

NI-488.2 Help

June 2008, 370003J-01

Use this help file for function information, troubleshooting your program, and an overall summary of the NI-488.2 software.

For more information about this help file, refer to the following topics:

Using Help

Related Documentation

<u>Glossary</u>

Important Information

Technical Support and Professional Services

To comment on National Instruments documentation, refer to the <u>National</u> <u>Instruments Web site</u>.

© 19992008 National Instruments Corporation. All rights reserved.

Related Documentation

The following documents contain information that you may find helpful as you use this help file:

- ANSI/IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation
- ANSI/IEEE Standard 488.1-2003, IEEE Standard for Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation
- ANSI/IEEE Standard 488.2-1987, IEEE Standard Codes, Formats, Protocols, and Common Commands
- ANSI/IEEE Standard 488.2-1992, IEEE Standard Codes, Formats, Protocols, and Common Commands
- Microsoft Windows help
- Microsoft Win32 Software Development Kit for Microsoft Windows
- Readme file

Using Help

<u>Conventions</u> <u>Navigating Help</u> <u>Searching Help</u> <u>Printing Help File Topics</u>

Conventions

This help file uses the following formatting and typographical conventions:

< > Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal namefor example, AO <0..3>. [] Square brackets enclose optional itemsfor example, [response]. 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. P This icon denotes a tip, which alerts you to advisory information. M This icon denotes a note, which alerts you to important information. bold Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names. Underlined text in this color denotes a link to a help topic, <u>green</u> help file, or Web address. Italic text denotes variables, emphasis, cross references, or italic an introduction to a key concept. Italic text 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. monospace Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This bold

font also emphasizes lines of code that are different from the other examples.

monospace Italic text in this font denotes text that is a placeholder for a *italic* word or value that you must supply.

Navigating Help (Windows Only)

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:

- **Hide**—Hides the navigation pane from view.
- Locate—Locates the currently displayed topic in the Contents tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- Forward—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

Searching Help (Windows Only)

Use the **Search** tab to the left of this window to locate content in this help file. If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.

Wildcards

You also can search using asterisk (*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



Note Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

Nested Expressions

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

Boolean Expressions

Click the **•** button to add Boolean expressions to a search. The following Boolean operators are available:

- **AND** (default)—Returns topics that contain both search terms. You do not need to specify this operator unless you are using nested expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

Search Options

Use the following checkboxes on the **Search** tab to customize a search:

- **Search previous results**—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- Match similar words—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- Search titles only—Searches only in the titles of topics.

Printing Help File Topics (Windows Only)

Complete the following steps to print an entire book from the **Contents** tab:

- 1. Right-click the book.
- 2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
- 3. Select the **Print the selected heading and all subtopics** option.
 - Note Select Print the selected topic if you want to print the single topic you have selected in the **Contents** tab.
- 4. Click the **OK** button.

Printing PDF Documents

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

Getting Started with Your Hardware and Software

To use your GPIB hardware and NI-488.2 software to communicate with an instrument, follow these steps:

- 1. Install your GPIB hardware according to the instructions in the getting started documentation on the NI-488.2 for Windows CD. For more information about the documentation, refer to <u>Access</u> <u>Additional Help and Resources for GPIB</u>.
- 2. Following the GPIB <u>configuration requirements</u>, connect an instrument to your GPIB hardware and power it on.
- 3. Launch Measurement & Automation Explorer by selecting Start»Programs»National Instruments»Measurement & Automation or clicking the following button.
 - > Start Measurement & Automation Explorer.
- 4. In Measurement & Automation Explorer, scan for instruments connected to your GPIB hardware. For more information, refer to <u>Scan for GPIB Instruments</u>.
- 5. Use the NI-488.2 Communicator to send commands to and read responses from your instrument. For more information, refer to <u>Communicate with a GPIB Instrument</u>.

Measurement & Automation Explorer also gives you access to all the NI-488.2 software utilities. You can access online help, troubleshooting assistance, and the National Instruments Web site from the Measurement & Automation Explorer **Help** menu.

For an overview of how to set up and configure your system, refer to the <u>Setting up and Configuring Your System</u> topic.

Access Additional Help and Resources for GPIB

To access additional help and resources for the NI-488.2 software and your GPIB hardware, you can refer to the National Instruments GPIB Web site or the *NI-488.2 for Windows* CD.

National Instruments GPIB Web Site

To access the National Instruments Web site for GPIB in Measurement & Automation Explorer, select Help»National Instruments on the Web»GPIB Home Page.

NI-488.2 for Windows CD

The following NI-488.2 documentation is available on your *NI-488.2 for Windows* CD:

- The *GPIB Installation Guide* briefly describes how to install the NI-488.2 software and your GPIB hardware.
- The *NI-488.2 User Manual* describes the features and functions of the NI-488.2 software.
- The *GPIB Hardware Guide* contains detailed instructions on how to install and configure your GPIB hardware. This guide also includes hardware specifications and compliance information.
- The *GPIB Analyzer User Manual* contains instructions to help you use the GPIB analyzer software.
- The *NI-488.2 API Quick Reference Card* lists status word conditions, error codes, functions, board options, device options, multiline interface messages, routines, and timeout values for the NI-488.2 API.

To view these documents online, insert your NI-488.2 CD. When the NI-488.2 for Windows screen appears, select View Documentation. The documentation utility helps you find the documentation you want to view. You can also view these documents on our <u>Web site</u>.

Learning More about GPIB

GPIB Overview Setting up and Configuring Your System Talkers, Listeners, and Controllers Controller-In-Charge and System Controller GPIB Addresses Sending Messages across the GPIB IEEE 488 Command Messages Related Documentation Access Additional Help and Resources for GPIB HS488 Overview

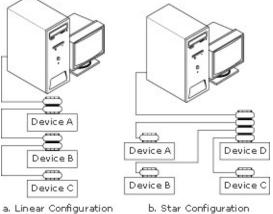
GPIB Overview

The ANSI/IEEE Standard 488.1-1987, also known as <u>GPIB</u>, 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 Mbytes/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 488.2-2003 is an extension of IEEE 488.1 that provides a means for achieving significantly higher data transfer rates (310 Mbytes/s) while maintaining compatibility with existing devices. 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.

For more information, refer to <u>HS488 Overview</u>.

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. The following illustration shows the linear and star configurations.



a. Linear Configuration

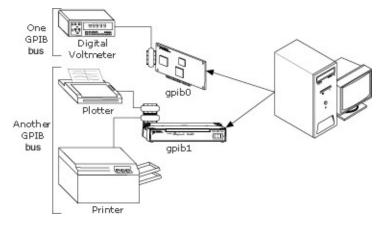
For more information, refer to the following topics:

Controlling More Than One Interface

Configuration Requirements

Controlling More Than One Interface

The following illustration 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.



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.

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.

Related Topics:

Communicating with Your Instrument FindLstn GPIB Addresses

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 Addresses

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

Most devices just use primary addressing. The GPIB Controller manages the communication across the GPIB by using the addresses to designate which devices should be listening or talking at any given moment. Typically your computer is the GPIB Controller and it manages communication with your GPIB device by sending messages to it and receiving messages from it.

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. The following table 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 030)				

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.

Related Topics:

Communicating with Your Instrument

<u>FindLstn</u>

Talkers, Listeners, and Controllers

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 following sections describe the 16 signal lines.

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. The <u>GPIB Handshake</u> <u>Lines</u> table summarizes the GPIB handshake lines.

Interface Management Lines

Five hardware lines manage the flow of information across the bus. The <u>GPIB Interface Management Lines</u> table summarizes the GPIB interface management lines.

GPIB Handshake Lines

The following table summarizes the GPIB handshake lines.

Line Description

NRFD (not pyte. Also used by the Talker to signal high-speed GPIB byte. Also used by the Talker to signal high-speed GPIB transfers.
 NDAC (not data accepted)

DAV Talking device indicates signals on data lines are stable (data (valid) data.

valid)

GPIB Interface Management Lines

The following table summarizes the GPIB interface management lines.

Line Description

ATN Controller drives ATN true when it sends commands and (attention) false when it sends data messages.

IFC System Controller drives the IFC line to initialize the bus and (interface make itself CIC. clear)

REN System Controller drives the REN line to place devices in remote or local program mode. enable)

SRQ Any device can drive the SRQ line to asynchronously request (service request)

EOI Talker uses the EOI line to mark the end of a data message. (end or controller uses the EOI line when it conducts a parallel poll.

NI-488.2 Utilities

Troubleshooting Tools Measurement & Automation Explorer Interactive Control Utility Analysis Tools

Troubleshooting Tools

You can use the following tools to troubleshoot any problems you encounter while using your GPIB hardware and the NI-488.2 software.

NI-488.2 Troubleshooting Utility

The NI-488.2 Troubleshooting Utility helps you troubleshoot any problems with your NI-488.2 software. This utility verifies that your GPIB hardware and the NI-488.2 software are installed correctly and working properly.

To start the NI-488.2 Troubleshooting Utility from within <u>Measurement &</u> <u>Automation Explorer</u>, select **Help*Troubleshooting*NI-488.2 Troubleshooting Utility** or click on the following button.

Start the NI-488.2 Troubleshooting

Utility.

Other Tools

Analysis Tools Measurement & Automation Explorer Interactive Control Utility

Measurement & Automation Explorer

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

- Launch troubleshooting tools for GPIB and NI-488.2 problems.
- <u>Scan for instruments</u> connected to your GPIB interface.
- Establish basic communication with your GPIB instruments.
- View information about your GPIB hardware and the NI-488.2 software.
- <u>Reconfigure GPIB interface settings</u>.
- Launch <u>analysis tools</u> to monitor NI-488.2 or VISA API calls to GPIB interfaces.

To start Measurement & Automation Explorer, select Start»Programs»National Instruments»Measurement & Automation or click on the following button.

- > Start Measurement & Automation Explorer.
- **Tip** To view the NI-488.2 utilities and help available for your GPIB interface, expand **Devices and Interfaces** by clicking on the **+** next to it. Right-click your GPIB interface and select an item from the context menu that appears.

Reconfigure GPIB Interface Settings

In Measurement & Automation Explorer, you can view GPIB hardware information and change default settings for new board handles.



Note The default settings are not applied to new board handles until all open handles are closed. Otherwise, the current board configuration continues to be used.

In most cases, you should change only the interface ID of your GPIB hardware. Applications that use ibconfig calls instead of customized default settings are more portable, because proper execution does not rely on a specific configuration.

For more information about GPIB interface settings, refer to the context help window in Measurement & Automation Explorer.

Using NI-488.2 Communicator

You can use the NI-488.2 Communicator to verify that you can establish simple communication with your GPIB instrument. The NI-488.2 Communicator is an interactive utility that allows you to write commands to your instrument and read responses back from your instrument. It provides detailed information about the status of the NI-488.2 calls and you can use it to print sample C source code that performs a simple query to a GPIB instrument.

To start NI-488.2 Communicator, complete the following steps:

- 1. > Start Measurement & Automation Explorer.
- 2. In Measurement & Automation Explorer, expand the **Devices** and Interfaces directory by clicking on the + next to the folder.
- 3. Right-click on the GPIB interface that your GPIB instrument is connected to.
- 4. Select **Scan for Instruments** from the drop-down menu that appears.
- 5. After the scan is complete, select your GPIB instrument in the left pane of the Measurement & Automation Explorer window.
- 6. Right-click on your instrument and select **Communicate with Instrument** from the drop-down menu that appears. The **NI-488.2 Communicator** dialog box appears.
- 7. Type a command in the **Send String** field and do one of the following:
 - To write a command to the instrument then read a response back, click on the **Query** button.
 - To write a command to the instrument, click on the **Write** button.
 - To read a response from the instrument, click on the **Read** button.
 - To configure special requirements for end of string (EOS) modes for your device, click on the Configure EOS button.

Interactive Control Utility

Overview Getting Started Using Status Reporting Error Information Count Return Syntax Rules Auxiliary Functions Commands

Interactive Control Overview

With the Interactive Control utility, you communicate with the GPIB devices through calls you interactively type in at the keyboard. For specific information about communicating with your particular device, refer to the manual 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 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 (<u>Iberr</u>) 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

You can access help in Interactive Control by entering help at the prompt, or you can get help on a specific function by entering help <function> at the prompt, where <function> is the name of the function for which you want help.

To start Interactive Control within <u>Measurement & Automation Explorer</u>, select **Tools»NI-488.2»Interactive Control** or click on the following button.

Start Interactive Control.

Related Topics:

Auxiliary Functions in Interactive Control

Count Return in Interactive Control

Error Information in Interactive Control Getting Started with Interactive Control Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Getting Started with Interactive Control

To get started with Interactive Control, complete the following steps:

1. Within <u>Measurement & Automation Explorer</u>, select **Tools**»NI-488.2»Interactive Control or click on the following button.

