to making calls from the Interactive Control utility:

- The `ud` or `BoardId` parameter is implied by the Interactive Control prompt, therefore it is never included in the call.
- The `count` parameter to functions is unnecessary because buffer lengths are automatically determined by Interactive Control.
- Function return values are handled automatically by Interactive Control. In addition to printing out the return `ibsta` value for the function, it also prints other return values.
- If you do not know what parameters are appropriate to pass to a given function call, type in the function name and press <Enter>. The Interactive Control utility then prompts you for each required parameter.

## Number Syntax

You can enter numbers in either hexadecimal or decimal format.

*Hexadecimal numbers*—You must prefix hexadecimal numbers with `0x`. For example, `ibpad 0x16` sets the primary address to 16 hexadecimal (22 decimal).

*Decimal numbers*—Enter the number only. For example, `ibpad 22` sets the primary address to 22 decimal.

## String Syntax

You can enter strings as an ASCII character sequence, hex bytes, or special symbols.

*ASCII character sequence*—You must enclose the entire sequence in quotation marks.

*Hex byte*—You must use a backslash character and an *x*, followed by the hex value. For example, hex 40 is represented by `\x40`.

*Special symbols*—Some instruments require special termination or end-of-string (EOS) characters that indicate to the device that a transmission has ended. The two most common EOS characters are `\r` and `\n`. `\r` represents a carriage return character and `\n` represents a linefeed character. You can use these special characters to insert the carriage return and linefeed characters into a string, as in `" *IDN?\r\n"`.

## Address Syntax

Some of the NI-488.2 calls have an address or address list parameter. An address is a 16-bit representation of the GPIB device address. The primary address is stored in the low byte and the secondary address, if any, is stored in the high byte. For example, a device at primary address 6 and secondary address 0x67 has an address of 0x6706. A NULL address is represented as 0xffff. An address list is represented by a comma-separated list of addresses, such as 1, 2, 3.

## Interactive Control Commands

---

Tables 6-1 and 6-2 summarize the syntax of the traditional NI-488.2 calls in the Interactive Control utility. Table 6-3 summarizes the syntax of the multi-device NI-488.2 calls in the Interactive Control utility. Table 6-4 summarizes the auxiliary functions that you can use in the Interactive Control utility. For more information about the function parameters, use the online help, available by typing in `help`. If you enter only the function name, the Interactive Control utility prompts you for parameters.

**Table 6-1.** Syntax for Device-Level Traditional NI-488.2 Calls in Interactive Control

Syntax	Description
ibask option	Return configuration information where option is a mnemonic for a configuration parameter
ibbna bname	Change access interface of device where bname is symbolic name of new interface
ibclr	Clear specified device
ibconfig option value	Alter configurable parameters where option is mnemonic for a configuration parameter
ibdev BdIndx pad sad tmo eot eos	Open an unused device; ibdev parameters are BdIndx pad sad tmo eot eos
ibeos v	Change/disable EOS message
ibeot v	Enable/disable END message
ibln pad sad	Check for presence of device on the GPIB at pad, sad
ibloc	Go to local
ibonl v	Place device online or offline
ibpad v	Change primary address
ibpct	Pass control
ibppc v	Parallel poll configure
ibrd count	Read data where count is the bytes to read
ibrda count	Read data asynchronously where count is the bytes to read
ibrdf flname	Read data to file where flname is pathname of file to read
ibrpp	Conduct a parallel poll
ibrsp	Return serial poll byte
ibsad v	Change secondary address
ibstop	Abort asynchronous operation
ibtmo v	Change/disable time limit
ibtrg	Trigger selected device

**Table 6-1.** Syntax for Device-Level Traditional NI-488.2 Calls in Interactive Control (Continued)

Syntax	Description
<code>ibwait mask</code>	Wait for selected event where <code>mask</code> is a hex or decimal integer or a list of mask bit mnemonics, such as <code>ibwait TIMO CMPL</code>
<code>ibwrt wrtbuf</code>	Write data
<code>ibwrta wrtbuf</code>	Write data asynchronously
<code>ibwrtf flname</code>	Write data from a file where <code>flname</code> is pathname of file to write

**Table 6-2.** Syntax for Board-Level Traditional NI-488.2 Calls in Interactive Control

Syntax	Description
<code>ibask option</code>	Return configuration information where <code>option</code> is a mnemonic for a configuration parameter
<code>ibcac v</code>	Become active Controller
<code>ibcmd cmdbuf</code>	Send commands
<code>ibcmda cmdbuf</code>	Send commands asynchronously
<code>ibconfig option value</code>	Alter configurable parameters where <code>option</code> is mnemonic for a configuration parameter
<code>ibdma v</code>	Enable/disable DMA
<code>ibeos v</code>	Change/disable EOS message
<code>ibeot v</code>	Enable/disable END message
<code>ibfind udname</code>	Return unit descriptor where <code>udname</code> is the symbolic name of interface (for example, <code>gpib0</code> )
<code>ibgts v</code>	Go from Active Controller to standby
<code>ibist v</code>	Set/clear <code>ist</code>
<code>iblines</code>	Read the state of all GPIB control lines
<code>ibln pad sad</code>	Check for presence of device on the GPIB at <code>pad</code> , <code>sad</code>
<code>ibloc</code>	Go to local
<code>ibonl v</code>	Place device online or offline
<code>ibpad v</code>	Change primary address

**Table 6-2.** Syntax for Board-Level Traditional NI-488.2 Calls in Interactive Control (Continued)

<b>Syntax</b>	<b>Description</b>
<code>ibppc v</code>	Parallel poll configure
<code>ibrd count</code>	Read data where <code>count</code> is the bytes to read
<code>ibrda count</code>	Read data asynchronously where <code>count</code> is the bytes to read
<code>ibrdf flname</code>	Read data to file where <code>flname</code> is pathname of file to read
<code>ibrpp</code>	Conduct a parallel poll
<code>ibrsc v</code>	Request/release system control
<code>ibrsv v</code>	Request service
<code>ibsad v</code>	Change secondary address
<code>ibsic</code>	Send interface clear
<code>ibsre v</code>	Set/clear remote enable line
<code>ibstop</code>	Abort asynchronous operation
<code>ibtmo v</code>	Change/disable time limit
<code>ibwait mask</code>	Wait for selected event where <code>mask</code> is a hex or decimal integer or a list of mask bit mnemonics, such as <code>ibwait TIMO CMPL</code>
<code>ibwrt wrtbuf</code>	Write data
<code>ibwrta wrtbuf</code>	Write data asynchronously
<code>ibwrtf flname</code>	Write data from a file where <code>flname</code> is pathname of file to write

**Table 6-3.** Syntax for Multi-Device NI-488.2 Calls in Interactive Control

<b>Routine Syntax</b>	<b>Description</b>
<code>AllSpoll addrlist</code>	Serial poll multiple devices
<code>DevClear address</code>	Clear a device
<code>DevClearList addrlist</code>	Clear multiple devices
<code>EnableLocal addrlist</code>	Enable local control
<code>EnableRemote addrlist</code>	Enable remote control
<code>FindLstn padlist limit</code>	Find all Listeners

**Table 6-3.** Syntax for Multi-Device NI-488.2 Calls in Interactive Control (Continued)

<b>Routine Syntax</b>	<b>Description</b>
FindRQS addrlist	Find device asserting SRQ
PassControl address	Pass control to a device
PPoll	Parallel poll devices
PPollConfig address dataline lineSense	Configure device for parallel poll
PPollUnconfig addrlist	Unconfigure device for parallel poll
RcvRespMsg count termination	Receive response message
ReadStatusByte address	Serial poll a device
Receive address count termination	Receive data from a device
ReceiveSetup address	Receive setup
ResetSys addrlist	Reset multiple devices
Send address buffer eotmode	Send data to a device
SendCmds buffer	Send command bytes
SendDataBytes buffer eotmode	Send data bytes
SendIFC	Send interface clear
SendList addrlist buffer eotmode	Send data to multiple devices
SendLLO	Put devices in local lockout
SendSetup addrlist	Send setup
SetRWLS addrlist	Put devices in remote with lockout state
TestSRQ	Test for service request
TestSys addrlist	Cause multiple devices to perform self-tests
Trigger address	Trigger a device
TriggerList addrlist	Trigger multiple devices
WaitSRQ	Wait for service request



**Table 6-4.** Auxiliary Functions in Interactive Control

Function	Description
set udname	Select active device or interface where udname is the symbolic name of the new device or interface (for example, dev1 or gpib0). Call ibfind or ibdev initially to open each device or interface.
set 488.2 v	Start using multi-device NI-488.2 calls for interface v.
help	Display the Interactive Control utility online help.
help option	Display help information about option, where option is any NI-488.2 or auxiliary call (for example, help ibwrt or help set).
!	Repeat previous function.
-	Turn OFF display.
+	Turn ON display.
n * function	Execute function n times where function represents the correct Interactive Control function syntax.
n * !	Execute previous function n times.
\$ filename	Execute indirect file where filename is the pathname of a file that contains Interactive Control functions to be executed.
buffer option	Set type of display used for buffers. Valid options are full, brief, ascii, and off. Default is full.
q	Exit or quit.

## Status Word

In the Interactive Control utility, all NI-488.2 calls (except `ibfind` and `ibdev`) return the status word `ibsta` in two forms: a hex value in square brackets and a list of mnemonics in parentheses. In the following example, the status word is on the second line, showing that the write operation completed successfully:

```
ud0: ibwrt "*IDN?"
[0100] (cml)
count: 5
ud0:
```

For more information about `ibsta`, refer to Chapter 3, [Developing Your NI-488.2 Application](#).

## Error Information

---

If an NI-488.2 call completes with an error, the Interactive Control utility displays the relevant error mnemonic. In the following example, an error condition EBUS has occurred during a data transfer:

```
ud0: ibwrt "*IDN?"  
[8100] (err cml)  
error: EBUS  
count: 1  
ud0:
```

In this example, the addressing command bytes could not be transmitted to the device. This indicates that either the GPIB device is powered off or the GPIB cable is disconnected.

For a detailed list of the error codes and their meanings, refer to Chapter 4, [Debugging Your Application](#).

## Count Information

---

When an I/O function completes, the Interactive Control utility displays the actual number of bytes sent or received, regardless of the existence of an error condition.

If one of the addresses in an address list is invalid, then the error is EARG and the Interactive Control utility displays the index of the invalid address as the count.

The count has a different meaning depending on which NI-488.2 call is made. For the correct interpretation of the count return, refer to the function descriptions in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select an interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI-488.2 Help** to view the online help.

---

# NI-488.2 Programming Techniques

This chapter describes techniques for using some NI-488.2 calls in your application.

For more information about each function or routine, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select an interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI-488.2 Help** to view the online help.

---

## Termination of Data Transfers

GPIB data transfers are terminated either when the GPIB EOI line is asserted with the last byte of a transfer or when a preconfigured end-of-string (EOS) character is transmitted. By default, EOI is asserted with the last byte of writes and the EOS modes are disabled.

You can use the `ibeot` function to enable or disable the end of transmission (EOT) mode. If EOT mode is enabled, the GPIB EOI line is asserted when the last byte of a write is sent out on the GPIB. If it is disabled, the EOI line is *not* asserted with the last byte of a write.

You can use the `ibeos` function to enable, disable, or configure the EOS modes. EOS mode configuration includes the following information:

- A 7-bit or 8-bit EOS byte.
- EOS comparison method—This indicates whether the EOS byte has seven or eight significant bits. For a 7-bit EOS byte, the eighth bit of the EOS byte is ignored.
- EOS write method—If this is enabled, the GPIB EOI line is automatically asserted when the EOS byte is written to the GPIB. If the buffer passed into an `ibwrt` call contains five occurrences of the EOS byte, the EOI line is asserted as each of the five EOS bytes are written to the GPIB. If an `ibwrt` buffer does not contain an occurrence of the

EOS byte, the EOI line is not asserted (unless the EOT mode is enabled, in which case the EOI line is asserted with the last byte of the write).

- EOS read method—If this is enabled, `ibrd`, `ibrda`, and `ibrdf` calls are terminated when the EOS byte is detected on the GPIB, when the GPIB EOI line is asserted, or when the specified count is reached. If the EOS read method is disabled, `ibrd`, `ibrda`, and `ibrdf` calls terminate only when the GPIB EOI line is asserted or the specified count has been read.

You can use the `ibconfig` function to configure the software to indicate whether the GPIB EOI line was asserted when the EOS byte was read in. Use the `IbcEndBitIsNormal` option to configure the software to report only the END bit in `ibsta` when the GPIB EOI line is asserted. By default, END is reported in `ibsta` when either the EOS byte is read in or the EOI line is asserted during a read.

## High-Speed Data Transfers (HS488)

---

National Instruments has designed a high-speed data transfer protocol for IEEE 488 called HS488. This protocol increases performance for GPIB reads and writes up to 8 Mbytes/s, depending on your system.

HS488 is a superset of the IEEE 488 standard; thus, you can mix IEEE 488.1, IEEE 488.2, and HS488 devices in the same system. If HS488 is enabled, the TNT4882C hardware implements high-speed transfers automatically when communicating with HS488 instruments. If you attempt to enable HS488 on a GPIB interface that does not have the TNT4882C hardware, the ECAP error code is returned.

### Enabling HS488

To enable HS488 for your GPIB interface, use the `ibconfig` function (option `IbcHSCableLength`). The value passed to `ibconfig` should specify the number of meters of cable in your GPIB configuration. If you specify a cable length that is much smaller than what you actually use, the transferred data could become corrupted. If you specify a cable length longer than what you actually use, the data is transferred successfully, but more slowly than if you specified the correct cable length.

In addition to using `ibconfig` to configure your GPIB interface for HS488, the Controller-In-Charge must send out GPIB command bytes (interface messages) to configure other devices for HS488 transfers.

If you are using device-level calls, the NI-488.2 software automatically sends the HS488 configuration message to devices. If you enabled the HS488 protocol in the NI-488.2 Configuration utility, the NI-488.2 software sends out the HS488 configuration message when you use `ibdev` to bring a device online. If you call `ibconfig` to change the GPIB cable length, the NI-488.2 software sends out the HS488 message again, the next time you call a device-level function.