Start Interactive Control.

For help on any Interactive Control command, type help followed by the command. For example, type help ibdev or help devclear.

2. Open either a board handle or device handle to use for further NI-488.2 calls. Use <u>ibdev</u> to open a device handle, <u>ibfind</u> to open a board handle, or the set 488.2 command to switch to a 488.2 prompt.

The following example uses ibdev to open a device, assigns it to access board gpib0, chooses a primary address of 6 with no secondary address, sets a timeout of 10 seconds, enables the END message, and disables the EOS mode.

:ibdev

enter board index: 0 enter primary address: 6 enter secondary address: 0 enter timeout: 13 enter 'EOI on last byte' flag: 1 enter end-of-string mode/byte: 0

ud0:



Note If you type a command and no parameters, Interactive Control prompts you for the necessary arguments. If you already know the required arguments, you can type them at the command prompt, as follows:

:**ibdev 0 6 0 13 1 0** ud0:

R

- **Note** If you do not know the primary and secondary address of your GPIB instrument, right-click on your GPIB interface in Measurement & Automation Explorer and select Scan for Instruments. After Measurement & Automation Explorer scans your interface, it displays your instrument address in the right window pane.
- 3. After 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 <u>ibclr</u>, as follows:

ud0: ibclr [0100] (cmpl)

4. To write data to the device, use <u>ibwrt</u>. Make sure that you refer to the instrument user manual that came with your GPIB instrument for recognized command messages.

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

or, equivalently:

ud0: ibwrt "*IDN?" [0100] (cmpl) count: 5 5. To read data from your device, use <u>ibrd</u>. 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] (cmpl) 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] (cmpl) 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.

6. When you finish communicating with the device, make sure you put it offline using the <u>ibonl</u> command, as follows:

ud0: ibonl 0 [0100] (cmpl)

:

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

7. To exit Interactive control, type q.

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Error Information in Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Using Interactive Control

You can use the Interactive Control utility to practice communication with the instrument, troubleshoot problems, and develop your application. For more information, refer to the <u>Interactive Control Overview</u> topic.

To start Interactive Control within <u>Measurement & Automation Explorer</u>, select **Tools»NI-488.2»Interactive Control** or click on the following button.

Start Interactive Control.

Related Topics:Auxiliary Functions in Interactive ControlCount Return in Interactive ControlError Information in Interactive ControlGetting Started with Interactive ControlInteractive Control OverviewStatus Reporting in Interactive ControlSyntax Rules in Interactive Control.

Status Reporting in Interactive Control

In the Interactive Control utility, all NI-488.2 calls (except ibfind and ibdev) return the status word <u>Ibsta</u> 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] (cmpl)
count: 5
ud0:
```

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Syntax Rules in Interactive Control. Using Interactive Control.

Error Information in Interactive Control

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 cmpl) error: EBUS count: 1 ud0:

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

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Count Return in Interactive Control

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 help for that function, which you can access through the Index of this help file.

Related Topics:

Auxiliary Functions in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Syntax Rules in Interactive Control

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.
- Except for reads, the count parameter to calls 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 <u>Ibsta</u> 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.

For information about specific syntax rules, refer to the following topics:

Number Syntax

String Syntax

Address Syntax in Interactive Control

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Using Interactive Control.

Number Syntax in Interactive Control

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.

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

String Syntax in Interactive Control

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-ofstring (EOS) characters that indicate to the device that a transmission has ended. The two most common EOS characters are r and n. rrepresents 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 "F3R5T1/r/n".

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Address Syntax in Interactive Control

Some of the multi-device 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.

Related Topics:

Auxiliary Functions in Interactive Control Count Return in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Auxiliary Functions in Interactive Control

Function Description

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

-

Related Topics:

Count Return in Interactive Control Error Information in Interactive Control Getting Started with Interactive Control Interactive Control Overview Status Reporting in Interactive Control Syntax Rules in Interactive Control. Using Interactive Control.

Set Udname Interactive Control Function

Purpose

Select active device or board where <udname> is the symbolic name of the new device or board. Call <u>ibfind</u> or <u>ibdev</u> initially to open each device or board.

This function is available only in the Interactive Control utility.

Interactive Control Syntax

set <udname>

Example

set gpib0

Purpose

Start using multi-device NI-488.2 calls for board v. This function is available only in the Interactive Control utility.

Interactive Control Syntax

set 488.2 v

Example

set 488.2 0

Purpose

Display this help file.

This function is available only in the Interactive Control utility.

Interactive Control Syntax

help

Help <option> Interactive Control Function

Purpose

Display a help topic available in this help file, where <option> is any NI-488.2 or auxiliary call.

This function is available only in the Interactive Control utility.

Interactive Control Syntax

help <option>

Example

help ibwrt

! (Repeat Previous Function) Interactive Control Function

Repeat the most recently executed function.

!

- (Turn OFF display) Interactive Control Function

Turn OFF display.

-

Example

-

+ (Turn ON display) Interactive Control Function

Turn ON display.

+

Example

+

n * (Execute Function n Times) Interactive Control Function

Execute function *n* times, where <function> represents the correct Interactive Control function syntax.

n * <function>

Example

20*ibwrt "Hello"

Execute most recently executed function *n* times.

n* !

Example

20* !

\$ (Execute Indirect File) Interactive Control Function

Execute an indirect file, where <filename> is the pathname of a file that contains Interactive Control calls to be executed.

\$ <filename>

Example

\$ C:\MyDirectory\MyFile.txt

Buffer <option> Interactive Control Function

Set type of display used for buffers. Valid options are full, brief, ascii, and off.

buffer <option>

Example

buffer brief

Quit the Interactive Control utility.

q

Interactive Control Commands

The following tables summarize commands in the Interactive Control utility:

- The Syntax tables for <u>Device-Level Traditional NI-488.2 Calls</u> and for <u>Board-Level Traditional NI-488.2 Calls</u> summarize the syntax of the traditional NI-488.2 calls in the Interactive Control utility.
- The Syntax table for <u>Multi-Device NI-488.2 Calls</u> summarizes the syntax of the multi-device NI-488.2 calls in the Interactive Control utility.
- The <u>Auxiliary Functions</u> table summarizes the auxiliary functions that you can use in the Interactive Control utility. For more information about the function parameters, use the help, available by typing help. If you type only the function name, the Interactive Control utility prompts you for its parameters.

Syntax for Board-Level Traditional NI-488.2 Calls in Interactive Control

Syntax	Description
<u>ibask</u> option	Return configuration information where option is a mnemonic for a configuration parameter
<u>ibcac</u> v	Become active Controller
<u>ibcmd</u> cmdbuf	Send commands
<u>ibcmda</u> cmdbuf	Send commands asynchronously
ibconfig option value	Alter configurable parameters where option is mnemonic for a configuration parameter
<u>ibdma</u> v	Enable/disable DMA
<u>ibeos</u> v	Change/disable EOS message
<u>ibeot</u> v	Enable/disable END message
ibfind_ udname	Return unit descriptor where udname is the symbolic name of interface (for example, gpib0)
<u>ibgts</u> v	Go from Active Controller to standby
<u>ibist</u> v	Set/clear ist
<u>iblck</u> v LockWaitTime	Acquire or release an exclusive interface lock for the current process
<u>iblines</u>	Read the state of all GPIB control lines
<u>ibln</u> pad sad	Check for presence of device on the GPIB at pad, sad
<u>ibloc</u>	Go to local
<u>ibonl</u> v	Place device online or offline
<u>ibpad</u> v	Change primary address
<u>ibppc_</u> v	Parallel poll configure
<u>ibrd</u> count	Read data where count is the bytes to read
<u>ibrda</u> count	Read data asynchronously where count is the bytes to read

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

Syntax for Device-Level Traditional NI-488.2 Calls in Interactive Control

Syntax	Description
<u>ibask</u> option	Return configuration information where option is a mnemonic for a configuration parameter
<u>ibclr</u>	Clear specified device
<u>ibconfig</u> option value	Alter configurable parameters where option is mnemonic for a configuration parameter
<u>ibdev</u> BdIndx pad sad tmo eot eos	Open an unused device; ibdev parameters are BdIndx pad sad tmo eot eos
<u>ibeos</u> v	Change/disable EOS message
<u>ibeot</u> v	Enable/disable END message
<u>ibloc</u>	Go to local
<u>ibonl</u> v	Place device online or offline
<u>ibpad</u> v	Change primary address
<u>ibpct</u>	Pass control
<u>ibppc</u> v	Parallel poll configure
ibrd count	Read data where count is the bytes to read
<u>ibrda</u> count	Read data asynchronously where count is the bytes to read
<u>ibrdf</u> flname	Read data to file where flname is pathname of file to read
<u>ibrpp</u>	Conduct a parallel poll
ibrsp	Return serial poll byte
<u>ibsad</u> v	Change secondary address
<u>ibstop</u>	Abort asynchronous operation
<u>ibtmo</u> v	Change/disable time limit
ibtrg	Trigger selected device
<u>ibwait</u> mask	Wait for selected event where $mask$ is a hex or decimal

integer or a list of mask bit mne	emonics, such as ibwait
TIMO CMPL	

ibwrt wrtbuf Write data

ibwrta wrtbuf Write data asynchronously

<u>ibwrtf</u> flname Write data from a file where flname is pathname of file to write

Syntax for Multi-Device NI-488.2 Calls in **Interactive Control**

Syntax

AllSpoll addrlist **DevClear** address **DevClearList** addrlist EnableLocal addrlist **EnableRemote** addrlist **FindLstn** padlist limit **FindRQS** addrlist **PassControl** address PPoll **PPollConfig** address dataline lineSense **PPollUnconfig** addrlist <u>RcvRespMsg</u> count termination ReadStatusByte address Receive address count termination Receive data from a device **ReceiveSetup** address **<u>ResetSys</u>** addrlist Send address buffer eotmode SendCmds buffer SendDataBytes buffer eotmode **SendIFC** SendList addrlist buffer eotmode SendLLO SendSetup addrlist **SetRWLS** addrlist

Description

Serial poll multiple devices Clear a device Clear multiple devices Enable local control Enable remote control Find all Listeners Find device asserting SRQ Pass control to a device Parallel poll devices Configure device for parallel poll

Unconfigure device for parallel poll Receive response message Serial poll a device Receive setup Reset multiple devices Send data to a device Send command bytes Send data bytes Send interface clear Send data to multiple devices Put devices in local lockout Send setup Put devices in remote with lockout

TestSRQ TestSys addrlist

<u>Trigger</u> address <u>TriggerList</u> addrlist <u>WaitSRQ</u> state Test for service request Cause multiple devices to perform selftests Trigger a device Trigger multiple devices Wait for service request

Analysis Tools

NI Spy

NI Spy monitors, records, and displays the NI-488.2 calls made from NI-488.2 applications. You can use it to troubleshoot errors in your application and to verify the communication with your GPIB instrument. NI Spy shows which NI-488.2 calls are being used to communicate with your instrument. If your application is not working properly, you can use NI Spy to search for failed NI-488.2 calls. For more information, refer to the NI Spy help.

To start NI Spy, select **Start»Programs»National Instruments»NI Spy**. Or, click on the following button.

Start NI Spy.

GPIB Analyzer

You can use GPIB Analyzer to test and debug your NI-488.2 application by monitoring and capturing GPIB events while your application is running. Analyzing this data can help you solve many of the difficulties associated with GPIB communication, such as addressing inconsistencies, protocol violations, and simple bus timeout conditions. For more information, refer to the GPIB Analyzer help.

To start GPIB Analyzer from within <u>Measurement & Automation Explorer</u>, select **Tools»NI-488.2»GPIB Analyzer** or click on the following button.

Start GPIB Analyzer.

Tip To capture GPIB bus activity, the GPIB Analyzer software requires special National Instruments GPIB Analyzer hardware.

Application Development with NI-488.2

Choosing a Development Environment Language-Specific Programming Instructions General Programming Considerations Advanced Programming Techniques Tools for Developing Your Application

How do I use an NI-488.2 application interface?

Microsoft C/C++ Application Interface Files

A documentation file, readme.txt, that contains information about the C application interface.

An include file, ni4882.h, that contains traditional and multi-device NI-488.2 call prototypes and various predefined constants.

A 32-bit and 64-bit C application interface file, ni4882.obj, that an application links with in order to access the 32-bit DLL and 64-bit DLL.

Borland C/C++ Application Interface Files

A documentation file, readme.txt, that contains information about the C application interface.

An include file, ni4882.h, that contains traditional and multi-device NI-488.2 call prototypes and various predefined constants.

A 32-bit C application interface file, ni4882.obj, that an application links with in order to access the 32-bit DLL.

Microsoft Visual Basic Application Interface Files

A documentation file, readme.txt, that contains information about the Visual Basic application interface.

A 32-bit Visual Basic global module, niglobal.bas, that contains certain predefined constant declarations.

A 32-bit Visual Basic source file, vbib-32.bas, that contains traditional and multi-device NI-488.2 call prototypes.

.NET NI-488.2 Interface

With the Microsoft .NET Framework version 1.1 or later, you can use NI-488.2 to create applications using Visual C# and Visual Basic .NET with or without Measurement Studio.

Refer to the *NI-488.2 .NET Framework Help* for an overview, concepts, and a function reference. To view the help, go to **Start»Programs»National Instruments»NI-488.2**.

The *NI-488.2 .NET Framework Help* is installed if the Microsoft .NET Framework version 1.1 or later is on the system during NI-488.2 installation. The help requires a viewer, available with Visual Studio or other .NET development environments. The help is fully integrated into the Visual Studio .NET documentation, under the NI Measurement Studio section of the help contents.

Other Application Development Environments

If you are using a programming environment that is not listed above, you need to use <u>direct entry</u> to access the NI-488.2 software.

Developing NI-488.2 Applications with LabVIEW, LabWindows/CVI, and Measurement Studio

National Instruments sells two application development environments, LabWindows/CVI and LabVIEW, and a suite of Visual Studio tools, Measurement Studio. All of these products include instrument driver libraries that make it easier to communicate with your GPIB instruments.

LabWindows/CVI is an interactive ANSI C development environment for building test and measurement and instrument control systems. It includes interactive code-generation tools and a graphical editor for building custom user interfaces. It also includes built-in libraries for IEEE 488.2, VXI, RS-232 control, and plug-in data acquisition. When you order LabWindows/CVI, you also get more than 300 complete instrument drivers, which are modular, source-code programs that handle the communication with your instrument so that you do not have to learn the programming details.

LabVIEW is a complete programming environment that departs from the sequential nature of traditional programming languages and features a graphical programming environment. It includes all the tools needed for instrument control, data acquisition, analysis, and presentation. LabVIEW also includes an extensive instrument driver library.

Measurement Studio is an integrated suite of measurement and automation controls, tools, and class libraries for Visual Studio 6.0 and Visual Studio .NET 2003 and later. Measurement Studio dramatically reduces application development time with ActiveX and .NET controls, object-oriented measurement hardware interfaces, advanced analysis libraries, scientific user interface controls, measurement data networking, wizards, interactive code designers, and highly extensible classes.

For more information about Measurement Studio, LabWindows/CVI, and LabVIEW, contact National Instruments.

Direct Entry with C Programming Instructions

Direct entry is available for only the 32-bit gpib-32.dll, and not for ni4882.dll.

gpib-32.dll Exports

Directly Accessing the gpib-32.dll Exports

Compiling Your Win32 C Application

gpib-32.dll Exports

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

int *user_ibsta; int *user_iberr; int *user_ibcnt; long *user_ibcntl;

Except for the calls 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 need to call GetProcAddress passing the name of the function as a parameter. For more information about the parameters that you use when you invoke the function, refer to the help for that function, which you can access through the Index of this help file.

The calls ibfind, ibrdf, and ibwrtf all require an argument that is a name. ibfind requires a board or device name, and ibrdf and ibwrtf take a file name. Because Windows supports both normal (8-bit) characters, and Unicode (16-bit) wide characters, gpib-32.dll exports two versions of each of these functions. An "ASCII" version is for 8-bit characters (ibfindA, ibrdA, ibwrtA) and a "wide" version for 16-bit characters (ibfindW, ibrdW, ibwrtW).

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 calls you use in your application. To see the prototypes for each function exported by gpib-32.dll, refer to the help for that function, which you can access through the Index of this help file. 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 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"{
```

```
#include <windows.h>
#include" "ni488.h"
#ifdef __cplusplus
}
```