If you are using board-level traditional NI-488.2 calls or multi-device NI-488.2 calls and you want to configure devices for high-speed, you must send the HS488 configuration messages using `ibcmd` or `sendCmds`. The HS488 configuration message is made up of two GPIB command bytes. The first byte, the Configure Enable (CFE) message (hex 1F), places all HS488 devices into their configuration mode. Non-HS488 devices should ignore this message. The second byte is a GPIB secondary command that indicates the number of meters of cable in your system. It is called the Configure (CFGn) message. Because HS488 can operate only with cable lengths of 1 to 15 m, only CFGn values of 1 through 15 (hex 61 through 6F) are valid. If the cable length was configured properly in the NI-488.2 Configuration utility, you can determine how many meters of cable are in your system by calling `ibask` (option `IbaHSCableLength`) in your application. For more information about CFE and CFGn messages, refer to the topic *Multiline Interface Messages* in the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select an interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI-488.2 Help** to view the online help.

## System Configuration Effects on HS488

Maximum HS488 data transfer rates can be limited by your host computer and GPIB system setup. For example, when using a PC-compatible computer with PCI bus, the maximum obtainable transfer rate is 8 Mbytes/s, but when using a PC-compatible computer with ISA bus, the maximum transfer rate obtainable is only 2 Mbytes/s. The same IEEE 488 cabling constraints for a 350 ns T1 delay apply to HS488. As you increase the amount of cable in your GPIB configuration, the maximum data transfer rate using HS488 decreases. For example, two HS488 devices connected by two meters of cable can transfer data faster than four HS488 devices connected by 4 m of cable.

## Waiting for GPIB Conditions

---

You can use the `ibwait` function to obtain the current `ibsta` value or to suspend your application until a specified condition occurs on the GPIB. If you use `ibwait` with a parameter of zero, it immediately updates `ibsta` and returns. If you want to use `ibwait` to wait for one or more events to occur, pass a wait mask to the function. The wait mask should always include the TIMO event; otherwise, your application is suspended indefinitely until one of the wait mask events occurs.

## Asynchronous Event Notification in Win32 NI-488.2 Applications

---

Win32 NI-488.2 applications can asynchronously receive event notifications using the `ibnotify` function. This function is useful if you want your application to be notified asynchronously about the occurrence of one or more GPIB events. For example, you might choose to use `ibnotify` if your application only needs to interact with your GPIB device when it is requesting service. After calling `ibnotify`, your application does not need to check the status of your GPIB device. Then, when your GPIB device requests service, the NI-488.2 driver automatically notifies your application that the event has occurred by invoking a callback function. The callback function is registered with the NI-488.2 driver when the `ibnotify` call is made.

### Calling the `ibnotify` Function

`ibnotify` has the following function prototype:

```
ibnotify (
    int ud, // unit descriptor
    int mask, // bit mask of GPIB events
    GpibNotifyCallback_t Callback,
    // callback function
    void * RefData // user-defined reference data
)
```

Both board-level and device-level `ibnotify` calls are supported by the NI-488.2 driver. If you are using device-level calls, you call `ibnotify` with a device handle for `ud` and a `mask` of RQS, CMPL, END, or TIMO. If you are using board-level calls, you call `ibnotify` with a board handle for `ud` and a `mask` of any values except RQS or ERR. The `ibnotify` mask bits are identical to the `ibwait` mask bits. In the example of waiting for your

GPIB device to request service, you might choose to pass `ibnotify` a mask with RQS (for device-level) or SRQI (for board-level).

The callback function that you register with the `ibnotify` call is invoked by the NI-488.2 driver when one or more of the mask bits passed to `ibnotify` is TRUE. The function prototype of the callback is as follows:

```
int __stdcall Callback (
    int ud, // unit descriptor
    int ibsta, // ibsta value
    int iberr, // iberr value
    long ibcntl, // ibcntl value
    void * RefData // user-defined reference data
)
```

The callback function is passed a unit descriptor, the current values of the NI-488.2 global variables, and the user-defined reference data that was passed to the original `ibnotify` call. The NI-488.2 driver interprets the return value for the callback as a mask value that is used to automatically rearm the callback if it is non-zero. For a complete description of `ibnotify`, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select an interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI-488.2 Help** to view the online help.



**Note** The `ibnotify` callback is executed in a separate thread of execution from the rest of your application. If your application will be performing other NI-488.2 operations while it is using `ibnotify`, use the per-thread NI-488.2 globals that are provided by the `ThreadIbsta`, `ThreadIberr`, `ThreadIbcnt`, and `ThreadIbcntl` functions described in the [Writing Multithreaded Win32 NI-488.2 Applications](#) section of this chapter. In addition, if your application needs to share global variables with the callback, use a synchronization primitive (for example, a semaphore) to protect access to any globals. For more information about the use of synchronization primitives, refer to the documentation about using Win32 synchronization objects that came with your development tools.

## ibnotify Programming Example

The following code is an example of how you can use `ibnotify` in your application. Assume that your GPIB device is a multimeter that you program it to acquire a reading by sending "SEND DATA". The multimeter requests service when it has a reading ready, and each reading is a floating point value.

In this example, globals are shared by the Callback thread and the main thread, and the access of the globals is not protected by synchronization. In this case, synchronization of access to these globals is not necessary because of the way they are used in the application: only a single thread is writing the global values and that thread only adds information (increases the count or adds another reading to the array of floats).

```
int __stdcall MyCallback (int ud, int LocalIbsta, int LocalIberr,
                        long LocalIbcntl, void *RefData);

int ReadingsTaken = 0;
float Readings[1000];
BOOL DeviceError = FALSE;
char expectedResponse = 0x43;

int main()
{
    int ud;

    // Assign a unique identifier to the device and store it in the
    // variable ud. ibdev opens an available device and assigns it to
    // access GPIB0 with a primary address of 1, a secondary address of 0,
    // a timeout of 10 seconds, the END message enabled, and the EOS mode
    // disabled. If ud is less than zero, then print an error message
    // that the call failed and exit the program.
    ud = ibdev          (0, // connect board
                       1,  // primary address of GPIB device
                       0,  // secondary address of GPIB device
                       T10s, // 10 second I/O timeout
                       1,  // EOT mode turned on
                       0); // EOS mode disabled

    if (ud < 0) {
        printf ("ibdev failed.\n");
        return 0;
    }

    // Issue a request to the device to send the data. If the ERR bit
    // is set in ibsta, then print an error message that the call failed
    // and exit the program.
    ibwrt (ud, "SEND DATA", 9L);
    if (ibsta & ERR) {
        printf ("unable to write to device.\n");
    }
}
```



```

        return 0;
    }

    // set up the asynchronous event notification on RQS
    ibnotify (ud, RQS, MyCallback, NULL);
    if (ibsta & ERR) {
        printf ("ibnotify call failed.\n");
        return 0;
    }

    while ((ReadingsTaken < 1000) && !(DeviceError)) {
        // Your application does useful work here. For example, it
        // might process the device readings or do any other useful work.
    }

    // disable notification
    ibnotify (ud, 0, NULL, NULL);

    // Call the ibonl function to disable the hardware and software.
    ibonl (ud, 0);
    return 1;
}

int __stdcall MyCallback (int LocalUd, int LocalIbsta, int LocalIberr,
    long LocalIbcntl, void *RefData)
{
    char SpollByte;
    char ReadBuffer[40];

    // If the ERR bit is set in LocalIbsta, then print an error
    // message and return.
    if (LocalIbsta & ERR) {
        printf (" GPIB error %d has occurred. No more callbacks.\n",
            LocalIberr);
        DeviceError = TRUE;
        return 0;
    }
    // Read the serial poll byte from the device. If the ERR bit is set
    // in ibsta, then print an error message and return.
    LocalIbsta = ibrsp (LocalUd, &SpollByte);
    if (LocalIbsta & ERR) {
        printf ("ibrsp failed. No more callbacks.\n");
        DeviceError = TRUE;
    }
}

```

```

        return 0;
    }

    // If the returned status byte equals the expected response, then
    // the device has valid data to send; otherwise it has a fault
    // condition to report.
    if (SpollByte != expectedResponse) {
        printf("Device returned invalid response. Status byte = 0x%x\n",
            SpollByte);
        DeviceError = TRUE;
        return 0;
    }

    // Read the data from the device. If the ERR bit is set in ibsta,
    // then print an error message and return.
    LocalIbsta = ibrd (LocalUd, ReadBuffer, 40L);
    if (LocalIbsta & ERR) {
        printf ("ibrd failed. No more callbacks.\n");
        DeviceError = TRUE;
        return 0;
    }

    // The string returned by ibrd is a binary string whose length is
    // specified by the byte count in ibcntl. However, many GPIB
    // instruments return ASCII data strings and this example makes this
    // assumption. Because of this, it is possible to add a NULL
    // character to the end of the data received and use the printf()
    // function to display the ASCII data. The following code
    // illustrates that.
    ReadBuffer[ibcntl] = '\0';

    // Convert the data into a numeric value.
    sscanf (ReadBuffer, "%f", &Readings[ReadingsTaken]);

    // Display the data.
    printf("Reading : %f\n", Readings[ReadingsTaken]);

    ReadingsTaken += 1;
    if (ReadingsTaken >= 1000) {
        return 0;
    }

    else {

```

```

// Issue a request to the device to send the data and rearm
// callback on RQS.
LocalIbsta = ibwrt (LocalUd, "SEND DATA", 9L);
if (LocalIbsta & ERR) {
    printf ("ibwrt failed. No more callbacks.\n");
    DeviceError = TRUE;
    return 0;
}

else {
    return RQS;
}
}

```

## Writing Multithreaded Win32 NI-488.2 Applications

---

If you are writing a multithreaded NI-488.2 application and you plan to make all of your NI-488.2 calls from a single thread, you can safely continue to use the traditional NI-488.2 global variables (`ibsta`, `iberr`, `ibcnt`, `ibcnt1`). The NI-488.2 global variables are defined on a per-process basis, so each process accesses its own copy of the NI-488.2 globals.

If you are writing a multithreaded NI-488.2 application and you plan to make NI-488.2 calls from more than a single thread, you cannot safely continue to use the traditional NI-488.2 global variables without some form of synchronization (for example, a semaphore). To understand why, refer to the following example.

Assume that a process has two separate threads that make NI-488.2 calls, thread #1 and thread #2. Just as thread #1 is about to examine one of the NI-488.2 globals, it gets preempted and thread #2 is allowed to run. Thread #2 proceeds to make several NI-488.2 calls that automatically update the NI-488.2 globals. Later, when thread #1 is allowed to run, the NI-488.2 global that it is ready to examine is no longer in a known state and its value is no longer reliable.

The previous example illustrates a well-known multithreading problem. It is unsafe to access process-global variables from multiple threads of execution. You can avoid this problem in two ways:

- Use synchronization to protect access to process-global variables.
- Do not use process-global variables.

If you choose to implement the synchronization solution, you must ensure that the code making NI-488.2 calls and examining the NI-488.2 globals modified by a NI-488.2 call is protected by a synchronization primitive. For example, each thread might acquire a semaphore before making a NI-488.2 call and then release the semaphore after examining the NI-488.2 globals modified by the call. For more information about the use of synchronization primitives, refer to the documentation about using Win32 synchronization objects that came with your development tools.

If you choose not to use process-global variables, you can access per-thread copies of the NI-488.2 global variables using a special set of NI-488.2 calls. Whenever a thread makes a NI-488.2 call, the driver keeps a private copy of the NI-488.2 globals for that thread. The driver keeps a separate private copy for each thread. The following code shows the set of functions you can use to access these per-thread NI-488.2 global variables:

```
int ThreadIbsta(); // return thread-specific ibsta
int ThreadIberr(); // return thread-specific iberr
int ThreadIbcnt(); // return thread-specific ibcnt
long ThreadIbcntl(); // return thread-specific ibcntl
```

In your application, instead of accessing the per-process NI-488.2 globals, substitute a call to get the corresponding per-thread NI-488.2 global. For example, the following line of code,

```
if (ibsta & ERR)
```

could be replaced by,

```
if (ThreadIbsta() & ERR)
```

A quick way to convert your application to use per-thread NI-488.2 globals is to add the following `#define` lines at the top of your C file:

```
#define ibsta ThreadIbsta()
#define iberr ThreadIberr()
#define ibcnt ThreadIbcnt()
#define ibcntl ThreadIbcntl()
```



**Note** If you are using `ibnotify` in your application (see the [Asynchronous Event Notification in Win32 NI-488.2 Applications](#) section of this chapter), the `ibnotify` callback is executed in a separate thread that is created by the NI-488.2 driver. Therefore, if your application makes NI-488.2 calls from the `ibnotify` callback function and makes NI-488.2 calls from other places, you must use the `ThreadIbsta`, `ThreadIberr`, `ThreadIbcnt`, and `ThreadIbcntl` functions described in this section, instead of the per-process NI-488.2 globals.

## Device-Level Calls and Bus Management

---

The device-level traditional NI-488.2 calls are designed to perform all of the GPIB management for your application. However, the NI-488.2 driver can handle bus management only when the GPIB interface is CIC (Controller-In-Charge). Only the CIC is able to send command bytes to the devices on the bus to perform device addressing or other bus management activities.

Use one of the following methods to make your GPIB interface the CIC:

- If your GPIB interface is configured as the System Controller (default), it automatically makes itself the CIC by asserting the IFC line the first time you make a device-level call.
- If your setup includes more than one Controller, or if your GPIB interface is not configured as the System Controller, use the CIC Protocol method. To use the protocol, issue the `ibconfig` function (option `IBCCICPROT`) or use the NI-488.2 Configuration utility to activate the CIC protocol. If the interface is not CIC, and you make a device-level call with the CIC protocol enabled, the following sequence occurs:
  1. The GPIB interface asserts the SRQ line.
  2. The current CIC serial polls the interface.
  3. The interface returns a response byte of hex 42.
  4. The current CIC passes control to the GPIB interface.

If the current CIC does not pass control, the NI-488.2 driver returns the ECIC error code to your application. This error can occur if the current CIC does not understand the CIC protocol. If this happens, you could send a device-specific command requesting control for the GPIB interface. Then, use a board-level `ibwait` command to wait for CIC.

## Talker/Listener Applications

---

Although designed for Controller-In-Charge applications, you can also use the NI-488.2 software in most non-Controller situations. These situations are known as Talker/Listener applications because the interface is not the GPIB Controller.

A Talker/Listener application typically uses `ibwait` with a mask of 0 to monitor the status of the interface. Then, based on the status bits set in `ibsta`, the application takes whatever action is appropriate. For example,

the application could monitor the status bits TACS (Talker Active State) and LACS (Listener Active State) to determine when to send data to or receive data from the Controller. The application could also monitor the DCAS (Device Clear Active State) and DTAS (Device Trigger Active State) bits to determine if the Controller has sent the device clear (DCL or SDC) or trigger (GET) messages to the interface. If the application detects a device clear from the Controller, it might reset the internal state of message buffers. If it detects a trigger message from the Controller, the application might begin an operation, such as taking a voltage reading if the application is actually acting as a voltmeter.

## Serial Polling

---

You can use serial polling to obtain specific information from GPIB devices when they request service. When the GPIB SRQ line is asserted, it signals the Controller that a service request is pending. The Controller must then determine which device asserted the SRQ line and respond accordingly. The most common method for SRQ detection and servicing is the serial poll. This section describes how to set up your application to detect and respond to service requests from GPIB devices.

### Service Requests from IEEE 488 Devices

IEEE 488 devices request service from the GPIB Controller by asserting the GPIB SRQ line. When the Controller acknowledges the SRQ, it serial polls each open device on the bus to determine which device requested service. Any device requesting service returns a status byte with bit 6 set and then unasserts the SRQ line. Devices not requesting service return a status byte with bit 6 cleared. Manufacturers of IEEE 488 devices use lower order bits to communicate the reason for the service request or to summarize the state of the device.

### Service Requests from IEEE 488.2 Devices

The IEEE 488.2 standard refined the bit assignments in the status byte. In addition to setting bit 6 when requesting service, IEEE 488.2 devices also use two other bits to specify their status. Bit 4, the Message Available bit (MAV), is set when the device is ready to send previously queried data. Bit 5, the Event Status bit (ESB), is set if one or more of the enabled IEEE 488.2 events occurs. These events include power-on, user request, command error, execution error, device dependent error, query error, request control, and operation complete. The device can assert SRQ when ESB or MAV are set, or when a manufacturer-defined condition occurs.

## Automatic Serial Polling

You can enable automatic serial polling if you want your application to conduct a serial poll automatically when the SRQ line is asserted. The autopolling procedure occurs as follows:

1. To enable autopolling, use the configuration function, `ibconfig`, with option `IbcAUTO POLL`, or the NI-488.2 Configuration utility. (Autopolling is enabled by default.)
2. When the SRQ line is asserted, the driver automatically serial polls the open devices.
3. Each positive serial poll response (bit 6 or hex 40 is set) is stored in a queue associated with the device that sent it. The RQS bit of the device status word, `ibsta`, is set.
4. The polling continues until SRQ is unasserted or an error condition is detected.
5. To empty the queue, use the `ibrsp` function. `ibrsp` returns the first queued response. Other responses are read in first-in-first-out (FIFO) fashion. If the RQS bit of the status word is not set when `ibrsp` is called, a serial poll is conducted and returns the response received. Empty the queue as soon as an automatic serial poll occurs, because responses might be discarded if the queue is full.
6. If the RQS bit of the status word is still set after `ibrsp` is called, the response byte queue contains at least one more response byte. If this happens, continue to call `ibrsp` until RQS is cleared.

## Stuck SRQ State

If autopolling is enabled and the GPIB interface detects an SRQ, the driver serial polls all open devices connected to that interface. The serial poll continues until either SRQ unasserts or all the devices have been polled.

If no device responds positively to the serial poll, or if SRQ remains in effect because of a faulty instrument or cable, a *stuck SRQ* state is in effect. If this happens during an `ibwait` for RQS, the driver reports the ESRQ error. If the stuck SRQ state happens, no further polls are attempted until an `ibwait` for RQS is made. When `ibwait` is issued, the stuck SRQ state is terminated and the driver attempts a new set of serial polls.

## Autopolling and Interrupts

If autopolling and interrupts are both enabled, the NI-488.2 software can perform autopolling after any device-level NI-488.2 call provided that no GPIB I/O is currently in progress. In this case, an automatic serial poll can

occur even when your application is not making any calls to the NI-488.2 software. Autopolling can also occur when a device-level `ibwait` for RQS is in progress. Autopolling is not allowed when an application calls a board-level traditional or multi-device NI-488.2 call, or the stuck SRQ (ESRQ) condition occurs.



**Note** The NI-488.2 software for Windows NT does not function properly if interrupts are disabled.

## SRQ and Serial Polling with Device-Level Traditional NI-488.2 Calls

You can use the device-level traditional NI-488.2 call `ibrsp` to conduct a serial poll. `ibrsp` conducts a single serial poll and returns the serial poll response byte to the application. If automatic serial polling is enabled, the application can use `ibwait` to suspend program execution until RQS appears in the status word, `ibsta`. The program can then call `ibrsp` to obtain the serial poll response byte.

The following example shows you how to use the `ibwait` and `ibrsp` functions in a typical SRQ servicing situation when automatic serial polling is enabled:

```
#include "decl-32.h"
char GetSerialPollResponse ( int DeviceHandle )
{

char SerialPollResponse = 0;
ibwait ( DeviceHandle, TIMO | RQS );
if ( ibsta & RQS ) {
printf ( "Device asserted SRQ.\n" );
/* Use ibrsp to retrieve the serial poll response. */
ibrsp ( DeviceHandle, &SerialPollResponse );
}
return SerialPollResponse;
}
```



## SRQ and Serial Polling with Multi-Device NI-488.2 Calls

The NI-488.2 software includes a set of multi-device NI-488.2 calls that you can use to conduct SRQ servicing and serial polling. Routines pertinent to SRQ servicing and serial polling are `AllSpoll`, `ReadStatusByte`, `FindRQS`, `TestSRQ`, and `WaitSRQ`. Following are descriptions of each of the routines:

- `AllSpoll` can serial poll multiple devices with a single call. It places the status bytes from each polled instrument into a predefined array. Then, you must check the RQS bit of each status byte to determine whether that device requested service.
- `ReadStatusByte` is similar to `AllSpoll`, except that it only serial polls a single device. It is also similar to the device-level NI-488.2 `ibrsp` function.
- `FindRQS` serial polls a list of devices until it finds a device that is requesting service or until it has polled all of the devices on the list. The routine returns the index and status byte value of the device requesting service.
- `TestSRQ` determines whether the SRQ line is asserted and returns to the program immediately.
- `WaitSRQ` is similar to `TestSRQ`, except that `WaitSRQ` suspends the application until either SRQ is asserted or the timeout period is exceeded.

The following examples use these calls to detect SRQ and then determine which device requested service. In these examples, three devices are present on the GPIB at addresses 3, 4, and 5, and the GPIB interface is designated as bus index 0. The first example uses `FindRQS` to determine which device is requesting service, and the second example uses `AllSpoll` to serial poll all three devices. Both examples use `WaitSRQ` to wait for the GPIB SRQ line to be asserted.

### Example 1: Using FindRQS

This example shows you how to use `FindRQS` to find the first device that is requesting service:

```
void GetASerialPollResponse ( char *DevicePad,
                             char *DeviceResponse )
{
    char SerialPollResponse = 0;
    int WaitResult;
    Addr4882_t Addrlist[4] = {3,4,5,NOADDR};
    WaitSRQ (0, &WaitResult);
}
```

```

    if (WaitResult) {
        printf ("SRQ is asserted.\n");
        FindRQS ( 0, AddrList, &SerialPollResponse );
        if (!(ibsta & ERR)) {
            printf ("Device at pad %x returned byte
                %x.\n", AddrList[ibcnt],(int)
                SerialPollResponse);
            *DevicePad = AddrList[ibcnt];
            *DeviceResponse = SerialPollResponse;
        }
    }
    return;
}

```

## Example 2: Using AllSpoll

This example shows you how to use AllSpoll to serial poll three devices with a single call:

```

void GetAllSerialPollResponses ( Addr4882_t AddrList[],
short ResponseList[] )
{
    int WaitResult;
    WaitSRQ (0, &WaitResult);
    if ( WaitResult ) {
        printf ( "SRQ is asserted.\n" );
        AllSpoll ( 0, AddrList, ResponseList );
        if (!(ibsta & ERR)) {
            for (i = 0; AddrList[i] != NOADDR; i++) {
                printf ("Device at pad %x returned byte
                    %x.\n", AddrList[i], ResponseList[i] );
            }
        }
    }
    return;
}

```

# Parallel Polling

---

Although parallel polling is not widely used, it is a useful method for obtaining the status of more than one device at the same time. The advantage of parallel polling is that a single parallel poll can easily check up to eight individual devices at once. In comparison, eight separate serial polls would be required to check eight devices for their serial poll response bytes. The value of the individual status bit (*ist*) determines the parallel poll response.

## Implementing a Parallel Poll

You can implement parallel polling with either the traditional or multi-device NI-488.2 calls. If you use multi-device NI-488.2 calls to execute parallel polls, you do not need extensive knowledge of the parallel polling messages. However, you should use the traditional NI-488.2 calls for parallel polling when the GPIB interface is not the Controller, and the interface must configure itself for a parallel poll and set its own individual status bit (*ist*).

## Parallel Polling with Traditional NI-488.2 Calls

Complete the following steps to implement parallel polling using traditional NI-488.2 calls. Each step contains example code.

1. Configure the device for parallel polling using the `ibppc` function, unless the device can configure itself for parallel polling.

`ibppc` requires an 8-bit value to designate the data line number, the *ist* sense, and whether the function configures the device for the parallel poll. The bit pattern is as follows:

```
0 1 1 E S D2 D1 D0
```

E is 1 to disable parallel polling and 0 to enable parallel polling for that particular device.

S is 1 if the device is to assert the assigned data line when *ist* is 1, and 0 if the device is to assert the assigned data line when *ist* is 0.

D2 through D0 determine the number of the assigned data line. The physical line number is the binary line number plus one. For example, DIO3 has a binary bit pattern of 010.

The following example code configures a device for parallel polling using traditional NI-488.2 calls. The device asserts DIO7 if its *ist* is 0.

In this example, the `ibdev` command opens a device that has a primary address of 3, has no secondary address, has a timeout of 3 s, asserts EOI with the last byte of a write operation, and has EOS characters disabled.

The following call configures the device to respond to the poll on DIO7 and to assert the line in the case when its `ist` is 0. Pass the binary bit pattern, 0110 0110 or hex 66, to `ibppc`.

```
#include "decl-32.h"
char ppr;
dev = ibdev(0,3,0,T3s,1,0);
ibppc(dev, 0x66);
```

If the GPIB interface configures itself for a parallel poll, you should still use the `ibppc` function. Pass the interface index or an interface unit descriptor value as the first argument in `ibppc`. Also, if the individual status bit (`ist`) of the interface needs to be changed, use the `ibist` function.

In the following example, the GPIB interface is to configure itself to participate in a parallel poll. It asserts DIO5 when `ist` is 1 if a parallel poll is conducted.

```
ibppc(0, 0x6C);
ibist(0, 1);
```

2. Conduct the parallel poll using `ibrpp` and check the response for a certain value. The following example code performs the parallel poll and compares the response to hex 10, which corresponds to DIO5. If that bit is set, the `ist` of the device is 1.

```
ibrpp(dev, &ppr);
if (ppr & 0x10) printf("ist = 1\n");
```

3. Unconfigure the device for parallel polling with `ibppc`. Notice that any value having the parallel poll disable bit set (bit 4) in the bit pattern disables the configuration, so you can use any value between hex 70 and 7E.

```
ibppc(dev, 0x70);
```

## Parallel Polling with Multi-Device NI-488.2 Calls

Complete the following steps to implement parallel polling the using multi-device NI-488.2 calls. Each step contains example code.

1. Configure the device for parallel polling using the `PPollConfig` routine, unless the device can configure itself for parallel polling. The following example configures a device at address 3 to assert data line 5 (DIO5) when its `ist` value is 1.

```
#include "decl-32.h"
char response;
Addr4882_t AddressList[2];
/* The following command clears the GPIB. */
SendIFC(0);
/* The value of sense is compared with the ist bit
   of the device and determines whether the data
   line is asserted.*/
PPollConfig(0,3,5,1);
```

2. Conduct the parallel poll using `PPoll`, store the response, and check the response for a certain value. In the following example, because DIO5 is asserted by the device if `ist` is 1, the program checks bit 4 (hex 10) in the response to determine the value of `ist`.

```
PPoll(0, &response);
/* If response has bit 4 (hex 10) set, the ist bit
   of the device at that time is equal to 1. If
   it does not appear, the ist bit is equal to 0.
   Check the bit in the following statement. */
if (response & 0x10) {
    printf("The ist equals 1.\n");
}
else {
    printf("The ist equals 0.\n");
}
```

3. Unconfigure the device for parallel polling using `PPollUnconfig`, as shown in the following example. In this example, the `NOADDR` constant must appear at the end of the array to signal the end of the address list. If `NOADDR` is the only value in the array, all devices receive the parallel poll disable message.

```
AddressList[0] = 3;
AddressList[1] = NOADDR;
PPollUnconfig(0, AddressList);
```

---

## GPIB Basics

The ANSI/IEEE Standard 488.1-1987, also known as General Purpose Interface Bus (GPIB), describes a standard interface for communication between instruments and controllers from various vendors. It contains information about electrical, mechanical, and functional specifications. GPIB is a digital, 8-bit parallel communications interface with data transfer rates of 1 Mbyte/s and higher, using a three-wire handshake. The bus supports one System Controller, usually a computer, and up to 14 additional instruments. The ANSI/IEEE Standard 488.2-1992 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands.

### Talkers, Listeners, and Controllers

---

GPIB devices can be Talkers, Listeners, or Controllers. A Talker sends out data messages. Listeners receive data messages. The Controller, usually a computer, manages the flow of information on the bus. It defines the communication links and sends GPIB commands to devices.

Some devices are capable of playing more than one role. A digital voltmeter, for example, can be a Talker and a Listener. If your system has a National Instruments GPIB interface and software installed, it can function as a Talker, Listener, and Controller.