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

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

```
return FALSE;
```

}

To see the prototypes for each function, refer to the help for that function, which you can access through the Index of this help file.

For calls that return an integer value, like <u>ibdev</u> or <u>ibwrt</u>, the pointer to the function needs to be cast as:

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

where *Pname is the name of the pointer to the function.

For calls that do not return a value, like <u>FindLstn</u> or <u>SendList</u>, the pointer to the function needs to be cast as:

```
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 help for that function, which you can access through the Index of this help file.

An example of how to cast the function pointer and how the parameter list is set up for *ibdev* and *ibonl* calls follows:

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 calls your application needs. The following code fragment shows you how to get the addresses of the pointers to the status variables and any calls 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 de-reference the pointer to access either the status variable 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 Win32 SDK (Software Development Kit) help.

Compiling Your Win32 C Application

Before you compile your Win32 C application that uses direct entry, include the following lines at the beginning of your program:

#include <windows.h>
#include "ni488.h"

When compiling and linking an application in a DOS shell, you can use the "NIEXTCCOMPILERSUPP" environment variable, which is provided as an alias to the location of C application support files.

Microsoft Visual C/C++ Programming Instructions (Version 6.0 or Later)

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

cl /I"%NIEXTCCOMPILERSUPP%\include" cprog.c

Borland C/C++ Programming Instructions (Version 5.02 or Later)

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

bcc32 -I"%NIEXTCCOMPILERSUPP%\include" -w32 cprog.c

Language-Specific Programming Instructions

Microsoft Visual C/C++ Borland C/C++ Microsoft Visual Basic .NET

Microsoft Visual C/C++ Programming Instructions (Version 6.0 or Later)

Before you compile your application, include the following line at the beginning of your program:

#include "ni4882.h"

The "NIEXTCCOMPILERSUPP" environment variable is provided as an alias to the location of C language support files. You can use this variable when compiling and linking an application.

With Microsoft Visual C++ (Version 6.0 or higher), to compile and link a Win32 console application named cprog in a DOS shell using the environment variable, "NIEXTCCOMPILERSUPP", type in the following on the command line:

cl /I"%NIEXTCCOMPILERSUPP%\include" cprog.c "%NIEXTCCOMPILERSUPP%\lib32\msvc\ni4882.obj" /MD

With Microsoft Visual C++ (Version 8.0 or higher), to compile and link a Win64 console application named cprog in a DOS shell using the environment variable, "NIEXTCCOMPILERSUPP", type in the following on the command line:

cl /I"%NIEXTCCOMPILERSUPP%\include" cprog.c "%NIEXTCCOMPILERSUPP%\lib64\msvc\ni4882.obj" /MD

Borland C/C++ Programming Instructions (Version 5.02 or Later)

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

#include "ni4882.h"

The "NIEXTCCOMPILERSUPP" environment variable is provided as an alias to the location of C language support files. You can use this variable when compiling and linking an application.

To compile and link a Win32 console application named cprog in a DOS shell using the environment variable, "NIEXTCCOMPILERSUPP", type in the following on the command line:

bcc32 -I"%NIEXTCCOMPILERSUPP%\include" -w32 cprog.c "%NIEXTCCOMPILERSUPP%\lib32\borland\ni4882.obj"

Borland/CodeGear does not have a 64-bit compiler at time of writing.

Visual Basic Programming Instructions (Version 6.0)

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 calls. 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.

To see the Visual Basic syntax for any function, refer to the help for that function, which you can access through the Index of this help file.

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

.NET Programming Instructions

Before you start using the NI-488.2 .NET API, you will need to add two assembly references to your C#/VB.NET project:

- NationalInstruments.Common
- NationalInstruments.NI4882

With the Microsoft .NET Framework version 1.1 or later, you can use NI-488.2 to create applications using Visual C# and Visual Basic .NET with or without Measurement Studio. You need Microsoft Visual Studio .NET 2003 or later for the API documentation to be installed.

The installed documentation contains the NI-488.2 API overview, concepts, and function reference. This help is fully integrated into the Visual Studio .NET documentation.

To view the NI-488.2 .NET documentation, go to Start»Programs»National Instruments»[Measurement Studio]»Measurement Studio Documentation. Under Contents, expand NI Measurement Studio Help»NI Measurement Studio .NET Class Library»Reference»NationalInstruments.NI4882 to view the function reference.

Expand NI Measurement Studio Help»NI Measurement Studio .NET Class Library»Using the Measurement Studio .NET Class Libraries to view conceptual topics for using NI-488.2 with Visual C# and Visual Basic .NET.

To get to the same help topics from within Visual Studio .NET 2003 or later, go to **Help**»**Contents** and expand **NI Measurement Studio Help**.

For Visual C# and Visual Basic .NET examples, go to Start»Programs»National Instruments»[Measurement Studio]».NET Examples and follow the GPIB shortcut. Refer to the Where to Find Examples topic in the Measurement Studio documentation under NI Measurement Studio Help for more information. The GPIB examples folder and the examples in it are added when you select the .NET Framework Application Support feature in the NI-488.2 Installer for the version of the .NET Framework you have installed.

General Programming Considerations

Communicating with Your Instrument Using the NI-488.2 API Checking Global Status After Each NI-488.2 Call

Communicating with Your Instrument

Each GPIB instrument has a specific set of commands that you use to program it. The commands are always transferred over the GPIB by the Controller, but the exact command sequences are totally dependent on the particular GPIB instrument. Your GPIB instrument came with documentation that describes the command sequences that you must use to get it to work properly. Refer to that documentation to determine how to communicate correctly with your GPIB instrument.

For simple instrument communication, use the NI-488.2 Communicator. For more information, refer to the <u>Using NI-488.2 Communicator</u> topic.

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 it to learn to communicate with your instruments using the NI-488.2 API. For more information, refer to the Interactive Control Overview topic.

To start Interactive Control within <u>Measurement & Automation Explorer</u>, select **Tools»NI-488.2»Interactive Control** or click on the following button.

Start Interactive Control.

Related Topics:

<u>FindLstn</u> <u>GPIB Addresses</u>

Talkers, Listeners, and Controllers

Using the NI-488.2 API

Choosing a Method to Access the NI-488.2 Driver

Differences Between the GPIB32 API and NI4882 API

Choosing How to Use the NI-488.2 API

Communicating with a Single GPIB Instrument

Communicating with Multiple Instruments and/or Multiple Interfaces

Header Files

Examples

Programming Model for Applications that Communicate with a Single GPIB Instrument

Programming Model for Applications that Communicate with Multiple Instruments and/or Multiple Interfaces

Choosing a Method to Access the NI-488.2 Driver

Applications using the older GPIB32 API can access the NI-488.2 dynamic link library (DLL), gpib-32.dll, either by using an NI-488.2 application interface or by direct access.

Applications using the new NI4882 API can access the NI-488.2 dynamic link library (DLL), ni42882.dll, by using an NI-488.2 application interface only.

NI-488.2 Application Interfaces

You can use an application interface if your program is written in Microsoft Visual C/C++ (6.0 or later), Borland C/C++ (5.02 or later), Microsoft Visual Basic (6.0), or any .NET programming language. Otherwise, you must access gpib-32.dll directly.

For more information about application interfaces, refer to <u>NI-488.2</u> <u>Application Interface Files</u>.

Direct Entry Access

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

For more information about direct entry access, refer to <u>Directly</u> <u>Accessing the gpib-32.dll Exports</u>.

Differences Between the GPIB32 API and NI4882 API

The NI-488.2 for Windows 2.6 release has introduced a new API as part of the 64-bit application interface. Every effort has been made to have the new NI4882 API closely match the existing GPIB32 API while incorporating API design best practices. To use the new API, you must recompile applications using the new header and object files. The following list describes the major changes in the NI4882 API.

- Judicious application of the const keyword has been added where appropriate.
- Wide variants of functions now use the wchar_t instead of unsigned short type.
- Functions taking in parameters that describe a pointer length now use size_t types.
- Status variables now use the unsigned long type.
- ThreadIbcntl() has been removed. Macros redirect calls to ThreadIbcnt().
- Global status functions have been added. These are Ibsta(), Iberr(), and Ibcnt(). New code should use these functions instead of ibsta, iberr, or ibcnt/ibcntl.
- Long-term deprecated functions have been completely removed.
- Most functions with an ibconfig have been removed. Using ibconfig is recommended for new code. Existing functions redirect to using ibconfig using macros. These are the affected functions:
 - ibpad
 - ibsad
 - ibtmo
 - ibeot
 - ibrsc
 - ibsre
 - ibeos
 - ibdma
 - ibist
 - ibrsv

• Many macro definitions have been improved for programmatic safety.

Modifying existing applications to use the NI4882 API should require minimal changes. In most cases, using the new include file (ni4882.h instead of windows.h and ni488.h) and linking to the new object file (ni4882.obj instead of gpib-32.obj) is sufficient to compile your application. There may still be warnings due to changes to the status variable type's signed property.

Complications may arise in several uncommon use cases. The following issues have been encountered.

- Storing function pointers for the ibnotify callback. This causes a type mismatch on the assignment. To solve this, fix the function prototype of the callback to use unsigned long for the status parameters.
- Using function pointers to ibfind. This causes a preprocessor error because the ibfind macro requires a one-parameter argument. To solve this, point to ibfindA or ibfindW, depending on the unicode convention in your application.
- Configuration functions show up in NI Spy as ibconfig calls. This is because macros redirect those calls to use ibconfig. Avoid confusion by using ibconfig directly.

In most cases, applications written in the NI4882 API will continue to work on older versions of the NI-488.2 for Windows software, back to version 1.7. Certain new ibask and ibconfig options break this backwards compatibility, and those options are easily avoidable by using alternative options. Existing applications using the GPIB32 API continue to execute unchanged. The GPIB32 API will continue to exist, but are available only for 32-bit applications. Applications written in the NI4882 API compile on both 32-bit and 64-bit environments. To port an application to a 64-bit environment requires that the application migrate to the NI4882 API and be recompiled.

The following NI4882 API constructs break API compatibility with older versions of NI-488.2 for Windows.

- ibask(IbaEOS)
- ibconfig (IbcEOS)

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 applications 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 or multiple instruments, you should use the multi-device NI-488.2 calls. Whichever you choose, bus management operations necessary for device communication are automatically performed.

Communicating with a Single GPIB Instrument

Communicating with Multiple Instruments and/or Multiple Interfaces

Communicating with a Single GPIB Instrument

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

- Initialization use <u>ibdev</u> to get a handle and use <u>ibclr</u> to clear the instrument.
- **Device Communication**use <u>ibwrt</u>, <u>ibrd</u>, <u>ibtrg</u>, <u>ibrsp</u>, and <u>ibwait</u> to communicate with the instrument.
- **Cleanup**use *ibonl* to put the handle offline.

For a detailed list, refer to the <u>Device-Level Traditional NI-488.2 Calls</u> topic.

For NI-488.2 applications that need to control the GPIB in non-typical ways, for example, to communicate with non-compliant GPIB instruments, there are a set of low-level calls that perform rudimentary GPIB applications. If you use these calls, you need to understand GPIB management details like how to address talkers and listeners. For some details on GPIB management, refer to the Learning More about GPIB topic.

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

- Communicating with non-compliant instruments
- Altering various low-level board configurations
- Managing the bus in non-typical ways

Board-level calls that an NI-488.2 application might use include the following: <u>ibcmd</u>, <u>ibrd</u>, <u>ibwrt</u>, and <u>ibconfig</u>. For a detailed list, refer to the <u>Board-Level Traditional NI-488.2 Calls</u> topic.

Communicating with Multiple Instruments and/or Multiple Interfaces

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

- Find all of the Listeners on the bus using FindListn.
- Find an instrument requesting service using **FindRQS** or **AllSpoll**.
- Determine the state of the SRQ line, or wait for SRQ to be asserted using <u>TestSRQ</u> or <u>WaitSRQ</u>.
- Send data bytes to multiple GPIB instruments using <u>SendList</u>.

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.

Header Files

In a C application, include the header file ni4882.h. The ni4882.h file contains prototypes for the NI-488.2 calls and constants that you can use in your application.

Examples

To run examples for the various NI-488.2 application interfaces, go to the Users\Public\Documents\National Instruments\NI-488.2\Examples directory on Windows Vista and the Documents and Settings\All

Users\Documents\National Instruments\NI-488.2\Examples directory on Windows 2000 or XP. These directories allow users to save changes or add files to the examples.

The directories have Readme files with additional details for compiling and running the examples.

Programming Model for Applications that Communicate with a Single GPIB Instrument

<u>General Program Steps and Examples</u> <u>Items to Include in Your Application</u>

General Program Steps and Examples for Your Traditional NI-488.2 Application

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

First use <u>ibdev</u> 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 your instrument user manual)
- 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 that is used for all devicelevel traditional NI-488.2 calls that communicate with the GPIB instrument.

Step 2. Clear the Device

Use <u>ibclr</u> 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 a "*IDN?" query and then reading back the response. Many devices respond to this query by returning a description of the device. You must 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 *ibrd* 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.

Items to Include in Your Traditional NI-488.2 Application

You should include the following items in your application:

- **Header files**In a C application, include the header file ni4882.h. The ni4882.h file contains prototypes for the NI-488.2 calls and constants that you can use in your application.
- Error checkingCheck 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 instrument offline and closes the application. If the function is declared as: void gpiberr (const char * msg); /*function prototype*/

Then, your application invokes it as follows:

if (Ibsta() & ERR) { gpiberr("GPIB error"); }

Programming Model for Applications that Communicate with Multiple Instruments and/or Multiple Interfaces

General Program Steps and Examples Items to Include in Your Application

General Program Steps and Examples for Your Multi-Device Application

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 <u>SendIFC</u> to initialize the bus and the GPIB interface so 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 Devices

Use <u>FindLstn</u> 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 in the list. 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.

Alternatively, 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 0x67, then the 16-bit value is 0x6701.

Step 3. Initialize the Devices

Use <u>DevClearList</u> 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 a "*IDN?" query and then reading back the responses. Many devices respond to this query by returning a description of the device. You must 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 <u>SendList</u> 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 <u>Receive</u> for each device to read the responses from each device.

Continue communicating with the GPIB devices until you are finished.

Items to Include in Your Multi-Device Application

You should include the following items in your application:

- **Header files**In a C application, include the header file ni4882.h. The ni4882.h file contains prototypes for the NI-488.2 calls and constants that you can use in your application.
- Error checkingCheck 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 instrument offline and closes the application. If the function is declared as:

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

Then, your application invokes it as follows:

```
if (Ibsta() & ERR) {
```

```
gpiberr("GPIB error");
```

```
}
```

Checking Global Status After Each NI-488.2 Call

For applications accessing the NI4882 API, each NI-488.2 call updates three global functions to reflect the status of the device or board that you are using. These global status functions are the status word (Ibsta), the error function (Iberr), and the count function (Ibcnt). They contain useful information about the performance of your application. Your application should check these functions after each NI-488.2 call. For more information about each status function, refer to the following sections.

For applications accessing the older GPIB32 API (including the Visual Basic 6.0 application interface), use the equivalent global variables. These global status variables are the status word (ibsta), the error variable (iberr), and the count variables (ibcnt and ibcntl). ibcnt is defined to be the type int, while ibcntl is the size of type long int. For all cases, if the sizes of ibcnt and ibcntl are the same, ibcnt and ibcntl are equal. For cross-platform compatibility, all applications should use ibcntl.

For applications accessing the newer NI4882 API, use the global function calls rather than the global variables. The global functions replace the global variables with the newer NI4882 API.

M

Note If you are writing a multithreaded application, use the threadspecific copies of the status functions in your application. To access the thread-specific copies, use the <u>ThreadIbsta</u>, <u>ThreadIberr</u>, and <u>ThreadIbcnt</u> calls.

Status Word (Ibsta)

All NI-488.2 calls update a global status word, Ibsta. It contains information about the state of the GPIB and your GPIB hardware. The value stored in Ibsta is the return value of all the traditional NI-488.2 calls, except <u>ibfind</u> and <u>ibdev</u>. You can examine various status bits in Ibsta and use that information to determine what to do next in your application. For more information about the status bits in Ibsta, refer to the <u>Ibsta Status Bit</u> <u>Values</u> table.

The language header file which is installed 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 a GPIB error, use the following statement after each GPIB call:

if (Ibsta() & ERR)

printf("GPIB error encountered");

Error Function (Iberr)

Iberr is the NI-488.2 error function. If a call failed with an error, the ERR bit is set in Ibsta. The Iberr value describes the NI-488.2 error that occurred. For more information about the Iberr values, refer to the Error Codes and Solutions table.

Count Function (lbcnt)

Ibent is the count function. It contains information about the number of bytes that went across the GPIB in the most recent I/O operation.

The count function is updated after each read, write, or command function. In addition, Ibcnt is updated after specific 488.2-style functions in certain error cases.

If the data that you are reading contains ASCII characters, you can use Ibcnt to NULL terminate the string and treat it like any other ASCII string. For example, you can use printf to print the result to the screen:

```
char rdbuf[21];
ibrd (ud, rdbuf, 20);
if (!(Ibsta() & ERR)){
  rdbuf[Ibcnt()] = '\0';
  printf ("Read in string: %s\n", rdbuf);
}
else {
  // GPIB Error encountered!
}
```

Advanced Programming Techniques

Serial Polling Terminating Data Transfers Waiting for GPIB Conditions Multithreaded NI-488.2 Applications Device-Level NI-488.2 Calls and Bus Management Talker/Listener Applications High-Speed Data Transfer (HS488) Asynchronous Event Notification Parallel Polling

Serial Polling

Overview Service Requests From Devices Serial Polling with Traditional NI-488.2 Calls Serial Polling with Multi-Device NI-488.2 Calls Automatic Serial Polling Overview Stuck SRQ State Autopolling and Interrupts

Serial Polling Overview

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.

Related Topics:

AllSpoll Automatic Serial Polling Overview FindRQS IBRSP IBRSV Serial Polling with Device-Level Traditional NI-488.2 Calls Serial Polling with Multi-Device NI-488.2 Calls Service Requests from Devices

Service Requests From Devices

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.

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.

Related Topics:

AllSpoll Automatic Serial Polling Overview FindRQS IBRSP IBRSV Serial Polling Overview Serial Polling with Device-Level Traditional NI-488.2 Calls

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 function, 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 calls in a typical SRQ servicing situation when automatic serial polling is enabled.

```
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;
}
```

Related Topics:

AllSpoll Automatic Serial Polling Overview FindRQS IBRSP IBRSV Serial Polling Overview Serial Polling with Multi-Device NI-488.2 Calls Service Requests from Devices

Serial Polling with Multi-Device NI-488.2 Calls

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

- 1. <u>AllSpoll</u> 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.
- 2. <u>ReadStatusByte</u> is similar to AllSpoll, except that it only serial polls a single device. It is also similar to the device-level traditional NI-488.2 call <u>ibrsp</u>.
- 3. <u>FindRQS</u> 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 call returns the index and status byte value of the device requesting service.
- 4. <u>TestSRQ</u> determines whether the SRQ line is asserted, and returns to the program immediately.
- 5. <u>WaitSRQ</u> 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
- Example 2: Using AllSpoll

Related Topics:

<u>AllSpoll</u>

Automatic Serial Polling Overview

FindRQS IBRSP IBRSV Serial Polling Overview Serial Polling with Device-Level Traditional NI-488.2 Calls Service Requests from Devices

Example 1: Using FindRQS

This example shows you how to use **<u>FindRQS</u>** 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 <u>AllSpoll</u> to serial poll three devices with a single call.