### Controller-In-Charge and System Controller

---

You can have multiple Controllers on the GPIB, but only one Controller at a time can be the active Controller, or Controller-In-Charge (CIC). The CIC can be either active or inactive (standby). Control can pass from the current CIC to an idle Controller, but only the System Controller, usually a GPIB interface, can make itself the CIC.

## GPIB Addressing

---

All GPIB devices and interfaces must be assigned a unique GPIB address. A GPIB address is made up of two parts: a primary address and an optional secondary address.

The primary address is a number in the range 0 to 30. The Controller uses this address to form a talk or listen address that is sent over the GPIB when communicating with a device.

A talk address is formed by setting bit 6, the TA (Talk Active) bit of the GPIB address. A listen address is formed by setting bit 5, the LA (Listen Active) bit of the GPIB address. For example, if a device is at address 1, the Controller sends hex 41 (address 1 with bit 6 set) to make the device a Talker. Because the Controller is usually at primary address 0, it sends hex 20 (address 0 with bit 5 set) to make itself a Listener. Figure A-1 shows the configuration of the GPIB address bits.

Bit Position	7	6	5	4	3	2	1	0
Meaning	0	TA	LA	GPIB Primary Address (range 0–30)				

**Figure A-1.** GPIB Address Bits

With some devices, you can use secondary addressing. A secondary address is a number in the range hex 60 to hex 7E. When you use secondary addressing, the Controller sends the primary talk or listen address of the device followed by the secondary address of the device.

## Sending Messages across the GPIB

---

Devices on the bus communicate by sending messages. Signals and lines transfer these messages across the GPIB interface, which consists of 16 signal lines and 8 ground return (shield drain) lines. The 16 signal lines are discussed in the following sections.

### Data Lines

Eight data lines, DIO1 through DIO8, carry both data and command messages.

## Handshake Lines

Three hardware handshake lines asynchronously control the transfer of message bytes between devices. This process is a three-wire interlocked handshake, and it guarantees that devices send and receive message bytes on the data lines without transmission error. Table A-1 summarizes the GPIB handshake lines.

**Table A-1.** GPIB Handshake Lines

Line	Description
NRFD (not ready for data)	Listening device is ready/not ready to receive a message byte. Also used by the Talker to signal high-speed GPIB transfers.
NDAC (not data accepted)	Listening device has/has not accepted a message byte.
DAV (data valid)	Talking device indicates signals on data lines are stable (valid) data.

## Interface Management Lines

Five hardware lines manage the flow of information across the bus. Table A-2 summarizes the GPIB interface management lines.

**Table A-2.** GPIB Interface Management Lines

Line	Description
ATN (attention)	Controller drives ATN true when it sends commands and false when it sends data messages.
IFC (interface clear)	System Controller drives the IFC line to initialize the bus and make itself CIC.
REN (remote enable)	System Controller drives the REN line to place devices in remote or local program mode.
SRQ (service request)	Any device can drive the SRQ line to asynchronously request service from the Controller.
EOI (end or identify)	Talker uses the EOI line to mark the end of a data message. Controller uses the EOI line when it conducts a parallel poll.



## Status Word Conditions

This appendix gives a detailed description of the conditions reported in the status word, `ibsta`.

For information about how to use `ibsta` in your application program, refer to the NI-488.2 online help, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start» Programs» National Instruments NI-488.2» Explore GPIB**. Then, select an interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI-488.2 Help** to view the online help.

Each bit in `ibsta` can be set for device calls (dev), board calls (brd), or both (dev, brd). The following table shows the status word layout.

Mnemonic	Bit Position	Hex Value	Type	Description
ERR	15	8000	dev, brd	NI-488.2 error
TIMO	14	4000	dev, brd	Time limit exceeded
END	13	2000	dev, brd	END or EOS detected
SRQI	12	1000	brd	SRQ interrupt received
RQS	11	800	dev	Device requesting service
CMPL	8	100	dev, brd	I/O completed
LOK	7	80	brd	Lockout State
REM	6	40	brd	Remote State
CIC	5	20	brd	Controller-In-Charge
ATN	4	10	brd	Attention is asserted
TACS	3	8	brd	Talker
LACS	2	4	brd	Listener
DTAS	1	2	brd	Device Trigger State
DCAS	0	1	brd	Device Clear State

## ERR (dev, brd)

---

ERR is set in the status word following any call that results in an error. You can determine the particular error by examining the error variable `iberr`. Appendix C, *Error Codes and Solutions*, describes error codes that are recorded in `iberr` along with possible solutions. ERR is cleared following any call that does not result in an error.

## TIMO (dev, brd)

---

TIMO indicates that the timeout period has expired. TIMO is set in the status word following any synchronous I/O functions (for example, `ibcmd`, `ibrd`, `ibwrt`, `Receive`, `Send`, and `SendCmds`) if the timeout period expires before the I/O operation has completed. TIMO is also set in the status word following an `ibwait` or `ibnotify` call if the TIMO bit is set in the `mask` parameter and the timeout period expires before any other specified `mask` bit condition occurs. TIMO is cleared in all other circumstances.

## END (dev, brd)

---

END indicates either that the GPIB EOI line has been asserted or that the EOS byte has been received, if the software is configured to terminate a read on an EOS byte. If the GPIB interface is performing a shadow handshake as a result of the `ibgts` function, any other function can return a status word with the END bit set if the END condition occurs before or during that call. END is cleared when any I/O operation is initiated.

Some applications might need to know the exact I/O read termination mode of a read operation—EOI by itself, the EOS character by itself, or EOI plus the EOS character. You can use the `ibconfig` function (option `IbcEndBitIsNormal`) to enable a mode in which the END bit is set only when EOI is asserted. In this mode, if the I/O operation completes because of the EOS character by itself, END is not set. The application should check the last byte of the received buffer to see if it is the EOS character.

## SRQI (brd)

---

SRQI indicates that a GPIB device is requesting service. SRQI is set whenever the GPIB interface is CIC, the GPIB SRQ line is asserted, and the automatic serial poll capability is disabled. SRQI is cleared either when the GPIB interface ceases to be the CIC or when the GPIB SRQ line is unasserted.

## RQS (dev)

---

RQS appears in the status word only after a device-level call and indicates that the device is requesting service. RQS is set whenever one or more positive serial poll response bytes have been received from the device. A positive serial poll response byte always has bit 6 asserted. Automatic serial polling must be enabled (it is enabled by default) for RQS to automatically appear in `ibsta`. You can also wait for a device to request service regardless of the state of automatic serial polling by calling `ibwait` with a mask that contains RQS. Do not issue an `ibwait` call on RQS for a device that does not respond to serial polls. Use `ibrsp` to acquire the serial poll response byte that was received. RQS is cleared when all of the stored serial poll response bytes have been reported to you through the `ibrsp` function.

## CMPL (dev, brd)

---

CMPL indicates the condition of I/O operations. It is set whenever an I/O operation is complete. CMPL is cleared while the I/O operation is in progress.

## LOK (brd)

---

LOK indicates whether the interface is in a lockout state. While LOK is set, the `EnableLocal` routine or `ibloc` function is inoperative for that interface. LOK is set whenever the GPIB interface detects that the Local Lockout (LLO) message has been sent either by the GPIB interface or by another Controller. LOK is cleared when the System Controller unasserts the Remote Enable (REN) GPIB line.

## REM (brd)

---

REM indicates whether the interface is in the remote state. REM is set whenever the Remote Enable (REN) GPIB line is asserted and the GPIB interface detects that its listen address has been sent either by the GPIB interface or by another Controller. REM is cleared in the following situations:

- When REN becomes unasserted.
- When the GPIB interface as a Listener detects that the Go to Local (GTL) command has been sent either by the GPIB interface or by another Controller.
- When the `ibloc` function is called while the LOK bit is cleared in the status word.

## CIC (brd)

---

CIC indicates whether the GPIB interface is the Controller-In-Charge. CIC is set when the `SendIFC` routine or `ibsic` function is executed either while the GPIB interface is System Controller or when another Controller passes control to the GPIB interface. CIC is cleared either when the GPIB interface detects Interface Clear (IFC) from the System Controller or when the GPIB interface passes control to another device.

## ATN (brd)

---

ATN indicates the state of the GPIB Attention (ATN) line. ATN is set whenever the GPIB ATN line is asserted, and it is cleared when the ATN line is unasserted.

## TACS (brd)

---

TACS indicates whether the GPIB interface is addressed as a Talker. TACS is set whenever the GPIB interface detects that its talk address (and secondary address, if enabled) has been sent either by the GPIB interface itself or by another Controller. TACS is cleared whenever the GPIB interface detects the Untalk (UNT) command, its own listen address, a talk address other than its own talk address, or Interface Clear (IFC).

## LACS (brd)

---

LACS indicates whether the GPIB interface is addressed as a Listener. LACS is set whenever the GPIB interface detects that its listen address (and secondary address, if enabled) has been sent either by the GPIB interface itself or by another Controller. LACS is also set whenever the GPIB interface shadow handshakes as a result of the `ibgts` function. LACS is cleared whenever the GPIB interface detects the Unlisten (UNL) command, its own talk address, Interface Clear (IFC), or that the `ibgts` function has been called without shadow handshake.

## DTAS (brd)

---

DTAS indicates whether the GPIB interface has detected a device trigger command. DTAS is set whenever the GPIB interface, as a Listener, detects that the Group Execute Trigger (GET) command has been sent by another Controller. DTAS is cleared on any call immediately following an `ibwait` call, if the DTAS bit is set in the `ibwait` mask parameter.

## DCAS (brd)

---

DCAS indicates whether the GPIB interface has detected a device clear command. DCAS is set whenever the GPIB interface detects that the Device Clear (DCL) command has been sent by another Controller, or whenever the GPIB interface as a Listener detects that the Selected Device Clear (SDC) command has been sent by another Controller.

If you use the `ibwait` or `ibnotify` function to wait for DCAS and the wait is completed, DCAS is cleared from `ibsta` after the next NI-488.2 call. The same is true of reads and writes. If you call a read or write function such as `ibwrt` or `Send`, and DCAS is set in `ibsta`, the I/O operation is aborted. DCAS is cleared from `ibsta` after the next NI-488.2 call.



---

# Error Codes and Solutions

This appendix lists a description of each error, some conditions under which it might occur, and possible solutions.

The following table lists the GPIB error codes.

<b>Error Mnemonic</b>	<b>iberr Value</b>	<b>Meaning</b>
EDVR	0	System error
ECIC	1	Function requires GPIB interface to be CIC
ENOL	2	No Listeners on the GPIB
EADR	3	GPIB interface not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB interface not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB interface
EDMA	8	DMA error
EOIP	10	Asynchronous I/O in progress
ECAP	11	No capability for operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial poll status byte queue overflow
ESRQ	16	SRQ stuck in ON position
ETAB	20	Table problem

## EDVR (0)

---

EDVR is returned when the interface or device name passed to `ibfind`, or the interface index passed to `ibdev`, cannot be accessed. The global variable `ibcntl` contains an error code. This error occurs when you try to access an interface or device that is not installed or configured properly.

EDVR is also returned if an invalid unit descriptor is passed to any traditional NI-488.2 call.

## Solutions

Possible solutions for this error are as follows:

- Use `ibdev` to open a device without specifying its symbolic name.
- Use only device or interface names that are configured in the NI-488.2 Configuration utility as parameters to the `ibfind` function.
- Use the NI-488.2 Troubleshooting Wizard to ensure that each interface you want to access is working properly. To start the NI-488.2 Troubleshooting Wizard, first open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select **Measurement & Automation** in the left window frame, and choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.
- Use the unit descriptor returned from `ibdev` or `ibfind` as the first parameter in subsequent traditional NI-488.2 calls. Examine the variable before the failing function to make sure its value has not been corrupted.
- Refer to the NI-488.2 online help for the NI-488.2 software, available through Measurement & Automation Explorer. To start Measurement & Automation Explorer, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and select **NI-488.2 Help** to view the online help.

## ECIC (1)

---

ECIC is returned when one of the following board functions or routines is called while the interface is not CIC:

- Any device-level traditional NI-488.2 calls that affect the GPIB.
- Any board-level traditional NI-488.2 calls that issue GPIB command bytes: `ibcmd`, `ibcmda`, `ibln`, and `ibrpp`.

- `ibcac` and `ibgts`.
- Any NI-488.2 multi-device calls that issue GPIB command bytes: `SendCmds`, `PPoll`, `Send`, and `Receive`.

## Solutions

Possible solutions for this error are as follows:

- Use `ibsic` or `SendIFC` to make the GPIB interface become CIC on the GPIB.
- Use `ibrsc 1` to make sure your GPIB interface is configured as System Controller.
- In multiple CIC situations, always be certain that the CIC bit appears in the status word `ibsta` before attempting these calls. If it does not appear, you can perform an `ibwait` (for CIC) call to delay further processing until control is passed to the interface.

## ENOL (2)

---

ENOL usually occurs when a write operation is attempted with no Listeners addressed. For a device write, ENOL indicates that the GPIB address configured for that device in the software does not match the GPIB address of any device connected to the bus, that the GPIB cable is not connected to the device, or that the device is not powered on.

ENOL can occur in situations where the GPIB interface is not the CIC and the Controller asserts ATN before the write call in progress has ended.

## Solutions

Possible solutions for this error are as follows:

- Make sure that the GPIB address of your device matches the GPIB address of the device to which you want to write data.
- Use the appropriate hex code in `ibcmd` to address your device.
- Check your cable connections and make sure at least two-thirds of your devices are powered on.
- Call `ibpad` (or `ibsad`, if necessary) to match the configured address to the device switch settings.
- Reduce the write byte count to that which is expected by the Controller.



## EADR (3)

---

EADR occurs when the GPIB interface is CIC and is not properly addressing itself before read and write functions. This error is usually associated with board-level functions.

EADR is also returned by the function `ibgts` when the shadow-handshake feature is requested and the GPIB ATN line is already unasserted. In this case, the shadow handshake is not possible and the error is returned to notify you of that fact.

### Solutions

Possible solutions for this error are as follows:

- Make sure that the GPIB interface is addressed correctly before calling `ibrdr`, `ibwrt`, `RcvRespMsg`, or `SendDataBytes`.
- Avoid calling `ibgts` except immediately after an `ibcmd` call. (`ibcmd` causes ATN to be asserted.)

## EARG (4)

---

EARG results when an invalid argument is passed to a function call. The following are some examples:

- `ibtmo` called with a value not in the range 0 through 17.
- `ibeos` called with meaningless bits set in the high byte of the second parameter.
- `ibpad` or `ibsad` called with invalid addresses.
- `ibppc` called with invalid parallel poll configurations.
- A board-level traditional NI-488.2 call made with a valid device descriptor, or a device-level traditional NI-488.2 call made with a board descriptor.
- A multi-device NI-488.2 call made with an invalid address.
- `PPollConfig` called with an invalid data line or sense bit.

### Solutions

Possible solutions for this error are as follows:

- Make sure that the parameters passed to the NI-488.2 call are valid.
- Do not use a device descriptor in a board function or vice-versa.

## ESAC (5)

---

ESAC results when `ibsic`, `ibsre`, `SendIFC`, or `EnableRemote` is called when the GPIB interface does not have System Controller capability.

### Solutions

Give the GPIB interface System Controller capability by calling `ibrsc 1` or by using the NI-488.2 Configuration utility to configure that capability into the software.

## EABO (6)

---

EABO indicates that an I/O operation has been canceled, usually due to a timeout condition. Other causes are calling `ibstop` or receiving the Device Clear message from the CIC while performing an I/O operation. Frequently, the I/O is not progressing (the Listener is not continuing to handshake or the Talker has stopped talking), or the byte count in the call which timed out was more than the other device was expecting.

### Solutions

Possible solutions for this error are as follows:

- Use the correct byte count in input functions or have the Talker use the END message to signify the end of the transfer.
- Lengthen the timeout period for the I/O operation using `ibtmo`.
- Make sure that you have configured your device to send data before you request data.

## ENEB (7)

---

ENEB occurs when no GPIB interface exists at the I/O address specified in the configuration program. This problem happens when the interface is not physically plugged into the system, the I/O address specified during configuration does not match the actual interface setting, or there is a system conflict with the base I/O address.

### Solutions

Make sure there is a GPIB interface in your computer that is properly configured both in hardware and software using a valid base I/O address by running the NI-488.2 Troubleshooting Wizard. To run the NI-488.2

Troubleshooting Wizard, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Then, select **Measurement & Automation** in the left window frame, and choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

## EDMA (8)

---

EDMA occurs if a system DMA error is encountered when the NI-488.2 software attempts to transfer data over the GPIB using DMA.

### Solutions

Possible solutions for this error are as follows:

- You can correct the EDMA problem in the hardware by using the NI-488.2 Configuration utility to reconfigure the hardware to not use a DMA resource.
- You can correct the EDMA problem in the software by using `ibdma` to disable DMA.

## EOIP (10)

---

EOIP occurs when an asynchronous I/O operation has not finished before some other call is made. During asynchronous I/O, you can only use `ibstop`, `ibnotify`, `ibwait`, and `ibonl` or perform other non-GPIB operations. If any other call is attempted, EOIP is returned.

### Solutions

Resynchronize the driver and the application before making any further NI-488.2 calls. Resynchronization is accomplished by using one of the following functions:

<code>ibnotify</code>	If the <code>ibsta</code> value passed to the <code>ibnotify</code> callback contains CMPL, the driver and application are resynchronized.
<code>ibwait</code>	If the returned <code>ibsta</code> contains CMPL, the driver and application are resynchronized.
<code>ibstop</code>	The I/O is canceled; the driver and application are resynchronized.

`ibonl` The I/O is canceled and the interface is reset; the driver and application are resynchronized.

## ECAP (11)

---

ECAP results when your GPIB interface lacks the ability to carry out an operation or when a particular capability has been disabled in the software and a call is made that requires the capability.

### Solutions

Check the validity of the call, or make sure your GPIB interface and the driver both have the needed capability.

## EFSO (12)

---

EFSO results when an `ibrdf` or `ibwrtf` call encounters a problem performing a file operation. Specifically, this error indicates that the function is unable to open, create, seek, write, or close the file being accessed. The specific operating system error code for this condition is contained in `ibcntl`.

### Solutions

Possible solutions for this error are as follows:

- Make sure the filename, path, and drive that you specified are correct.
- Make sure that the access mode of the file is correct.
- Make sure there is enough room on the disk to hold the file.

## EBUS (14)

---

EBUS results when certain GPIB bus errors occur during device functions. All device functions send command bytes to perform addressing and other bus management. Devices are expected to accept these command bytes within the time limit specified by the default configuration or the `ibtmo` function. EBUS results if a timeout occurred while sending these command bytes.

## Solutions

Possible solutions for this error are as follows:

- Verify that the instrument is operating correctly.
- Check for loose or faulty cabling or several powered-off instruments on the GPIB.
- If the timeout period is too short for the driver to send command bytes, increase the timeout period.

## ESTB (15)

---

ESTB is reported only by the `ibrsp` function. ESTB indicates that one or more serial poll status bytes received from automatic serial polls have been discarded because of a lack of storage space. Several older status bytes are available; however, the oldest is being returned by the `ibrsp` call.

## Solutions

Possible solutions for this error are as follows:

- Call `ibrsp` more frequently to empty the queue.
- Disable autopolling with the `ibconfig` function (option `IbcAUTOPOLL`) or the NI-488.2 Configuration utility. To start the NI-488.2 Configuration utility, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **Properties**.

## ESRQ (16)

---

ESRQ can only be returned by a device-level `ibwait` call with `RQS` set in the mask. ESRQ indicates that a wait for `RQS` is not possible because the GPIB `SRQ` line is stuck on. This situation can be caused by the following events:

- Usually, a device unknown to the software is asserting `SRQ`. Because the software does not know of this device, it can never serial poll the device and unassert `SRQ`.
- A GPIB bus tester or similar equipment might be forcing the `SRQ` line to be asserted.
- A cable problem might exist involving the `SRQ` line.

Although the occurrence of ESRQ warns you of a definite GPIB problem, it does not affect GPIB operations, except that you cannot depend on the `ibsta` RQS bit while the condition lasts.

## Solutions

Check to see if other devices not used by your application are asserting SRQ. Disconnect them from the GPIB if necessary.

## ETAB (20)

---

ETAB occurs only during the `FindLstn` and `FindRQS` functions. ETAB indicates that there was some problem with a table used by these functions:

- In the case of `FindLstn`, ETAB means that the given table did not have enough room to hold all the addresses of the Listeners found.
- In the case of `FindRQS`, ETAB means that none of the devices in the given table were requesting service.

## Solutions

In the case of `FindLstn`, increase the size of result arrays. In the case of `FindRQS`, check to see if other devices not used by your application are asserting SRQ. Disconnect them from the GPIB if necessary.

---

# Windows 98/95: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems and answers some common questions for Windows 98/95 users.

---

## Troubleshooting EDVR Error Conditions

---

In some cases, NI-488.2 calls may return with the ERR bit set in `ibsta` and the value EDVR in `iberr`. The value stored in `ibcntl` is useful in troubleshooting the error condition.

### EDVR Error Condition with `ibcntl` Set to 0xE028002C (–534249428)

If a call is made with an interface number that is within the range of allowed interface numbers (typically 0 to 3), but which has not been assigned to a GPIB interface, an EDVR error condition occurs with `ibcntl` set to 0xE028002C. You can assign an interface number to a GPIB interface by configuring the NI-488.2 software and selecting an interface name. For information about how to configure the NI-488.2 software, refer to the online help in the NI-488.2 Configuration utility. To start the NI-488.2 Configuration utility, open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. Select a GPIB interface under **Devices and Interfaces** in the left window frame, right-click, and choose **Properties**.

### EDVR Error Condition with `ibcntl` Set to 0xE0140025 (–535560155)

If a call is made with an interface number that is not within the range of allowed interface numbers (typically 0 to 3), an EDVR error condition occurs with `ibcntl` set to 0xE0140025.

## EDVR Error Condition with `ibcntl` Set to `0xE0140035` (–535560139)

If a call is made with a device name that is not listed in the logical device templates that are part of the NI-488.2 Configuration utility, an EDVR error condition occurs with `ibcntl` set to `0xE0140035`.

## EDVR Error Condition with `ibcntl` Set to `0xE0320029` (–533594071) or `0xE1050029` (–519765975)

If a call is made with an interface number that is assigned to a GPIB interface that is unusable because of a resource conflict, an EDVR error condition occurs with `ibcntl` set to `0xE0320029` or `0xE1050029`. This error is also returned if you remove a PCMCIA-GPIB or PCMCIA-GPIB+ while the driver is accessing it or if you try to access a PCMCIA-GPIB when 32-bit PCMCIA drivers are not enabled.

## EDVR Error Condition with `ibcntl` Set to `0xE0140004` (–535560188)

This error might occur if the GPIB interface has not been correctly installed and detected by Windows. For details on how to install the GPIB hardware, refer to the online GPIB hardware guide, which is available by choosing **View Documentation** from the *NI-488.2 for Windows* CD autorun screen.

## EDVR Error Condition with `ibcntl` set to `0xE1030043` (–519897021)

This error occurs if you have enabled DOS NI-488.2 support and attempted to run an existing NI-488.2 DOS application that was compiled with an older, unsupported DOS language interface.

# Troubleshooting Device Manager Problems

---

If you are having trouble with your GPIB interface, use the Windows Device Manager to troubleshoot your problems. To start the Device Manager, double-click on the **System** icon under **Start»Settings»Control Panel**. In the **System Properties** box that appears, select the **Device Manager** tab and click on the **View devices by type** button at the top of the tab.

Check to see if the interface listing in the Device Manager appears with an exclamation point (!) or X by it. If it does, click on the interface listing, and then click on the **Properties** button to view the **General** tab for the interface. In the **Device Status** section, look for the status description and



status code number. Locate the error code in the following list to find out why your GPIB interface is not working properly:

- Code 9: Windows had a problem reading information from the GPIB interface. This problem can occur if you are using an older revision of the AT-GPIB/TNT+ or AT-GPIB/TNT (PnP) interface. Contact National Instruments to upgrade your GPIB interface.
- Code 22: The GPIB interface is disabled. To enable the GPIB interface, check the appropriate configuration checkbox in the **Device Usage** section of the **General** tab.
- Code 24: The GPIB interface is not present, or the Device Manager is unaware that the GPIB interface is present. To solve this problem, select the interface in the Device Manager, and click on the **Remove** button. Next, click on the **Refresh** button. At this point, the system rescans the installed hardware, and the GPIB interface should show up without any problems. If the problem persists, contact National Instruments.
- Code 27: Windows was unable to assign the GPIB interface any resources. To solve this problem, free up system resources by disabling other unnecessary hardware so that enough resources are available for the GPIB interface.

## Common Questions

---

**What do I do if my GPIB hardware is listed in the Windows Device Manager with a circled X or an exclamation point (!) overlaid on it?**

Refer to the *Troubleshooting Device Manager Problems* section of this appendix for specific information about what might cause this problem. If you have already completed the troubleshooting steps, contact National Instruments.

**How can I determine which type of GPIB hardware I have installed?**

Open Measurement & Automation Explorer by selecting **Start» Programs»National Instruments NI-488.2»Explore GPIB**. The Measurement & Automation Explorer lists your installed GPIB hardware under **Devices and Interfaces**.

**How can I determine which version of the NI-488.2 software I have installed?**

Open Measurement & Automation Explorer by selecting **Start» Programs»National Instruments NI-488.2»Explore GPIB**. Select

**Measurement & Automation** in the left window frame, then choose **Help»About Measurement & Automation Explorer**. In the **About Measurement & Automation Explorer** dialog box, click on the **System Info** button. Then, select the **Software** tab. The **Name, Type, Value,** and **Description** columns appear, describing the NI-488.2 software attributes.

### **How do I get started?**

Use the NI-488.2 Getting Started Wizard to get started with the NI-488.2 software. Open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**, select **Measurement & Automation** in the left window frame, then choose **Help»Getting Started»NI-488.2 Getting Started Wizard** and follow the instructions.

### **How do I troubleshoot problems?**

Run the NI-488.2 Troubleshooting Wizard. Open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

### **How many GPIB interfaces can I configure for use with my NI-488.2 software?**

You can configure the NI-488.2 software to communicate with up to 100 GPIB interfaces.

### **How many devices can I configure for use with my NI-488.2 software?**

The NI-488.2 software provides a total of 1,024 logical devices for applications to use. The default number of devices is 32. The maximum number of physical devices you should connect to a single GPIB interface is 14, or fewer, depending on your system configuration.

### **Are interrupts and DMA required for the NI-488.2 software?**

Neither interrupts nor DMA are required under Window 98/95, unless you are using a PCMCIA-GPIB or GPIB hardware with Analyzer capability, in which case at least one interrupt level is required.

**How can I determine if my GPIB hardware and NI-488.2 software are installed properly?**

Run the NI-488.2 Troubleshooting Wizard to make sure that your hardware and software are correctly installed.

Open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

**When should I use the Interactive Control utility?**

You can use the Interactive Control utility to test and verify instrument communication, troubleshoot problems, and develop your application. For more information, refer to Chapter 6, *Interactive Control Utility*.

**How do I use an NI-488.2 language interface?**

For information about using NI-488.2 language interfaces, refer to Chapter 3, *Developing Your NI-488.2 Application*.

**What do I need to know to communicate properly with my GPIB instrument?**

Refer to the documentation that came from the instrument manufacturer. The command sequences you use are dependent on the specific instrument. The documentation for each instrument should include the GPIB commands you need to communicate with it. In most cases, device-level traditional NI-488.2 calls are sufficient for communicating with instruments. For more information, refer to Chapter 3, *Developing Your NI-488.2 Application*.

**How can I easily start communicating with my GPIB instrument?**

Use the NI-488.2 Communicator for simple instrument communication.

To start the NI-488.2 Communicator, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, right-click on the GPIB instrument in the right window frame. Then, choose **Communicate With Instrument**. For more information about communicating with your GPIB instrument, refer to the section *Communicate with a GPIB Instrument* in Chapter 2, *Measurement & Automation Explorer*.