```
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;
}
```

Automatic Serial Polling Overview

If you want your application to conduct a serial poll automatically when the SRQ line is asserted, you can enable automatic serial polling. You can use automatic serial polling with traditional device-level calls only. The autopolling procedure occurs as follows:

- 1. Autopolling is enabled by default. However, if you want to disable autopolling, use the configuration function, <u>ibconfig</u>, with the IbcAUTOPOLL option, or <u>Measurement & Automation Explorer</u>.
- 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, <u>lbsta</u>, 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. You should empty the queue as soon as an automatic serial poll occurs.
- 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, you should continue to call ibrsp until RQS is cleared.

Related Topics:

AllSpoll FindRQS IBRSP IBRSV Serial Polling Overview Serial Polling with Device-Level Traditional NI-488.2 Calls Serial Polling with Multi-Device NI-488.2 Calls Service Requests from Devices

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 <u>ibwait</u> 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.

Related Topics:

Automatic Serial Polling Overview Autopolling and Interrupts

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 <u>ibwait</u> for RQS is in progress. Autopolling is not allowed when an application calls board-level traditional or multi-device NI-488.2 calls, or the stuck SRQ (ESRQ) condition occurs.

Related Topics:

Automatic Serial Polling Overview Stuck SRQ State

Terminating 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-ofstring (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 *ibconfig* function (option IbcEOT) 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 *ibconfig* function (option IbcEOS) 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 methodIf 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 methodIf 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 <u>ibconfig</u> 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 <u>Ibsta</u> 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.

Waiting for GPIB Conditions

You can use the <u>ibwait</u> function to obtain the current <u>Ibsta</u> 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 <u>TIMO</u> event; otherwise, your application is suspended indefinitely until one of the wait mask events occurs.

You can also wait for GPIB conditions asynchronously. For more information, refer to the <u>Asynchronous Event Notification</u> topic.

Device-Level NI-488.2 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.

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.

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 <u>ibwait</u> with a mask of 0 to monitor the status of the interface. Then, based on the status bits set in <u>Ibsta</u>, 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.

High-Speed Data Transfer (HS488)

Overview Enabling HS488 Transfers System Configuration Effects on HS488

HS488 Overview

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 part of the IEEE 488.1 2003 specification; thus, you can mix IEEE 488.1, IEEE 488.2, and HS488 devices in the same system. If HS488 is enabled, National Instruments GPIB hardware implements high-speed transfers automatically when communicating with HS488 instruments. If you attempt to enable HS488 on a GPIB interface that does not support it, the ECAP error code is returned.

Related Topics:

Enabling HS488 Transfers
System Configuration Effects on HS488

Enabling HS488 Transfers

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 NI-488.2 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 <u>device-level traditional NI-488.2 calls</u>, the NI-488.2 software automatically sends the HS488 configuration message to devices. If you set the HS488 cable length in <u>Measurement & Automation</u> <u>Explorer</u> to a non-zero value, the NI-488.2 software sends out the HS488 configuration message when you use <u>ibdev</u> to bring a device online. If you call ibconfig to change the GPIB cable length, the NI-488.2 software sends out the HS488 configuration message again, the next time you call a device-level function.

If you are using <u>board-level traditional NI-488.2 calls</u> or <u>multi-device NI-488.2 calls</u> and you want to configure devices for high-speed, you must send the HS488 configuration messages using <u>ibcmd</u> or <u>SendCmds</u>. 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 meters, only CFGn values of 1 through 15 (hex 61 through 6F) are valid. If the cable length was configured properly in Measurement & Automation Explorer, you can determine how many meters of cable are in your system by calling <u>ibask</u> (option IbaHSCableLength) in your application. For CFE and CFGn messages, refer to the <u>IEEE 488 Command Messages</u> section.

Related Topics:

<u>HS488 Overview</u> <u>System Configuration Effects on HS488</u>

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 PCcompatible computer with PCI bus, the maximum obtainable transfer rate is 8 Mbytes/s, but when using another bus, such as USB or Ethernet, the maximum data transfer rate depends on the maximum transfer rate of that bus.

The same IEEE 488 cabling constraints for a 350 ns T1 delay apply to HS488. As you increase the amount of cable in your NI-488.2 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 four meters of cable.

Related Topics:

Enabling HS488 Transfers HS488 Overview

Asynchronous Event Notification

Overview ibnotify Usage GpibNotify Usage

Asynchronous Event Notification Overview

NI-488.2 applications can asynchronously receive event notifications using the <u>ibnotify</u> 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.

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 value except RQS. Note that the ibnotify mask bits are identical to the <u>ibwait</u> 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.

For more information about usage, please refer to either the <u>ibnotify</u> <u>Usage</u> or the <u>GpibNotify Usage</u> section. For more specific information about ibnotify, refer to the <u>ibnotify</u> function definition. For more information about the <u>GpibNotify</u> OLE control, refer to the <u>GpibNotify</u> function definition.

ibnotify Usage

The ibnotify function is passed a unit descriptor, the bit mask of the desired GPIB events, the address of your callback function, and user-defined reference data. ibnotify has the following prototype:

unsigned long ibnotify (int ud, int mask, GpibNotifyCallback_t Callback, void * RefData)

С

Visual Basic (6.0)

CALL ibnotify (ud%, mask%, AddressOf Callback&, RefData\$)

or

status% = ilnotify (ud%, mask%, AddressOf Callback&, RefData\$)

Input for ibnotify

ud Board or device descriptor

mask Bit mask of GPIB events to notice

Callback Pointer to the Callback function

RefData User-defined reference data for the callback

The ibnotify callback has the following prototype:

С

int __stdcall Callback (int LocalUd, unsigned long LocalIbsta, unsigned long LocalIberr, unsigned long LocalIbcnt, void *RefData)

Visual Basic (6.0)

Function Callback (LocalUd%, LocalIbsta%, LocalIberr%, LocalIbcnt&, RefData\$)

Callback Parameters

LocalUd	Board or device descriptor
LocalIbsta	Value of <u>Ibsta</u>
LocalIberr	Value of <u>Iberr</u>
LocalIbcnt	Value of Ibcnt
RefData	User-defined reference data for the callback

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.

You can view an <u>example written in C of how you can use ibnotify</u> in your application. Assume that your GPIB device is a multimeter that you program to acquire a reading by sending it "SEND DATA". The multimeter requests service when it has a reading ready, and each reading is a floating point value.

In this example, global functions/variables are shared by the Callback thread and the main thread, and the access of the global functions/variables is not protected by synchronization. In this case, synchronization of access to these global functions/variables 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 always just adds information (increases the count or adds another reading to the array of floats).

For overview information about asynchronous event notification within an NI-488.2 application, please refer to the <u>Asynchronous Event Notification</u> <u>Overview</u> section. For more specific information about ibnotify, please refer to the <u>ibnotify</u> function definition.

Note The ibnotify Callback is executed in a separate thread of execution from the rest of your application. If your application might be performing other NI-488.2 operations while it is using ibnotify, you should use the per-thread NI-488.2 global functions/variables that are provided by the **Thread** calls (<u>Threadlbsta</u>, <u>Threadlberr</u>, and <u>Threadlbcnt</u>). In addition, if your application needs to share global functions/variables with the **Callback**, you should use a synchronization primitive (for example, semaphore) to protect access to any global variables. For more information about the use of synchronization primitives, refer to the documentation on using synchronization objects that came with your development tools.

Related Topics:

ibnotify Usage Example ibnotify GpibNotify GpibNotify Usage

GpibNotify Usage

The GpibNotify OLE control is implemented using a method called SetupNotify and an event called Notify. The SetupNotify method is used to enable the NI-488.2 driver to look for one or more GPIB conditions for a particular GPIB handle. After it is set up, the OLE control fires the Notify event when one or more of the GPIB conditions is TRUE. A user-defined callback is invoked when the Notify event is fired.

This section covers the major highlights regarding the sample program that uses the GpibNotify control. The program contains three buttons: **Run**, **Message**, and **Quit**.

Clicking the **Run** button sets into motion a chain of commands that read ten measurements from a Fluke 45 multimeter. First, the program gets a handle to the device. Next, it sends a set of commands that initialize the Fluke 45 multimeter. Then a trigger command is sent. Next the program asks the device to send data. Lastly, it issues a SetupNotify to the GpibNotify OLE control with a mask of the RQS GPIB condition.

When the RQS GPIB condition is TRUE, the Notify event is fired and the user-defined callback is invoked. Each time through the callback, the RearmMask is set to RQS so that the event notification is rearmed for the next RQS GPIB condition. After the callback has read ten measurements from the Fluke 45 multimeter, the RearmMask is set to zero in order to disable the event notification mechanism.

Clicking the **Message** button causes a message to be displayed in a text box every time the button is clicked.

Clicking the **Quit** button closes the program.

The NI-488.2 software includes a sample Visual Basic program. Please refer to the readme.txt file for more detailed information about the project, classes, and code.

For overview information about asynchronous event notification within an NI-488.2 application, please refer to the <u>Asynchronous Event Notification</u> <u>Overview</u> section. For more specific information about the GpibNotify OLE control, please refer to the <u>GpibNotify</u> function definition.

 $\overline{\mathbb{N}}$

Note The GpibNotify OLE control is implemented using the apartment model. Therefore, it only works correctly if your application responds to Windows messages in a timely fashion.

Parallel Polling

Overview Parallel Polling with Traditional NI-488.2 Calls Parallel Polling with Multi-Device NI-488.2 Calls

Parallel Polling Overview

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.

You can implement parallel polling with either the <u>traditional</u> or <u>multi-</u> <u>device NI-488.2 calls</u>. 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).

Related Topics:

IBIST IBPPC IBRPP Parallel Polling with Multi-Device NI-488.2 Calls Parallel Polling with Traditional NI-488.2 Calls PPoll PPollConfig PPollUnconfig

Parallel Polling with Traditional NI-488.2 Calls

To implement parallel polling using traditional NI-488.2 calls, complete the following steps. Each step contains example code.

1. Configure the device for parallel polling using the <u>ibppc</u> 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 <u>ibdev</u> 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 "ni4882.h"

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 board index or a board 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 <u>ibist</u> 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 <u>ibrpp</u> 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.

char ppr; 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);

Related Topics:

<u>IBIST</u> IBPPC IBRPP Parallel Polling overview Parallel Polling with Multi-Device NI-488.2 Calls PPoll PPollConfig

<u>PPollUnconfig</u>

Parallel Polling with Multi-Device NI-488.2 Calls

To implement parallel polling using multi-device NI-488.2 calls, complete the following steps. Each step contains example code.

 Configure the device for parallel polling using the PPollConfig call, 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 "ni4882.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 the PPoll call, 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 the

PPollUnconfig call, 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);

Related Topics:

<u>IBIST</u>

IBPPC

IBRPP

Parallel Polling Overview

Parallel Polling with Traditional NI-488.2 Calls

<u>PPoll</u>

PPollConfig

PPollUnconfig

Tools for Developing Your Application

Monitoring NI-488.2 Calls Using NI Spy Simple Instrument Communication Using NI-488.2 Communicator

Using NI Spy

NI Spy monitors, records, and displays the NI-488.2 calls made from applications. You can use it to troubleshoot errors in your application and to verify the communication with your GPIB instrument. NI Spy shows which NI-488.2 calls are being used to communicate with your instrument. If your application is not working properly, you can use NI Spy to search for failed NI-488.2 calls. For more information, refer to its help.

To start NI Spy, select **Start»Programs»National Instruments»NI Spy**. Or click on the following button.

Start NI Spy.

Using NI-488.2 Communicator

You can use the NI-488.2 Communicator to verify that you can establish simple communication with your GPIB instrument. The NI-488.2 Communicator is an interactive utility that allows you to write commands to your instrument and read responses back from your instrument. It provides detailed information about the status of the NI-488.2 calls and you can use it to print sample C source code that performs a simple query to a GPIB instrument.

To start NI-488.2 Communicator, complete the following steps:

- 1. > Start Measurement & Automation Explorer.
- 2. In Measurement & Automation Explorer, expand the **Devices** and Interfaces directory by clicking on the + next to the folder.
- 3. Right-click on the GPIB interface that your GPIB instrument is connected to.
- 4. Select **Scan for Instruments** from the drop-down menu that appears.
- 5. After the scan is complete, select your GPIB instrument in the left pane of the Measurement & Automation Explorer window.
- 6. Right-click on your instrument and select **Communicate with Instrument** from the drop-down menu that appears. The **NI-488.2 Communicator** dialog box appears.
- 7. Type a command in the **Send String** field and do one of the following:
 - To write a command to the instrument then read a response back, click on the **Query** button.
 - To write a command to the instrument, click on the **Write** button.
 - To read a response from the instrument, click on the **Read** button.
 - To configure special requirements for end of string (EOS) modes for your device, click on the Configure EOS button.

Troubleshooting Problems

If you cannot communicate with your instrument, use the NI-488.2 Troubleshooting Utility. This utility verifies that your GPIB hardware and the NI-488.2 software are installed correctly and working properly.

To start the NI-488.2 Troubleshooting Utility within <u>Measurement &</u> <u>Automation Explorer</u>, select Help»Troubleshooting»NI-488.2 Troubleshooting Utility or click on the following button.

Start the NI-488.2 Troubleshooting

Utility.

Otherwise, refer to the following topics:

Troubleshooting Tools

Troubleshooting EDVR Error Conditions

Why Can't I Communicate with My GPIB Instrument?

Why Can't I Find My GPIB-ENET/100?

Debugging an NI-488.2 Application

Frequently Asked Questions

Troubleshooting EDVR Error Conditions

In some cases, NI-488.2 calls may return with the ERR bit set in <u>Ibsta</u> and the value EDVR in <u>Iberr</u>. The value stored in Ibert is useful in troubleshooting the error condition. When Ibert is set to one of the following values, you can troubleshoot an EDVR error as follows:

0xE014002C (-535560148)

0xE0140025 (-535560155)

0xE0140035 (-535560139)

0xE1080080 (-519569280) or 0xE1080081 (-519569279)

<u>0xE00A0047 (-536215481)</u>

0XE1030043 (-519897021)

0XE1060075 (-519700363)

0XE1060078 (-519700360)

EDVR Error with Ibcnt Set to 0xE014002C (-535560148)

If a call is made with a board number that is within the range of allowed board numbers, but which has not been assigned to a GPIB interface, an EDVR error condition occurs with Ibcnt set to 0xE014002C. You can assign a board 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 Measurement & Automation Explorer topic.

EDVR Error with Ibcnt Set to 0xE0140025 (-535560155)

If a call is made with a board number that is not within the range of allowed board numbers, an EDVR error condition occurs with Ibcnt set to 0xE0140025.

EDVR Error with Ibcnt 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 <u>Measurement & Automation Explorer</u>, an EDVR error condition occurs with Ibcnt set to 0xE0140035.

EDVR Error with Ibcnt Set to 0xE1080080 (-519569280) or 0xE1080081 (-519569279)

These errors are returned if you are using a removable interface (for example, a GPIB-USB-HS) and you removed or ejected the interface while the software is trying to communicate with it.

EDVR Error with Ibcnt Set to 0xE00A0047 (-536215481)

This error is returned when the driver encounters an access violation when attempting to access an object supplied by the user. This can happen if the user's buffer does not have appropriate read/write characteristics. For example, this error is returned if a required pointer passed to a call is NULL.

EDVR Error with Ibcnt Set to 0xE1030043 (-519897021)

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

EDVR Error with Ibcnt Set to 0xE1060075 (-519700363)

This error is returned when the driver is unable to communicate with a GPIB-ENET/100 during an <u>ibfind</u> or <u>ibdev</u> call for one of the following reasons:

- The IP address is incorrect
- The box is not powered on
- The box did not pass its self-diagnostic test
- In general, the box is not configured or installed correctly

EDVR Error with Ibcnt Set to 0xE1060078 (-519700360)

This error occurs if you are using a GPIB-ENET/100 and the network link is broken between the host and the GPIB-ENET/100 interface. This error can be returned for one of the following reasons:

- The cables are broken or removed.
- The GPIB-ENET/100 is switched off during communication.
- The virtual connection between your host computer and the GPIB-ENET/100 has encountered unrecoverable errors.

Why Can't I Communicate with My GPIB Instrument?

The answer to this question depends on the type of behavior you are seeing. The following topics describe some common problems and their solutions. You may be trying to communicate with your instrument from Measurement & Automation Explorer (MAX), using Interactive Control (IBIC), LabVIEW, LabWindows/CVI, or another application program. Usually, to establish basic communication, you are trying to write to your instrument and then read some useful data from that instrument.

Cannot Find Your Instrument During a Scan for Instruments

Error on a Write

Error on a Read and No Data is Returned

Cannot Find Your Instrument During a Scan for Instruments

The <u>Scan for GPIB Instruments</u> functionality of Measurement & Automation Explorer can be helpful in quickly testing your GPIB setup. Clicking this button in Measurement & Automation Explorer conducts a FindLstn test for all GPIB addresses 031 to find active listeners on the bus. Measurement & Automation Explorer attempts to identify these instruments by sending them *IDN? queries and then performing reads. While not all instruments respond to the *IDN? query, most should be detected by the initial FindLstn test.

If your instruments are not detected during this scan:

- Check your cabling. GPIB has strict distance requirements. (Refer to <u>Configuration Requirements</u>.) Make sure the cable is securely connected to both your instrument and your GPIB interface. Try another cable if possible, especially a shorter one.
- If more than one instrument is connected, disconnect all but one instrument. An instrument with a bad GPIB interface could prevent other instruments from communicating on the bus.
- Remember that GPIB address 0 is usually reserved for the controller (National Instruments GPIB boards default to address 0), so if your controller has an address of 0, make sure your instrument has a different address. You must set the address of your instrument on its front panel or via dip switches somewhere on the instrument. Refer to the instrument documentation to set the GPIB address.

Error on a Write

You may be experiencing an NI-488.2 error on a write.

- For EABO and other write errors, make sure you are writing to the correct GPIB address. You can configure most instruments for any GPIB address 031. In Windows, the Scan for Instruments functionality of Measurement & Automation Explorer usually returns the valid address for your instrument. This is especially useful if you are unsure of its current setting.
- An <u>EBUS</u> error indicates there is no GPIB cable connected or no active devices on that cable, so make sure the cable is securely connected and all devices are powered on.
- Remember that GPIB address 0 is usually reserved for the controller (National Instruments GPIB boards default to address 0). If your controller has an address of 0, make sure your instrument has a different address. Refer to the instrument documentation to set the GPIB address.
- If you are using advanced board-level calls and not device-level calls such as ibdev, make sure your GPIB program is correctly addressing the instrument. When using board handles, you must address the device as a listener and the board as a talker before writing data.

Error on a Read and No Data is Returned

If you wrote to your instrument without the write returning an error, you can reasonably assume your written message was properly sent to the instrument. If you wrote a command that the instrument responds to, you should perform a read (ibrd function call). If you receive an error on a read, it most likely is an EABO (abort) due to a TIMO (timeout) condition. This indicates the GPIB controller board was waiting for a response from the instrument, but the response never came.

To correct this error, check the following items:

- The command you previously wrote to your instrument may not be valid. If you wrote an invalid command, the instrument may not respond with anything at all. Refer to your instrument user manual or contact the manufacturer for a simple command the instrument will respond to. Many instruments respond to *IDN? or ID, but others do not support these basic identification commands.
- The termination method used on your write string may be incorrect. There are two main ways to terminate data messages sent to instruments: assert the EOI (End or Identify) line on the GPIB with the last byte of the transmission, or send an EOS (End of String) character at the end. The default termination for the National Instruments GPIB driver software is to assert the EOI line with the last character sent on GPIB writes. If your device requires a termination character, add it to the end of the string you write to the instrument. You can change the EOI setting with the <u>ibconfig</u> call, or in <u>Measurement & Automation Explorer</u> on Windows or GPIB Explorer on other operating systems.

Refer to <u>Iberr Error Codes and Solutions</u> for a complete list of GPIB error codes and their common causes. If you continue to have difficulty communicating with your instrument, contact <u>National Instruments</u> <u>technical support</u>.

Why Can't I Find My GPIB-ENET/100?

Several situations might prevent the Add GPIB-ENET/100 Wizard from discovering your GPIB-ENET/100 hardware.

The GPIB-ENET/100 is Not Properly Connected

Make sure the GPIB-ENET/100 is plugged in, connected to the network, and switched on. Both the PWR/RDY and LINK 10/100 lights on the front of the GPIB-ENET/100 must be lit before the device can be discovered. The PWR/RDY light on the front of the unit indicates whether it is powered on, and a green or amber LINK 10/100 light indicates that the network is properly connected.

The LINK 10/100 light will not turn on if the Ethernet cable you are using is damaged, or if the wrong Ethernet cable type (that is, standard instead of crossover) is being used.

The GPIB-ENET/100 is Outside the Subnet Boundary

NI-488.2 uses broadcast packets to discover GPIB-ENET/100 units, but most routers do not propagate these packets across subnet boundaries. If the GPIB-ENET/100 is not connected to the computer's subnet, NI-488.2 cannot discover it.

To ensure the computer running NI-488.2 and the GPIB-ENET/100 are on the same subnet, connect both of them to the same Ethernet hub. Your network administrator can also help you identify your subnet boundary and connect the GPIB-ENET/100 inside it.

Although only a computer on the same subnet can discover the GPIB-ENET/100 or change its network settings, any computer can use it as a GPIB interface. To add a GPIB-ENET/100 that is outside the computer's local subnet, specify the hostname or IP address in the Add GPIB-ENET/100 Wizard.

A Firewall is Blocking Communication

Firewalls can block traffic that needs to pass between NI-488.2 and a GPIB-ENET/100. Refer to the firewall documentation for instructions on opening ports. The following ports should be open:

- TCP 5000, 5003, 5005, 5010, 5015, and 44516
- UDP 1024 through 5000 and 44515

Debugging an NI-488.2 Application

Determining Whether an NI-488.2 Call Failed Checking Global Status Variables to Debug Your Application Debugging Existing Applications Solving Errors

Determining Whether an NI-488.2 Call Failed

To determine whether an NI-488.2 call failed, you can <u>check the global</u> <u>status functions</u>. You can also use NI Spy to monitor the NI-488.2 calls made from NI-488.2 applications. If your application is not working properly, you can use NI Spy to search for failed NI-488.2 calls. For more information, refer to its help.

To start NI Spy, select **Start»Programs»National Instruments»NI Spy**. Or, click on the following button.

Start NI Spy.

Checking Global Status Functions to Debug Your Application

At the end of each NI-488.2 call, the global status functions (Ibsta, Iberr, and Ibcnt) are updated. If you are developing an NI-488.2 application, you should check for errors after each NI-488.2 call. If an NI-488.2 call fails, 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 Ibcnt contains even more error information. For more information about the global status functions, refer to the <u>Checking Global Status After</u> <u>Each NI-488.2 Call</u> topic.

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 the <u>Ibsta Status Bit Values</u> and the <u>Error Codes and Solutions</u> topics to understand why the NI-488.2 call failed.

Start NI Spy.

Checking Global Status After Each NI-488.2 Call

For applications accessing the NI4882 API, each NI-488.2 call updates three global functions to reflect the status of the device or board that you are using. These global status functions are the status word (Ibsta), the error function (Iberr), and the count function (Ibcnt). They contain useful information about the performance of your application. Your application should check these functions after each NI-488.2 call. For more information about each status function, refer to the following sections.

For applications accessing the older GPIB32 API (including the Visual Basic 6.0 application interface), use the equivalent global variables. These global status variables are the status word (ibsta), the error variable (iberr), and the count variables (ibcnt and ibcntl). ibcnt is defined to be the type int, while ibcntl is the size of type long int. For all cases, if the sizes of ibcnt and ibcntl are the same, ibcnt and ibcntl are equal. For cross-platform compatibility, all applications should use ibcntl.

For applications accessing the newer NI4882 API, use the global function calls rather than the global variables. The global functions replace the global variables with the newer NI4882 API.

M

Note If you are writing a multithreaded application, use the threadspecific copies of the status functions in your application. To access the thread-specific copies, use the <u>ThreadIbsta</u>, <u>ThreadIberr</u>, and <u>ThreadIbcnt</u> calls.

Status Word (Ibsta)

All NI-488.2 calls update a global status word, Ibsta. It contains information about the state of the GPIB and your GPIB hardware. The value stored in Ibsta is the return value of all the traditional NI-488.2 calls, except <u>ibfind</u> and <u>ibdev</u>. You can examine various status bits in Ibsta and use that information to determine what to do next in your application. For more information about the status bits in Ibsta, refer to the <u>Ibsta Status Bit</u> <u>Values</u> table.

The language header file which is installed 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 a GPIB error, use the following statement after each GPIB call:

if (Ibsta() & ERR)

printf("GPIB error encountered");

Error Function (Iberr)

Iberr is the NI-488.2 error function. If a call failed with an error, the ERR bit is set in Ibsta. The Iberr value describes the NI-488.2 error that occurred. For more information about the Iberr values, refer to the Error Codes and Solutions table.

Count Function (lbcnt)

Ibent is the count function. It contains information about the number of bytes that went across the GPIB in the most recent I/O operation.

The count function is updated after each read, write, or command function. In addition, Ibcnt is updated after specific 488.2-style functions in certain error cases.

If the data that you are reading contains ASCII characters, you can use Ibcnt to NULL terminate the string and treat it like any other ASCII string. For example, you can use printf to print the result to the screen:

```
char rdbuf[21];
ibrd (ud, rdbuf, 20);
if (!(Ibsta() & ERR)){
  rdbuf[Ibcnt()] = '\0';
  printf ("Read in string: %s\n", rdbuf);
}
else {
  // GPIB Error encountered!
}
```

Ibsta() or ibsta Status Bit Values

All calls update a global status function, Ibsta, which contains information about the state of the GPIB and your GPIB hardware. 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. When using the GPIB32 API, ibsta is the global variable.

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). Ibsta is a 32-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.

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

ATN Status Condition

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.

CIC Status Condition

CIC indicates whether the GPIB interface is the Controller-In-Charge. CIC is set when the <u>SendIFC</u> or <u>ibsic</u> call 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.

CMPL Status Condition

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

DCAS Status Condition

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

If you use the <u>ibwait</u> or <u>ibnotify</u> 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 <u>ibwrt</u> or <u>Send</u>, and DCAS is set in Ibsta, the I/O operation is aborted. DCAS is cleared from Ibsta after the next NI-488.2 call.

DTAS Status Condition

DTAS indicates whether the GPIB interface has detected a device trigger command. DTAS is set when 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 or ibnotify call, if the DTAS bit is set in the ibwait mask parameter.

END Status Condition

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 operationEOI 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.

ERR Status Condition

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 function Iberr. To view the GPIB error codes, refer to the <u>Error codes and</u> <u>Solutions</u> topic. ERR is cleared following any call that does not result in an error.

LACS Status Condition

LACS indicates whether the GPIB interface is addressed as a Listener. LACS is set when 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 when the GPIB interface shadow handshakes as a result of the ibgts function. LACS is cleared when 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.

LOK Status Condition

LOK indicates whether the board is in a lockout state. While LOK is set, the <u>EnableLocal</u> or <u>ibloc</u> call is inoperative for that board. LOK is set when 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 Status Condition

REM indicates whether the board is in the remote state. REM is set when 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 <u>ibloc</u> function is called while the LOK bit is cleared in the status word

RQS Status Condition

RQS appears in the status word only after a device-level call and indicates that the device is requesting service. RQS is set when 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 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.

SRQI Status Condition

SRQI indicates that a GPIB device is requesting service. SRQI is set when the GPIB interface is CIC and the GPIB SRQ line is asserted. SRQI is cleared either when the GPIB interface ceases to be the CIC or when the GPIB SRQ line is unasserted.

TACS Status Condition

TACS indicates whether the GPIB interface is addressed as a Talker. TACS is set when 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 when 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).

TIMO Status Condition

TIMO indicates that the timeout period has been exceeded. TIMO is set in the status word following an <u>ibwait</u> or <u>ibnotify</u> call if the TIMO bit of the mask parameter is set and the time limit expires. TIMO is also set following any synchronous I/O calls (for example, <u>ibcmd</u>, <u>ibrd</u>, <u>ibwrt</u>, <u>Receive</u>, <u>Send</u>, and <u>SendCmds</u>) if a timeout occurs during one of these calls. TIMO is cleared in all other circumstances.

Debugging 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.

The NI Spy utility monitors NI-488.2 calls made by NI-488.2 applications. It records NI-488.2 input and output values for all NI-488.2 applications. For more information about NI Spy, refer to its help.

To start NI Spy, select **Start»Programs»National Instruments»NI Spy**. Or, click on the following button.

Start 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. When you determine which NI-488.2 call failed, refer to the <u>Ibsta Status Bit</u> <u>Values</u> and the <u>Error Codes and Solutions</u> topics to understand why the NI-488.2 call failed.

Solving Errors

Iberr Error Codes and Solutions Configuration Errors Communication Errors Timing Errors Other Errors

Iberr() or iberr Error Codes and Solutions

Iberr is the NI-488.2 error function. If a call failed within an error, the Iberr value describes the NI-488.2 error that occurred. When using the GPIB32 API, iberr is the global variable.

The following table lists the NI-488.2 error codes. Remember that the error function is meaningful only when the ERR bit in the status function, <u>Ibsta</u>, is set. For a detailed description of each error and possible solutions, click on the error mnemonic.

Mnemonic Value Meaning

,		
<u>EDVR</u>	0	System error
<u>ECIC</u>	1	Function requires GPIB interface to be CIC
<u>ENOL</u>	2	No Listeners on the GPIB
<u>EADR</u>	3	GPIB interface not addressed correctly
<u>EARG</u>	4	Invalid argument to function call
<u>ESAC</u>	5	GPIB interface not System Controller as required
<u>EABO</u>	6	I/O operation aborted (timeout)
<u>ENEB</u>	7	Nonexistent GPIB interface
<u>EDMA</u>	8	DMA error
<u>EOIP</u>	10	Asynchronous I/O in progress
<u>ECAP</u>	11	No capability for operation
<u>EFSO</u>	12	File system error
<u>EBUS</u>	14	GPIB bus error
<u>ESRQ</u>	16	SRQ stuck in ON position
<u>ETAB</u>	20	Table problem
<u>ELCK</u>	21	Interface is locked
<u>EARM</u>	22	ibnotify callback failed to rearm
<u>EHDL</u>	23	Input handle is invalid
<u>EWIP</u>	26	Wait in progress on specified input handle

- ERST 27 The event notification was cancelled due to a reset of the interface
- **EPWR** 28 The interface lost power

EABO Error Code

EABO indicates that an I/O operation has been canceled, usually due to a timeout condition. Other causes are calling <u>ibstop</u> 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

Use the correct byte count in input calls 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 the IbcTMO option for <u>ibconfig</u>.

Make sure that you have configured your device to send data before you request data.

Return to list of error codes

EADR Error Code

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

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

Make sure that the GPIB interface is <u>addressed correctly</u> before calling <u>ibrd</u>, <u>ibwrt</u>, <u>RcvRespMsg</u>, or <u>SendDataBytes</u>.

Avoid calling <u>ibgts</u> except immediately after an <u>ibcmd</u> call. (ibcmd causes ATN to be asserted.)

-

Return to list of error codes

EARG Error Code

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

- **ibconfig** with the IbcTMO option called with a value not in the range 0 through 17.
- **ibconfig** with the IbcEOS option called with meaningless bits set in the high byte of the second parameter.
- **ibconfig** with the IbcPAD or IbcSAD option called with invalid addresses.
- <u>ibppc</u> called with invalid parallel poll configurations.
- A multi-device NI-488.2 call made with an invalid address.
- **PPollConfig** called with an invalid data line or sense bit.

Solutions

Make sure that the parameters passed to the NI-488.2 call are valid.

Return to list of error codes

EARM Error code

EARM indicates that ibnotify's asynchronous event notification mechanism failed to rearm itself. This generally occurs when an <u>ibnotify</u> Callback has attempted to rearm itself by returning an illegal value or when a fatal driver error (EDVR) has occurred.

Ensure that the value being returned by your Callback function is a valid ibnotify mask value.

Return a zero value from your Callback function to unregister the asynchronous event notification mechanism. Then call ibnotify to reenable notification.

EBUS Error Code

EBUS results when certain GPIB bus errors occur during device calls. All device calls 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 IbcTMO option for ibconfig. EBUS results if a timeout occurred while sending these command bytes.

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.

ECAP Error Code

ECAP results when your GPIB interface cannot carry out an operation or when a particular capability has been disabled in the software and a call is made that requires the capability.

Check the validity of the call.

Make sure your GPIB interface and the driver both have the needed capability.

-

ECIC Error Code

ECIC is returned when one of the following calls is made while the board is not CIC:

- Any device-level traditional NI-488.2 calls that affect the GPIB
- Any <u>board-level traditional NI-488.2 calls</u> that issue GPIB command bytes (ibcmd, ibcmda, ibln, and ibrpp)
- ibcac and ibgts
- Any <u>multi-device NI-488.2 calls</u> that issue GPIB command bytes (SendCmds, PPoll, Send, and Receive)

Use **ibsic** or **SendIFC** to make the GPIB interface become CIC on the GPIB.

Use the IbcSC option in <u>ibconfig</u> to make sure your GPIB interface is configured as System Controller.

In multiple CIC situations, always make sure the CIC bit appears in the status word <u>Ibsta</u> before attempting these calls. If it does not appear, you can perform an <u>ibwait</u> (for CIC) call to delay further processing until control is passed to the board.

EDMA Error Code

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

You can correct the EDMA problem in the software by using *ibconfig* with the IbcDMA option to disable DMA.

EDVR Error Code

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

EDVR is also returned if there is an internal driver error.

Use *ibdev* to open a device without specifying its symbolic name.

Use only device or board names that are configured in the GPIB Configuration utility or <u>Measurement & Automation Explorer</u> as parameters to the <u>ibfind</u> function.

To ensure that each board you want to access is working properly, use the NI-488.2 Troubleshooting Utility. This utility verifies that your GPIB hardware and the NI-488.2 software are installed correctly and working properly.

To start the NI-488.2 Troubleshooting Utility within <u>Measurement &</u> <u>Automation Explorer</u>, select Help»Troubleshooting»NI-488.2 Troubleshooting Utility or click on the following button.

Start the NI-488.2 Troubleshooting

Utility.

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.

For more troubleshooting information, refer to the <u>EDVR Error Conditions</u> topic.

EFSO Error code

EFSO results when an <u>ibrdf</u> or <u>ibwrtf</u> 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.

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.

EHDL Error code

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

- A valid board handle is passed in as a handle parameter to a device-level NI-488 function or a valid device handle is passed in as a handle parameter to a board-level NI-488 function.
- An invalid board or device unit descriptor is passed as input to any NI-488 function.
- A boardID outside the range of 099 is passed in to a traditional NI-488 board-level function or NI-488.2 routine.
- <u>ibconfig</u> or <u>ibask</u> is called with a device unit descriptor and a board-only configuration option, or with a board unit descriptor and a device-only configuration option.

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

ELCK Error code

ELCK indicates that the requested operation could not be performed because of an existing lock by another process accessing the same interface. ELCK is also returned when a process attempts to unlock an interface for which it currently has no lock.

Call <u>iblck</u> to lock the interface. If iblck continues to return ELCK, lengthen the LockWaitTime and wait for the other process to relinquish its interface lock.

Ensure that you have successfully locked the interface prior to unlocking it.

ENEB Error code

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

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 Utility.

To start the NI-488.2 Troubleshooting Utility within <u>Measurement &</u> <u>Automation Explorer</u>, select Help»Troubleshooting»NI-488.2 Troubleshooting Utility or click on the following button.

Start the NI-488.2 Troubleshooting

Utility.

ENOL Error code

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.

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 *ibconfig* with the IbcPAD (or IbcSAD, if necessary) options to match the configured address to the device switch settings.

EOIP Error code

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.

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

- <u>ibnotify</u>If the <u>lbsta</u> value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- <u>ibwait</u>If the returned Ibsta contains CMPL, the driver and application are resynchronized.
- <u>ibstop</u>The I/O is canceled; the driver and application are resynchronized.
- **ibon** The I/O is canceled and the interface is reset; the driver and application are resynchronized.

EPWR Error code

EPWR results when an interface loses power. This often results when the system goes to and returns from a standby state.

Take all handles offline and reinitialize the application.

Quit the application and restart.

Disable standby and hibernate modes on the PC.

ERST Error code

ERST results when an event notification was cancelled due to a reset of the interface.

An *ibwait* call pending in the driver returns ERST in the following situations:

- Another thread in the same process calls *ibonl* using the same unit descriptor as ibwait.
- Another thread or another process issues a board-level ibonl 1.

An **ibnotify** Callback may be invoked with ERST in the following situations:

• Another process issues a board-level ibonl 1.

Do not call ibonl with ibwait calls still pending in the driver.

Prevent other applications from calling ibonl by locking the interface with <u>iblck</u>.

_

ESAC Error code

ESAC results when <u>ibsic</u>, <u>ibsre</u>, <u>SendIFC</u>, <u>EnableRemote</u>, or the IbcSRE option in <u>ibconfig</u> is called when the GPIB interface does not have System Controller capability.

Give the GPIB interface System Controller capability by calling the IbcSC option in <u>ibconfig</u> or by using <u>Measurement & Automation Explorer</u> to configure that capability into the software.

ESRQ Error code

ESRQ can be returned by a device-level <u>ibwait</u> 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 <u>Ibsta</u> RQS bit while the condition lasts.

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

ETAB Error code

ETAB occurs only during the FindLstn and FindRQS calls. ETAB indicates that there was some problem with a table used by these calls, as follows:

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.

In the case of <u>FindLstn</u>, 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.

EWIP Error code

EWIP indicates that an <u>ibwait</u> call is already in progress on the specified unit descriptor. This error occurs when one thread within a process calls ibwait on a given descriptor when another thread within the same process is already performing an ibwait using that same descriptor.

Make sure that for any given unit descriptor only one thread calls ibwait at a time using that descriptor.

_

Configuration Errors

Some applications require customized configuration of the NI-488.2 driver. For example, you might want to <u>terminate reads</u> on a special end-of-string character, or you might require <u>secondary addressing</u>. In these cases, you can either reconfigure from your application using the ibconfig function or reconfigure using <u>Measurement & Automation Explorer</u>.



Note Configuring the NI-488.2 driver from within your application facilitates application portability when used with additional computers or hardware.

Communication Errors

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 <u>GPIB addressing</u> 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 <u>Measurement & Automation Explorer</u>.



Note National Instruments recommends using ibconfig to modify the configuration.