### **What can I do to check for errors in my NI-488.2 application?**

Examine the value of `ibsta` after each NI-488.2 call. If a call fails, the ERR bit of `ibsta` is set and an error code is stored in `iberr`. For more information about global status variables, refer to Chapter 4, *Debugging Your Application*.

### **What information should I have before I call National Instruments?**

When you call National Instruments, you should have the results of the NI-488.2 Troubleshooting Wizard. Run the NI-488.2 Troubleshooting Wizard. Open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

---

# Windows NT: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems and answers some common questions for Windows NT users.

## Using Windows NT Diagnostic Tools

---

There are many reasons why the NI-488.2 driver might not load. If the software is not properly installed or if there is a conflict between the GPIB hardware and the other hardware in the system, the NI-488.2 driver fails to start. Two Windows NT utilities are useful in determining the source of the problem: the Devices applet in the Control Panel, and the Event Viewer. The following sections describe information available through each utility.

### Examining NT Devices to Verify the Installation

To verify whether the devices are installed correctly (that is, that the devices are started), select **Start»Settings»Control Panel** and double-click on the **Devices** icon.

This utility lists all of the devices detected by Windows NT. Each device has a status associated with it. If the NI-488.2 driver is installed correctly, the following lines appear in the list of NT devices:

<u>Device</u>	<u>Status</u>	<u>Started</u>
GPIB Board Class Driver	Started	Automatic
GPIB Device Class Driver	Started	Automatic

You should also see one or more lines similar to the following:

<u>Device</u>	<u>Status</u>	<u>Started</u>
GPIB Port Driver (AT-GPIB)	****	System
GPIB Port Driver (PCI-GPIB)	****	System

The GPIB Board Class Driver and the GPIB Device Class Driver should have a status of **started**. If not, refer to the next section, [Examining the NT System Log Using the Event Viewer](#).

At least one of the GPIB Port Drivers listed by the Devices applet should have a status of **started**. If not, refer to the next section, *Examining the NT System Log Using the Event Viewer*.

If the GPIB Class Driver lines are not present or at least one GPIB Port Driver line is not present, the NI-488.2 software is not installed properly. You should reinstall the NI-488.2 software from the *NI-488.2 for Windows* CD.

## Examining the NT System Log Using the Event Viewer

Windows NT maintains a system log. If the NI-488.2 driver is unable to start, it records entries in the system log explaining why it failed to start. To examine the system log through the Event Viewer utility, select **Start»Programs»Administrative Tools»Event Viewer**.

Events that might appear in the system log include the following:

- The system cannot locate the device file for one or more of the devices that make up the NI-488.2 driver and an event is logged that **The system cannot find the file specified**. In this case, the NI-488.2 software is not installed properly. You should reinstall the NI-488.2 software from the *NI-488.2 for Windows* CD.
- A conflict exists between the GPIB hardware and the other hardware in the system. If this is the case, an event is logged that indicates the nature of the resource conflict. To correct this conflict, run the Troubleshooting Wizard and follow its instructions for resolving resource conflicts. To run the Troubleshooting Wizard, first open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

## Common Questions

---

**How can I determine which type of GPIB hardware I have installed?**

Run the NI-488.2 Troubleshooting Wizard. It lists all the known GPIB hardware.

Open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

### **How can I determine which version of the NI-488.2 software I have installed?**

Open Measurement & Automation Explorer by selecting **Start» Programs»National Instruments NI-488.2»Explore GPIB**. Select **Measurement & Automation** in the left window frame, then choose **Help»About Measurement & Automation Explorer**. In the **About Measurement & Automation Explorer** dialog box, click on the **System Info** button. Then select the **Software** tab. The **Name**, **Type**, **Value**, and **Description** columns appear, describing the NI-488.2 software attributes.

### **How many GPIB interfaces can I configure for use with my NI-488.2 software?**

You can configure the NI-488.2 software to communicate with up to four GPIB interfaces.

### **How many devices can I configure for use with my NI-488.2 software?**

The NI-488.2 software provides a total of 100 logical devices for applications to use. The default number of devices is 32.

### **Are interrupts and DMA required with the NI-488.2 software?**

Interrupts are required, but DMA is not.

### **How can I determine if my GPIB hardware and NI-488.2 software are installed properly?**

Run the NI-488.2 Troubleshooting Wizard to make sure that your hardware and software are correctly installed.

Open Measurement & Automation Explorer by selecting **Start» Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

### **When should I use the Interactive Control utility?**

You can use the Interactive Control utility to test and verify instrument communication, troubleshoot problems, and develop your application. For more information, refer to Chapter 6, *Interactive Control Utility*.

### **How do I use an NI-488.2 language interface?**

For information about using NI-488.2 language interfaces, refer to Chapter 3, *Developing Your NI-488.2 Application*.

### **What do I need to know to communicate properly with my GPIB instrument?**

Refer to the documentation that came from the instrument manufacturer. The command sequences you use are dependent on the specific instrument. The documentation for each instrument should include the GPIB commands you need to communicate with it. In most cases, device-level traditional NI-488.2 calls are sufficient for communicating with instruments. For more information, refer to Chapter 3, *Developing Your NI-488.2 Application*.

### **How can I easily start communicating with my GPIB instrument?**

Use the NI-488.2 Communicator for simple instrument communication.

To start the NI-488.2 Communicator, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, right-click on the GPIB instrument in the right window frame. Then, choose **Communicate With Instrument**.

### **What can I do to check for errors in my NI-488.2 application?**

Examine the value of `ibsta` after each NI-488.2 call. If a call fails, the ERR bit of `ibsta` is set and an error code is stored in `iberr`. For more information about global status variables, refer to Chapter 4, *Debugging Your Application*.

### **What information should I have before I call National Instruments?**

When you call National Instruments, you should have the results of the NI-488.2 Troubleshooting Wizard. Open Measurement & Automation Explorer by selecting **Start»Programs»National Instruments NI-488.2»Explore GPIB**. From there, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.



---

# Technical Support Resources

National Instruments offers technical support through electronic, fax, and telephone systems. The electronic services include our Web site, an FTP site, and a fax-on-demand system. If you have a hardware or software problem, please first try the electronic support systems. If the information available on these systems does not answer your questions, contact one of our technical support centers, which are staffed by applications engineers, for support by telephone and fax. To comment on the documentation supplied with our products, send e-mail to [techpubs@natinst.com](mailto:techpubs@natinst.com).

## Web Site

The InstrumentationWeb address is <http://www.natinst.com>.

From this Web site you can connect to our Web sites around the world (<http://www.natinst.com/niglobal/>) and access technical support (<http://www.natinst.com/support/>).

## FTP Site

To access our FTP site, log in to our Internet host, [ftp.natinst.com](ftp://ftp.natinst.com), as `anonymous` and use your e-mail address, such as `yourname@anywhere.com`, as your password. The support files and documents are located in the `\support` directories.

## Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents in English on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

## E-Mail Support

You can submit technical support questions to the applications engineering team through e-mail at [support@natinst.com](mailto:support@natinst.com). Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

## Telephone and Fax Support

National Instruments has branch offices all over the world. Use the following list to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 284 5011	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Québec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	0 1 48 14 24 24	0 1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
India	91805275406	91805275410
Israel	03 6120092	03 6120095
Italy	02 413091	02 4139215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico (D.F.)	5 280 7625	5 520 3282
Mexico (Monterrey)	8 357 7695	8 365 8543
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain (Madrid)	91 640 0085	91 640 0533
Spain (Barcelona)	93 582 0251	93 582 4370
Sweden	08 587 895 00	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 2377 1200	02 2737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

# Glossary

---

Prefix	Meaning	Value
n-	nano-	$10^{-9}$
m-	milli-	$10^{-3}$
M-	mega-	$10^6$

## A

acceptor handshake	Listeners use this GPIB interface function to receive data, and all devices use it to receive commands. <i>See</i> source handshake and handshake.
access board	The GPIB board that controls and communicates with the devices on the bus that are attached to it.
ANSI	American National Standards Institute.
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange.
asynchronous	An action or event that occurs at an unpredictable time with respect to the execution of a program.
automatic serial polling	A feature of the GPIB software in which serial polls are executed automatically by the driver whenever a device asserts the GPIB SRQ line. Also called autopolling.

## B

base I/O address	<i>See</i> I/O address.
BIOS	Basic Input/Output System.
board-level function	A rudimentary function that performs a single operation.

## C

CFE	Configuration Enable. The GPIB command which precedes CFGn and is used to place devices into their configuration mode.
CFGn	These GPIB commands (CFG1 through CFG15) follow CFE and are used to configure all devices for the number of meters of cable in the system so HS488 transfers occur without errors.
CIC	Controller-In-Charge. The device that manages the GPIB by sending interface messages to other devices.
CPU	Central processing unit.

## D

DAV	Data Valid. One of the three GPIB handshake lines. <i>See</i> handshake.
DCL	Device Clear. The GPIB command used to reset the device or internal functions of all devices. <i>See</i> SDC.
device-level function	A function that combines several rudimentary board operations into one function so that the user does not have to be concerned with bus management or other GPIB protocol matters.
DIO1 through DIO8	The GPIB lines that are used to transmit command or data bytes from one device to another.
DLL	Dynamic link library.
DMA	Direct memory access. High-speed data transfer between the GPIB board and memory that is not handled directly by the CPU. Not available on some systems. <i>See</i> programmed I/O.
driver	Device driver software installed within the operating system.

## E

END or END Message	A message that signals the end of a data string. END is sent by asserting the GPIB End or Identify (EOI) line with the last data byte.
EOI	A GPIB line that signals either the last byte of a data message (END) or the parallel poll Identify (IDY) message.

EOS or EOS Byte	A 7- or 8-bit end-of-string character that is sent as the last byte of a data message.
EOT	End of transmission.
ESB	The Event Status bit. Part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.

## F

FIFO	First-in-first-out.
------	---------------------

## G

GET	Group Execute Trigger. The GPIB command used to trigger a device or internal function of an addressed Listener.
GPIB	General Purpose Interface Bus is the common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1992.
GPIB address	The address of a device on the GPIB, composed of a primary address (MLA and MTA) and perhaps a secondary address (MSA). The GPIB board has both a GPIB address and an I/O address.
GPIB board	Refers to the National Instruments family of GPIB interfaces.
GTL	Go To Local. The GPIB command used to place an addressed Listener in local (front panel) control mode.

## H

handshake	<p>The mechanism used to transfer bytes from the source handshake function of one device to the acceptor handshake function of another device. DAV, NRFD, and NDAC, three GPIB lines, are used in an interlocked fashion to signal the phases of the transfer, so that bytes can be sent asynchronously (for example, without a clock) at the speed of the slowest device.</p> <p>For more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987.</p>
-----------	---

hex	Hexadecimal; a number represented in base 16. For example, decimal 16 is hex 10.
high-level function	<i>See</i> device-level function.
HS488	A high-speed data transfer protocol for IEEE 488. This protocol increases performance for GPIB reads and writes up to 8 Mbytes/s, depending on your system.
Hz	Hertz.
<b>I</b>	
ibcnt	After each NI-488 I/O function, this global variable contains the actual number of bytes transmitted.
iberr	A global variable that contains the specific error code associated with a function call that failed.
ibsta	At the end of each function call, this global variable (status word) contains status information.
IEEE	Institute of Electrical and Electronic Engineers.
interface message	A broadcast message sent from the Controller to all devices and used to manage the GPIB.
I/O	Input/Output. In this manual, it is the transmission of commands or messages between the system via the GPIB board and other devices on the GPIB.
I/O address	The address of the GPIB board from the point of view of the CPU, as opposed to the GPIB address of the GPIB board. Also called port address or board address.
ISA	Industry Standard Architecture.
ist	An Individual Status bit of the status byte used in the Parallel Poll Configure function.
<b>K</b>	
KB	Kilobytes.

**L**

LAD	Listen address. <i>See</i> MLA.
language interface	Code that enables an application program that uses NI-488 functions or NI-488.2 routines to access the driver.
Listener	A GPIB device that receives data messages from a Talker.
LLO	Local Lockout. The GPIB command used to tell all devices that they may or should ignore remote (GPIB) data messages or local (front panel) controls, depending on whether the device is in local or remote program mode.
low-level function	A rudimentary board or device function that performs a single operation.

**M**

m	Meters.
MAV	The Message Available bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.
MB	Megabytes.
memory-resident	Resident in RAM.
MLA	My Listen Address. A GPIB command used to address a device to be a Listener. It can be any one of the 31 primary addresses.
MSA	My Secondary Address. The GPIB command used to address a device to be a Listener or a Talker when extended (two-byte) addressing is used. The complete address is a MLA or MTA address followed by an MSA address. There are 31 secondary addresses for a total of 961 distinct listen or talk addresses for devices.
MTA	My Talk Address. A GPIB command used to address a device to be a Talker. It can be any one of the 31 primary addresses.
multitasking	The concurrent processing of more than one program or task.

## N

NDAC	Not Data Accepted. One of the three GPIB handshake lines. <i>See</i> handshake.
NRFD	Not Ready For Data. One of the three GPIB handshake lines. <i>See</i> handshake.

## P

parallel poll	The process of polling all configured devices at once and reading a composite poll response. <i>See</i> serial poll.
PC	Personal computer.
PCI	Peripheral Component Interconnect.
PIO	<i>See</i> programmed I/O.
PPC	Parallel Poll Configure. It is the GPIB command used to configure an addressed Listener to participate in polls.
PPD	Parallel Poll Disable. It is the GPIB command used to disable a configured device from participating in polls. There are 16 PPD commands.
PPE	Parallel Poll Enable. It is the GPIB command used to enable a configured device to participate in polls and to assign a DIO response line. There are 16 PPE commands.
PPU	Parallel Poll Unconfigure. It is the GPIB command used to disable any device from participating in polls.
programmed I/O	Low-speed data transfer between the GPIB board and memory in which the CPU moves each data byte according to program instructions. <i>See</i> DMA.

## R

RAM	Random-access memory.
resynchronize	The GPIB software and the user application must resynchronize after asynchronous I/O operations have completed.
RQS	Request Service.



**S**

s	Seconds.
SDC	Selected Device Clear. The GPIB command used to reset internal or device functions of an addressed Listener. <i>See</i> DCL.
semaphore	An object that maintains a count between zero and some maximum value, limiting the number of threads that are simultaneously accessing a shared resource.
serial poll	The process of polling and reading the status byte of one device at a time. <i>See</i> parallel poll.
service request	<i>See</i> SRQ.
source handshake	The GPIB interface function that transmits data and commands. Talkers use this function to send data, and the Controller uses it to send commands. <i>See</i> acceptor handshake and handshake.
SPD	Serial Poll Disable. The GPIB command used to cancel an SPE command.
SPE	Serial Poll Enable. The GPIB command used to enable a specific device to be polled. That device must also be addressed to talk. <i>See</i> SPD.
SRQ	Service Request. The GPIB line that a device asserts to notify the CIC that the device needs servicing.
status byte	The IEEE 488.2-defined data byte sent by a device when it is serially polled.
status word	<i>See</i> <i>ibsta</i> .
synchronous	Refers to the relationship between the GPIB driver functions and a process when executing driver functions is predictable; the process is blocked until the driver completes the function.
System Controller	The single designated Controller that can assert control (become CIC of the GPIB) by sending the Interface Clear (IFC) message. Other devices can become CIC only by having control passed to them.

## T

TAD	Talk Address. <i>See</i> MTA.
Talker	A GPIB device that sends data messages to Listeners.
TCT	Take Control. The GPIB command used to pass control of the bus from the current Controller to an addressed Talker.
timeout	A feature of the GPIB driver that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.
TLC	An integrated circuit that implements most of the GPIB Talker, Listener, and Controller functions in hardware.

## U

ud	Unit descriptor. A variable name and first argument of each function call that contains the unit descriptor of the GPIB interface or other GPIB device that is the object of the function.
UNL	Unlisten. The GPIB command used to unaddress any active Listeners.
UNT	Untalk. The GPIB command used to unaddress an active Talker.

# Index

---

## Numbers/Symbols

- ! (repeat previous function) function, Interactive Control, 6-11
- \$ filename (execute indirect file) function, Interactive Control, 6-11
- + (turn ON display) function, Interactive Control, 6-11
- (turn OFF display) function, Interactive Control, 6-11

## A

active Controller. *See* Controller-in-Charge (CIC).

addresses. *See* GPIB addresses.

AllSpoll routine

- description, 7-15

- SRQ and serial polling (example), 7-16

application development. *See also* debugging; NI-488.2 API.

- accessing NI-488.2 driver, 3-3 to 3-4

  - choosing a method, 3-3 to 3-4

  - direct entry access, 3-4

  - NI-488.2 language interfaces, 3-4

- checking status with global variables, 3-6 to 3-8

  - count variables (ibcnt and ibcntl), 3-8

  - error variable (iberr), 3-7 to 3-8

  - status word (ibsta), 3-6 to 3-7

- communicating with single GPIB device, 3-8 to 3-10

  - clearing devices, 3-10

  - device communication, 3-10

  - general steps and examples, 3-9 to 3-10

  - items to include, 3-8 to 3-9

  - opening devices, 3-9 to 3-10

  - placing device offline, 3-10

- instrument control

  - interactive, 3-3

  - simple, using NI-488.2 Communicator, 3-1 to 3-3

- Interactive Control for communicating with devices, 3-8

- language-specific instructions, 3-13 to 3-17

  - Borland C/C++, 3-13

  - direct entry with C, 3-14 to 3-17

    - directly accessing gpib-32.dll

      - exports, 3-15 to 3-17

      - gpib-32.dll exports, 3-14 to 3-15

  - Microsoft Visual Basic, 3-13 to 3-14

  - Microsoft Visual C/C++, 3-13

- multiple interfaces or multiple GPIB devices

  - becoming Controller-In-Charge, 3-11

  - communicating with devices, 3-12

  - determining GPIB address of device, 3-11 to 3-12

  - general steps and examples, 3-11 to 3-13

  - initialization, 3-11 to 3-12

  - initializing devices, 3-12

  - items to include, 3-10 to 3-11

  - placing device offline, 3-13

- NI-488.2 API, 3-4 to 3-5

  - communicating with one GPIB device, 3-4 to 3-5

  - low-level functions, 3-5

  - multiple interfaces and/or multiple devices, 3-5

  - overview, 3-4

- applications, existing. *See* existing applications.

- asynchronous event notification in Win32

  - applications, 7-4 to 7-9

    - calling ibnotify function, 7-4 to 7-5

    - ibnotify programming example, 7-5 to 7-9

- ATN (attention) line (table), A-3
- ATN status word condition
  - bit position, hex value, and type (table), 3-7
  - description, B-4
- automatic serial polling. *See* serial polling.
- auxiliary functions, Interactive Control (table), 6-11
  
- B**
- Borland C/C++, 3-13
- buffer option function, Interactive Control, 6-11
  
- C**
- C language direct entry for application development, 3-14 to 3-17
  - directly accessing gpib-32.dll exports, 3-15 to 3-17
  - gpib-32.dll exports, 3-14 to 3-15
- cable length for high-speed data transfers, 7-2, 7-3
- CIC. *See* Controller-in-Charge (CIC).
- CIC status word condition
  - bit position, hex value, and type (table), 3-7
  - description, B-4
- CMPL status word condition
  - bit position, hex value, and type (table), 3-7
  - description, B-3
- common questions. *See* troubleshooting and common questions.
- communicating with GPIB devices
  - advanced communication, 2-9
  - instrument commands, 2-10
  - multiple GPIB devices, 3-10 to 3-13
    - becoming Controller-In-Charge, 3-11
    - communicating with devices, 3-12
    - determining GPIB address of device, 3-11 to 3-12
    - general steps and examples, 3-11 to 3-13
    - initialization, 3-11 to 3-12
    - initializing devices, 3-12
    - items to include, 3-10 to 3-11
    - placing device offline, 3-13
    - programming considerations, 3-5
  - query/read/write communication, 2-7 to 2-9
  - single GPIB device, 3-8 to 3-10
    - clearing devices, 3-10
    - device communication, 3-10
    - general steps and examples, 3-9 to 3-10
    - items to include, 3-8 to 3-9
    - opening devices, 3-9 to 3-10
    - placing device offline, 3-10
    - programming considerations, 3-4 to 3-5
    - using NI-488.2 Communicator, 2-7 to 2-10
- communication errors, 4-5
  - repeat addressing, 4-5
  - termination method, 4-5
- configuration, 1-1 to 1-3. *See also* Interactive Control utility.
  - errors in configuration, 4-4
  - linear and star system configuration (figure), 1-1
  - requirements, 1-2 to 1-3
  - system configuration effects on HS488, 7-3
  - viewing or changing GPIB interface settings, 2-14 to 2-16
    - Windows 98/95, 2-14 to 2-15
    - Windows NT, 2-15 to 2-16
- Configure (CFGn) message, 7-3
- Configure Enable (CFE) message, 7-3

Controller-in-Charge (CIC)  
 active Controller as CIC, A-1  
 becoming Controller-In-Charge  
 (example), 3-11  
 making GPIB board CIC, 7-11  
 System Controller as, A-1

Controllers  
 definition, A-1  
 idle Controller, A-1  
 monitoring by Talker/Listener  
 applications, 7-11 to 7-12  
 System Controller, A-1

count information, in Interactive Control, 6-12  
 count variables (ibcnt and ibcntl), 3-8

## D

data lines, A-2  
 data transfers  
 high-speed (HS488), 7-2 to 7-3  
 enabling, 7-2 to 7-3  
 system configuration effects, 7-3  
 terminating, 7-1 to 7-2

DAV (data valid) line (table), A-3

DCAS status word condition  
 bit position, hex value, and type  
 (table), 3-7  
 description, B-5  
 Talker/Listener applications, 7-12

debugging. *See also* NI Spy utility;  
 troubleshooting and common questions.  
 communication errors, 4-5  
 repeat addressing, 4-5  
 termination method, 4-5  
 configuration errors, 4-4  
 existing applications, 4-3  
 global status variables, 4-2 to 4-3  
 GPIB error codes (table), C-1  
 NI Spy, 4-1 to 4-2

NI-488.2 error codes, 4-3  
 other errors, 4-6  
 timing errors, 4-4

DevClearList routine (example), 3-12

Device Manager device status codes,  
 troubleshooting, D-2 to D-3

device-level calls and bus management, 7-11

direct access to NI-488.2 dynamic link  
 library, 3-4

documentation  
 conventions used in manual, *xii*  
 overview of NI-488.2 documentation, *xi*  
 related documentation, *xii*

DOS NI-488.2 applications  
 enabling/disabling support  
 under Windows 98/95, 2-18  
 under Windows NT, 2-19  
 running  
 under Windows 98/95, 3-17 to 3-18  
 under Windows NT, 3-18

DTAS status word condition  
 bit position, hex value, and type  
 (table), 3-7  
 description, B-5  
 Talker/Listener applications, 7-12

## E

EABO error code, C-5  
 EADR error code, C-4  
 EARG error code, C-4  
 EBUS error code, C-7 to C-8  
 ECAP error code, C-7  
 ECIC error code, C-2 to C-3  
 EDMA error code, C-6  
 EDVR error code  
 description, C-2  
 troubleshooting, D-1 to D-2

EFSO error code, C-7

e-mail support, F-1

- END status word condition
    - bit position, hex value, and type (table), 3-6
    - description, B-2
  - end-of-string character. *See* EOS.
  - ENEB error code, C-5 to C-6
  - ENOL error code, C-3
  - EOI (end or identify) line
    - purpose (table), A-3
    - termination of data transfers, 7-1
  - EOIP error code, C-6 to 6-7
  - EOS
    - configuring EOS mode, 7-1 to 7-2
    - EOS comparison method, 7-1
    - EOS read method, 7-2
    - EOS write method, 7-1 to 7-2
  - ERR status word condition
    - bit position, hex value, and type (table), 3-6
    - description, B-2
  - error codes and solutions
    - debugging applications, 4-3
    - EABO, C-5
    - EADR, C-4
    - EARG, C-4
    - EBUS, C-7 to C-8
    - ECAP, C-7
    - ECIC, C-2 to C-3
    - EDMA, C-6
    - EDVR, C-2
    - EFSO, C-7
    - ENEB, C-5 to C-6
    - ENOL, C-3
    - EOIP, C-6 to 6-7
    - ESAC, C-5
    - ESRQ, C-8 to C-9
    - ESTB, C-8
    - ETAB, C-9
    - GPIB error codes (table), C-1
  - error conditions
    - communication errors, 4-5
      - repeat addressing, 4-5
      - termination method, 4-5
    - configuration errors, 4-4
    - Interactive Control error information, 6-12
    - timing errors, 4-4
  - error variable (iberr), 3-7 to 3-8
  - ESAC error code, C-5
  - ESRQ error code, C-8 to C-9
  - ESTB error code, C-8
  - ETAB error code, C-9
  - event notification. *See* asynchronous event notification in Win32 applications.
  - Event Status bit (ESB), 7-12
  - execute function n times (n \*) function, Interactive Control, 6-11
  - execute indirect file (\$) function, Interactive Control, 6-11
  - execute previous function n times (n \* !) function, Interactive Control, 6-11
  - existing applications
    - debugging, 4-3
    - running under Windows 98/95
      - DOS NI-488.2 applications, 3-17 to 3-18
      - Win16 NI-488.2 applications, 3-17
    - running under Windows NT, 3-18
- ## F
- fax and telephone support, F-2
  - Fax-on-Demand support, F-1
  - FindLstn routine
    - determining GPIB address (example), 3-11 to 3-12
    - using in Interactive Control (example), 6-3

FindRQS routine  
 description, 7-15  
 SRQ and serial polling (example),  
 7-15 to 7-16  
 FTP support, F-1  
 functions. *See* NI-488.2 API.

## G

General Purpose Interface Bus. *See* GPIB.  
 Getting Started Wizard, 2-3 to 2-4  
 global variables, 3-6 to 3-8  
 count variables (ibcnt and ibcntl), 3-8  
 debugging applications, 4-2 to 4-3  
 error variable (iberr), 3-7 to 3-8  
 status word (ibsta), 3-6 to 3-7  
 writing multithreaded Win32 NI-488.2  
 applications, 7-9 to 7-10  
 GPIB  
 configuration, 1-1 to 1-3  
 controlling more than one  
 interface, 1-2  
 linear and star system configuration  
 (figure), 1-1  
 requirements, 1-2 to 1-3  
 definition, A-1  
 overview, A-1  
 resources on National Instruments GPIB  
 Web site, 2-19  
 sending messages across, A-2 to A-3  
 data lines, A-2  
 handshake lines, A-3  
 interface management lines, A-3  
 Talkers, Listeners, and Controllers, A-1  
 viewing or changing interface settings,  
 2-14 to 2-16  
 Windows 98/95, 2-14 to 2-15  
 Windows NT, 2-15 to 2-16  
 GPIB addresses  
 address bit configuration (figure), A-2  
 listen address, A-2

primary, A-2  
 purpose, A-2  
 repeat addressing, 4-5  
 secondary, A-2  
 syntax in Interactive Control, 6-6  
 talk address, A-2

## GPIB instruments

adding new, 2-10 to 2-11  
 changing device templates, 2-17 to 2-18  
 Windows 98/95, 2-18  
 Windows NT, 2-18  
 communicating with. *See* communicating  
 with GPIB devices.  
 multiple. *See* multiple interfaces or  
 multiple GPIB devices.  
 scanning for, 2-6 to 2-7  
 viewing information, 2-16 to 2-17

## gpib-32.dll exports

accessing directly, 3-15 to 3-17  
 direct entry with C, 3-14 to 3-15

## GPIB-ENET network settings (Windows 98/95)

assigning IP address, 2-20  
 configuring advanced IP settings, 2-20  
 updating GPIB-ENET firmware, 2-20

## H

handshake lines, A-3  
 help  
 accessing additional help resources  
 National Instruments GPIB Web site,  
 2-19  
 NI-488.2 online help, 2-19  
 NI Spy help, 5-3  
 help (display Interactive utility online help)  
 function (table), 6-11  
 help option function, Interactive utility, 6-11

HS488 (high-speed data transfers), 7-2 to 7-3  
 enabling HS488, 7-2 to 7-3  
 setting cable length, 7-2  
 system configuration effects, 7-3  
 HS488 configuration message, 7-3

## I

ibask function, 7-3  
 ibclr function  
   clearing devices (example), 3-10  
   using in Interactive Control (example), 6-4  
 ibcnt and ibcntl variables, 3-8  
 ibconfig function  
   configuring GPIB board as CIC, 7-2  
   configuring NI-488.2 driver, 4-4  
   determining assertion of EOI line, 7-2  
   enabling autopoling, 7-13  
   enabling high-speed data transfers, 7-2 to 7-3  
 ibdev function  
   opening devices (example), 3-9 to 3-10  
   using in Interactive Control (example), 6-3  
 ibeos function, 7-1  
 ibeot function, 7-1  
 iberr error variable, 3-7 to 3-8  
 ibnotify function  
   asynchronous event notification in Win32 applications (example), 7-5 to 7-9  
   calling, 7-4 to 7-5  
 ibonl function  
   placing device offline (example), 3-10, 3-13  
   using in Interactive Control (example), 6-5  
 ibppc function  
   conducting parallel polls, 7-17 to 7-18  
   unconfiguring device for parallel polling, 7-18  
 ibrd function  
   communicating with device (example), 3-10  
   using in Interactive Control (example), 6-4  
 ibrpp function, 7-18  
 ibrsp function  
   automatic serial polling, 7-13  
   SRQ and serial polling, 7-14  
 ibsta. *See* status word (ibsta).  
 ibwait function  
   Talker/Listener applications, 7-11  
   terminating stuck SRQ state, 7-13  
   waiting for GPIB conditions, 7-4  
 ibwrt function  
   communicating with device (example), 3-10  
   using in Interactive Control (example), 6-4  
 IFC (interface clear) line, A-3  
 Interactive Control utility  
   advanced GPIB communication, 2-9  
   auxiliary functions (table), 6-11  
   communicating with devices, 3-3, 3-8  
   count information, 6-12  
   error information, 6-12  
   getting started, 6-1 to 6-5  
   NI-488.2 call examples, 6-2 to 6-5  
   overview, 6-1  
   programming considerations, 3-8  
   starting, 3-3  
   status word, 6-11  
   syntax, 6-5 to 6-6  
     addresses, 6-6  
     board-level calls (table), 6-8 to 6-9  
     device-level calls (table), 6-7 to 6-8  
     multi-device NI-488.2 calls (table), 6-9 to 6-10  
     numbers, 6-5  
     strings, 6-6



interface management lines, A-3  
 interrupts and autopolling, 7-13 to 7-14

## L

LACS status word condition  
   bit position, hex value, and type  
   (table), 3-7  
   description, B-5  
   Talker/Listener applications, 7-11 to 7-12

languages

  language-specific instructions,  
   3-13 to 3-17  
     Borland C/C++, 3-13  
     direct entry with C, 3-14 to 3-17  
     directly accessing gpib-32.dll  
     exports, 3-15 to 3-17  
     gpib-32.dll exports, 3-14 to 3-15  
     Microsoft Visual Basic, 3-13 to 3-14  
     Microsoft Visual C/C++, 3-13  
   NI-488.2 language interfaces, 3-4

listen address, setting, A-2

Listeners, A-1. *See also* Talker/Listener  
 applications.

LOK status word condition

  bit position, hex value, and type  
   (table), 3-7  
   description, B-3

## M

manual. *See* documentation.

Measurement & Automation Explorer,  
 2-1 to 2-20

  accessing help and resources  
     National Instruments GPIB Web site,  
     2-19  
     NI-488.2 online help, 2-19  
   adding new GPIB instruments,  
   2-10 to 2-11

changing GPIB device templates,  
 2-17 to 2-18

  Windows 98/95, 2-18

  Windows NT, 2-18

communicating with GPIB instruments,  
 2-7 to 2-10

  advanced communication, 2-9

  instrument commands, 2-10

  query/read/write communication,  
   2-7 to 2-9

enabling/disabling NI-488.2 DOS  
 support, 2-18 to 2-19

  Windows 98/95, 2-18

  Windows NT, 2-19

getting started with NI-488.2, 2-3 to 2-4

GPIB-ENET network settings (Windows  
 98/95)

  assigning IP address, 2-20

  configuring advanced IP  
   settings, 2-20

  updating GPIB-ENET firmware,  
   2-20

monitoring, recording, and displaying  
 NI-488.2 calls, 2-13

overview, 2-1

scanning for GPIB instruments, 2-6 to 2-7

starting, 2-2

troubleshooting NI-488.2 problems,  
 2-4 to 2-5

viewing GPIB instrument information,  
 2-16 to 2-17

viewing NI-488.2 software version, 2-12

viewing or changing GPIB interface  
 settings, 2-14 to 2-16

  Windows 98/95, 2-14 to 2-15

  Windows NT, 2-15 to 2-16

Message Available (MAV) bit, 7-12

messages, sending across GPIB, A-2 to A-3  
 data lines, A-2

handshake lines, A-3

- interface management lines, A-3
  - Microsoft Visual Basic, 3-13 to 3-14
  - Microsoft Visual C/C++ programming instructions, 3-13
  - multiple interfaces or multiple GPIB devices
    - becoming Controller-In-Charge, 3-11
    - communicating with devices, 3-12
    - determining GPIB address of device, 3-11 to 3-12
    - general steps and examples, 3-11 to 3-13
    - initialization, 3-11 to 3-12
    - initializing devices, 3-12
    - items to include, 3-10 to 3-11
    - placing device offline, 3-13
  - multithreaded Win32 NI-488.2 applications, writing, 7-9 to 7-10
- N**
- n \* ! (execute previous function n times) function, Interactive Control, 6-11
  - n \* (execute function n times) function, Interactive Control, 6-11
  - NDAC (not data accepted) line (table), A-3
  - NI Spy utility
    - debugging applications, 4-1 to 4-2
      - existing applications, 4-3
      - using global status variables, 4-2 to 4-3
    - exiting, 5-4
    - locating errors, 5-3
    - main window (figure), 5-3
    - online help, 5-3
    - overview, 5-1
    - performance considerations, 5-4
    - starting, 4-1 to 4-2, 5-1 to 5-2
    - viewing properties for recorded calls, 5-3
  - NI-488.2 API. *See also* application development; debugging.
    - asynchronous event notification in Win32 applications, 7-4 to 7-9
      - calling ibnotify function, 7-4 to 7-5
      - ibnotify programming example, 7-5 to 7-9
    - communicating with one GPIB device, 3-4 to 3-5
    - device-level calls and bus management, 7-11
    - getting started with NI-488.2, 2-3 to 2-4
    - high-speed data transfers, 7-2 to 7-3
      - enabling HS488, 7-2 to 7-3
      - system configuration effects, 7-3
    - Interactive Control syntax, 6-5 to 6-11
      - board-level traditional NI-488.2 calls (table), 6-8 to 6-9
      - device-level traditional NI-488.2 calls (table), 6-7 to 6-8
      - examples, 6-2 to 6-5
      - multi-device NI-488.2 calls (table), 6-9 to 6-10
    - language interfaces, 3-4
    - low-level functions, 3-5
    - monitoring, recording, and displaying calls, 2-13
    - multiple interfaces and/or multiple devices, 3-5
    - overview, 3-4
    - parallel polling, 7-17 to 7-19
      - implementing, 7-17 to 7-19
      - using multi-device NI-488.2 calls, 7-19
      - using traditional NI-488.2 calls, 7-17 to 7-18
    - serial polling, 7-12 to 7-16
      - automatic serial polling, 7-13 to 7-14
      - autopolling and interrupts, 7-13 to 7-14
      - stuck SRQ state, 7-13

- service requests
  - from IEEE 488 devices, 7-12
  - from IEEE 488.2 devices, 7-12
- SRQ and serial polling
  - AllSpoll (example), 7-16
  - with device-level traditional NI-488.2 calls, 7-14
  - FindRQS (example), 7-15 to 7-16
  - with multi-device NI-488.2 calls, 7-15 to 7-16
- Talker/Listener applications, 7-11 to 7-12
- termination of data transfers, 7-1 to 7-2
- troubleshooting. *See* troubleshooting and common questions.
- viewing software version, 2-12
- waiting for GPIB conditions, 7-4
- writing multithreaded Win32 NI-488.2 applications, 7-9 to 7-10
- NI-488.2 Communicator
  - query/read/write communication, 2-7 to 2-9
  - simple instrument control, 3-1 to 3-3
- NI-488.2 driver
  - choosing access method, 3-3 to 3-4
  - direct entry access, 3-4
- NRFD (not ready for data) line (table), A-3
- number syntax, in Interactive Control, 6-5

## O

online help for NI Spy, 5-3

## P

- parallel polling, 7-17 to 7-19
  - implementing, 7-17 to 7-19
  - using multi-device NI-488.2 calls, 7-19
  - using traditional NI-488.2 calls, 7-17 to 7-18

- PPoll routine, 7-19
- PPollConfig routine, 7-19
- PPollUnconfig routine, 7-19
- primary GPIB address, A-2
- problem solving. *See* debugging; troubleshooting and common questions.
- programming. *See* application development; debugging; NI-488.2 API.

## Q

- q function, Interactive Control, 6-11

## R

- ReadStatusByte routine, 7-15
- Receive routine (example), 3-12
- REM status word condition
  - bit position, hex value, and type (table), 3-7
  - description, B-4
- REN (remote enable) line (table), A-3
- repeat addressing, 4-5
- repeat previous function (!) function, Interactive Control, 6-11
- RQS status word condition
  - bit position, hex value, and type (table), 3-7
  - description, B-3
- running existing applications. *See* existing applications.

## S

- scanning for GPIB instruments, 2-6 to 2-7
- secondary GPIB address, A-2
- SendIFC routine
  - becoming Controller-in-Charge (example), 3-11
  - using in Interactive Control (example), 6-3

- SendList routine (example), 3-12
  - serial polling, 7-12 to 7-16
    - automatic serial polling, 7-13 to 7-14
      - autopolling and interrupts, 7-13 to 7-14
      - stuck SRQ state, 7-13
    - service requests
      - from IEEE 488 devices, 7-12
      - from IEEE 488.2 devices, 7-12
    - SRQ and serial polling
      - with device-level traditional NI-488.2 calls, 7-14
      - with multi-device NI-488.2 calls, 7-15 to 7-16
  - set 488.2 v function, Interactive Control utility, 6-11
  - set uptime function, Interactive Control utility, 6-11
  - setting up your system. *See* configuration.
  - single GPIB device, communicating with. *See* communicating with GPIB devices.
  - SRQ (service request) line
    - automatic serial polling, 7-13 to 7-14
    - purpose (table), A-3
    - stuck SRQ state, 7-13
  - SRQI status word condition
    - bit position, hex value, and type (table), 3-7
    - description, B-3
  - status word (ibsta), 3-6 to 3-7
    - ATN, B-4
    - CIC, B-4
    - CMPL, B-3
    - DCAS, 7-12, B-5
    - DTAS, 7-12, B-5
    - END, B-2
    - ERR, B-2
    - Interactive Control example, 6-11
    - LACS, 7-12, B-5
    - LOK, B-3
    - programming considerations, 3-6 to 3-7
    - REM, B-4
    - RQS, B-3
    - SRQI, B-3
    - status word layout (table), 3-6 to 3-7, B-1
    - TACS, 7-12, B-4
    - TIMO, B-2
  - string syntax, in Interactive Control, 6-6
  - stuck SRQ state, 7-13
  - System Controller as Controller-in-Charge, A-1
- ## T
- TACS status word condition
    - bit position, hex value, and type (table), 3-7
    - description, B-4
    - Talker/Listener applications, 7-12
  - talk address, setting, A-2
  - Talker/Listener applications, 7-11 to 7-12
  - Talkers, A-1
  - technical support resources, F-1 to F-2
  - telephone and fax support, F-2
  - termination methods, errors caused by, 4-5
  - termination of data transfers, 7-1 to 7-2
  - TestSRQ routine, 7-15
  - timing errors, 4-4
  - TIMO status word condition
    - bit position, hex value, and type (table), 3-6
    - description, B-2
  - troubleshooting and common questions. *See also* debugging; Interactive Control utility; NI Spy utility.
    - NI-488.2 problems, using Troubleshooting Wizard, 2-4 to 2-5
    - Windows 98/95, D-1 to D-6
      - common questions, D-3 to D-6
      - Device Manager device status code, D-2 to D-3
      - EDVR error conditions, D-1 to D-2

- Windows NT, E-1 to E-4
  - common questions, E-2 to E-4
  - examining NT system log using Event Viewer, E-2
  - using diagnostic tools, E-1 to E-2
  - verifying NI-488.2 installation, E-1 to E-2
- turn OFF display (-) function, Interactive Control, 6-11
- turn ON display (+) function, Interactive Control, 6-11

## V

- Visual Basic, 3-13 to 3-14
- Visual C/C++ programming instructions, 3-13

## W

- wait function. *See* *ibwait* function.
- WaitSRQ routine, 7-15
- Web site support, F-1
- Windows environment
  - asynchronous event notification in Win32 applications, 7-4 to 7-9
    - calling *ibnotify* function, 7-4 to 7-5
    - ibnotify* programming example, 7-5 to 7-9
  - changing GPIB device templates
    - Windows 98/95, 2-18
    - Windows NT, 2-18

- enabling/disabling NI-488.2 DOS support
  - Windows 98/95, 2-18
  - Windows NT, 2-19
- GPIB-ENET network settings (Windows 98/95), 2-20
- running Win16 NI-488.2 applications under Windows 98/95, 3-17
- running Win32 applications, 3-17 to 3-18
- troubleshooting and common questions
  - Windows 98/95, D-1 to D-6
    - common questions, D-3 to D-6
    - Device Manager device status code, D-2 to D-3
    - EDVR error conditions, D-1 to D-2
  - Windows NT, E-1 to E-4
    - common questions, E-2 to E-4
    - examining NT system log using Event Viewer, E-2
    - using diagnostic tools, E-1 to E-2
    - verifying NI-488.2 installation, E-1 to E-2
- viewing or changing GPIB interface settings
  - Windows 98/95, 2-14 to 2-15
  - Windows NT, 2-15 to 2-16