Termination Method

You should be aware of the <u>data termination method</u> 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\x0D\x0A",9);

Refer to your instrument's documentation to determine which, if any, termination characters to use.

Timing Errors

If your application fails, but the same calls issued interactively in the <u>Interactive Control utility</u> 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 is only a problem with older, non-standard GPIB devices.

To start Interactive Control within <u>Measurement & Automation Explorer</u>, select **Tools»NI-488.2»Interactive Control** or click on the following button.

Start 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 will be 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 may not eliminate timing errors.

Other Errors

If you experience other errors in your application, refer to the <u>Troubleshooting Problems</u> topic.

How to

Add a New GPIB Interface Remove a GPIB Interface Scan for GPIB Instruments View GPIB Instrument Information Communicate with a GPIB Instrument Set Network Settings for the GPIB-ENET/100 Use Static IP Address Settings Suggested for the GPIB-ENET/100 View NI-488.2 Software Version Enable/Disable NI-488.2 DOS Support Access Additional Help and Resources for GPIB Add a New GPIB Interface

Plug and Play Hardware

Windows automatically detects Plug and Play hardware. Install your GPIB hardware according to the instructions in the getting started documentation on the NI-488.2 for Windows CD. For more information about the documentation, refer to <u>Access Additional Help and Resources for GPIB</u>.

GPIB-ENET/100

To add a new GPIB-ENET/100 to your system, complete the following steps:

1. Select Start»Programs»National Instruments»NI-488.2»Add GPIB-ENET-100 Wizard.

Click here to run the Add

GPIB-ENET/100 Wizard.

- 2. The Add GPIB-ENET/100 Wizard appears. Follow the instructions in the wizard to add a new GPIB interface.
- 3. Open Measurement & Automation Explorer or select View»Refresh to update the list of GPIB interfaces in Measurement & Automation Explorer.
 - > Start Measurement & Automation Explorer.

Remove a GPIB Interface

To remove a Plug and Play interface from your computer, disconnect it, making sure to turn off the computer if the interface requires it.

To remove a GPIB-ENET/100 interface from your computer, you must manually delete it from within <u>Measurement & Automation Explorer</u>.

- 1. > Start Measurement & Automation Explorer.
- 2. Expand **Devices and Interfaces** by clicking the + next to the folder.
- 3. Right-click on your GPIB-ENET/100 interface and select **Delete** from the context menu.
- 4. When prompted, confirm your selection.
- 5. Select **View**»**Refresh** to update the list of interfaces in Measurement & Automation Explorer.

Scan for GPIB Instruments

To scan for instruments connected to your GPIB interface, complete the following steps:

- 1. Make sure that your instrument is powered on and connected to your GPIB interface.
- 2. > Start Measurement & Automation Explorer.
- 3. In Measurement & Automation Explorer, expand the **Devices and Interfaces** directory by clicking on the + next to the folder.
- 4. Right-click on your GPIB interface and select **Scan for Instruments** from the drop-down menu that appears. Measurement & Automation Explorer displays the connected instruments in the right window pane.

Instruments Not Found

If the **Instruments not Found** message appears in the right window pane, Measurement & Automation Explorer did not find any instruments. To solve this problem, make sure that your GPIB instruments are powered on and properly connected to the GPIB interface with a GPIB cable. Then, scan for instruments again, as described in the previous section.

Too Many Listeners

If the **Too Many Listeners** message appears in the right window pane, Measurement & Automation Explorer found too many Listeners on the GPIB. To solve this problem, refer to the following possible solutions:

If you are using an analyzer card on the GPIB, and it is set to participate in the acceptor handshake, your instruments will not be found. Stop capturing with the analyzer and retry the scan for instruments.

If you have a GPIB extender in your system, Measurement & Automation Explorer cannot detect any instruments connected to your GPIB interface. Instead, you can verify communication with your instruments using the Interactive Control utility.

To do so, select Tools»NI-488.2»Interactive Control or click here

to start the Interactive Control utility. For more information about verifying instrument communication, type help "Interactive Control:getting started" at the Interactive Control command prompt.

View GPIB Instrument Information

To view information about your GPIB interfaces, complete the following steps:

- 1. If you have not already done so, scan for connected instruments.
- 2. In Measurement & Automation Explorer, expand the **Devices and Interfaces** directory by clicking on the + next to the folder.
- Select your GPIB interface.
 Measurement & Automation Explorer displays the connected instruments in the right window pane.
- 4. Double-click on the instrument displayed in the right window pane.

Measurement & Automation Explorer lists all the attributes for the instrument, such as the primary address, the secondary address (if applicable), the instrument's response to the identification query (*IDN?), and the GPIB interface number to which the device is connected.

> Start Measurement & Automation Explorer.

Communicate with a GPIB Instrument

To establish <u>basic</u> or <u>advanced</u> communication with your instruments, refer to the following sections.

For more information about instrument communication and a list of the commands that your instrument understands, refer to the documentation that came with your GPIB instrument. Most instruments respond to the *IDN? command by returning an identification string.

Basic Communication (Query/Write/Read)

To establish basic communication with your instrument, use the NI-488.2 Communicator, as follows:

- 1. If you have not already done so, scan for connected instruments.
- 2. > Start Measurement & Automation Explorer.
- 3. In Measurement & Automation Explorer, expand the **Devices and Interfaces** directory by clicking on the + next to the folder.
- 4. Select your GPIB interface. Measurement & Automation Explorer displays the connected instruments in the right window pane.
- 5. Right-click on your GPIB instrument and select **Communicate** with Instrument from the drop-down menu that appears. The NI-488.2 Communicator dialog box appears.
- 6. Type a command in the **Send String** field and do one of the following:
 - To write a command to the instrument then read a response back, click on the **Query** button.
 - To write a command to the instrument, click on the **Write** button.
 - To read a response from the instrument, click on the **Read** button.
 - To configure special requirements for end of string (EOS) modes for your device, click on the Configure EOS button.

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

Advanced Communication

For advanced interactive communication with GPIB instruments, use the Interactive Control utility, as follows:

- 1. In Measurement & Automation Explorer, select **Tools»NI-**488.2»Interactive Control from the menu.
- 2. At the command prompt, type NI-488.2 API calls to communicate interactively with your instrument. For example, you might use ibdev, ibclr, ibwrt, ibrd, and ibonl.

To view the help for Interactive Control, type help at the Interactive Control command prompt.

Set Network Settings for the GPIB-ENET/100

To view or change the network settings of your GPIB-ENET/100, refer to the following sections. For more information about your GPIB-ENET/100 network settings, refer to the *GPIB Hardware Guide*, which was installed with your NI-488.2 software.

Click here to open the GPIB Hardware

Guide.

Configure Ethernet Settings

Use the NI Ethernet Device Configuration utility if you need to manually configure the network parameters of the GPIB-ENET/100. If your network uses DHCP, the network configuration is performed automatically at startup and you do not need to run this utility unless you want to change the hostname. Consult your network administrator if you do not know whether your network uses DHCP.

Click here to start the **NI Ethernet**

Device Configuration utility.

Refer to the *GPIB Hardware Guide* for information about using the utility to configure the network settings for your GPIB-ENET/100.

Update GPIB-ENET/100 Firmware

You can run the Firmware Update utility in Measurement & Automation Explorer, as follows:

1.

Start Measurement &

Automation Explorer.

- 2. In Measurement & Automation Explorer, expand the **Devices and Interfaces** directory by clicking on the + next to the folder.
- 3. Right-click on your GPIB-ENET/100 interface and select **Update Firmware** from the drop-down menu that appears.

For more information on how to use the Firmware Update utility, refer to the *GPIB Hardware Guide*.

Use Static IP Address Settings Suggested for the GPIB-ENET/100

The best way to assign your GPIB-ENET/100 a static IP address is to work with your network administrator, as network problems can occur if improper settings are selected. However, if you are using an unmanaged network or a crossover cable, the IP address settings suggested by the Add GPIB-ENET/100 Wizard may be useful.

When you use the **Suggest Values** button in the Add GPIB-ENET/100 Wizard to get suggested static IP address settings, your computer network settings are used as a basis for the suggestions. If the GPIB-ENET/100 will use the same subnet as your computer, you most likely can reuse the **Gateway**, **DNS Server**, and **Subnet Mask** settings. The **IP Address** field, however, must be changed. For a more detailed description of these fields, refer to <u>Static IP Parameters</u>.

IP Address Selection

The initially suggested static IP address is the subnet address of the network your computer is using and cannot be used for your GPIB-ENET/100. This address was generated by combining your computer's IP address and subnet mask values, and is separated into four numbers, or octets.

To generate a valid IP address inside the subnet range, use the suggested IP address and the subnet mask from the Add GPIB-ENET/100 Wizard. If an octet in the subnet mask is 255, you should not change the corresponding octet in the suggested IP address. If the subnet mask octet is 0, you can give the corresponding octet in the suggested IP address a value in the range 1254. For other subnet mask octet values, consult your network administrator or an IP address reference to determine a range of values that can be used. When you have determined a value for each octet in the IP address, make sure the entire address is unique on your network.

Multihomed Computers

Computers with more than one network interface are known as *multihomed*. The suggested IP address values are generated from the first active network interface discovered, and not necessarily the interface by which the GPIB-ENET/100 is connected to your computer. Be sure the settings are correct for the GPIB-ENET/100 network.

Assigning Static IP Addresses on a DHCP Network

Assigning a static IP address on a dynamically configured network is not recommended. If you are using a static IP address anyway, be sure to work with your network administrator to assign an address that the DHCP server will not also assign.

Static IP Parameters

To set a static IP address, you must provide the GPIB-ENET/100 with several important network parameters. These parameters are listed below.

- **IP address**The unique, computer-readable address of a device on your network. An IP address typically is represented as four decimal numbers separated by periods (for example, 130.164.54.215).
- **Subnet mask**A code that helps the network device determine whether another device is on the same network or a different network.
- **Gateway IP**The IP address of a device that acts as a gateway, which is a connection between two networks. If your network does not have a gateway, set this parameter to 0.0.0.0.
- **DNS server**The IP address of a network device that stores hostnames and translates them into IP addresses. If your network does not have a DNS server, set this parameter to 0.0.0.0.

The Add GPIB-ENET/100 Wizard can suggest values for these network parameters based on your computer configuration.

Click here to run the Add GPIB-

ENET/100 Wizard.

View NI-488.2 Software Version

To determine which version of the NI-488.2 software version you have installed, complete the following steps:

- 1. > Start Measurement & Automation Explorer.
- 2. Expand the **Software** directory by clicking on the + next to the folder.
- 3. Click on NI-488.2

Measurement & Automation Explorer displays the version number of the NI-488.2 software in the right window pane.

Enable/Disable NI-488.2 DOS Support

NI-488.2 DOS support allows GPIB programs compiled for MS-DOS to run on Windows 2000/XP.

Note DOS support is not available on Windows Vista or later.

To enable or disable NI-488.2 DOS support, complete the following steps:

1.

Start Measurement &

Automation Explorer.

- 2. In Measurement & Automation Explorer, select **Tools**»NI-488.2»DOS Support... from the **Tools** menu.
- 3. Enable or disable DOS support in the **NI-488.2 Settings** dialog box and click the **OK** button.
- 4. If you are prompted to do so, restart your system.

Require Administrator Privileges

NI-488.2 allows administrators to allow non-administrators to make changes to GPIB settings. This is controlled using the Measurement & Automation Explorer **Tools»NI-488.2»Require Administrator Privileges...** menu. This menu is enabled only if MAX has been run as administrator. In either case, a check next to the menu indicates that GPIB setting changes can be made only when MAX is run as administrator.

To toggle between different modes, complete the following steps:

- Run MAX as administrator.
- In MAX, select **Tools»NI-488.2»Require Administrator Privileges...** from the menu.
- Enable or disable the requirement in the NI-488.2 Settings dialog box and click the **OK** button.

Access Additional Help and Resources for GPIB

To access additional help and resources for the NI-488.2 software and your GPIB hardware, you can refer to the National Instruments GPIB Web site or the *NI-488.2 for Windows* CD.

National Instruments GPIB Web Site

To access the National Instruments Web site for GPIB in Measurement & Automation Explorer, select Help»National Instruments on the Web»GPIB Home Page.

NI-488.2 for Windows CD

The following NI-488.2 documentation is available on your *NI-488.2 for Windows* CD:

- The *GPIB Installation Guide* briefly describes how to install the NI-488.2 software and your GPIB hardware.
- The *NI-488.2 User Manual* describes the features and functions of the NI-488.2 software.
- The *GPIB Hardware Guide* contains detailed instructions on how to install and configure your GPIB hardware. This guide also includes hardware specifications and compliance information.
- The *GPIB Analyzer User Manual* contains instructions to help you use the GPIB analyzer software.
- The *NI-488.2 API Quick Reference Card* lists status word conditions, error codes, functions, board options, device options, multiline interface messages, routines, and timeout values for the NI-488.2 API.

To view these documents online, insert your NI-488.2 CD. When the NI-488.2 for Windows screen appears, select View Documentation. The documentation utility helps you find the documentation you want to view. You can also view these documents on our <u>Web site</u>.

Frequently Asked Questions

How do I get error information about failed NI-488.2 calls?
How do I communicate with my instrument?
How do I change a GPIB device template?
How can I NULL terminate an ASCII response from my instrument?
Are interrupts required for the NI-488.2 software?
Is DMA required for the NI-488.2 software?
Is my instrument 488.1 or 488.2 compliant?
How can I determine which type of GPIB hardware I have installed?
How do I use an NI-488.2 application interface?

How do I get error information about failed NI-488.2 calls?

Each NI-488.2 call updates three global functions to reflect the status of the device or board that you are using. These global status functions are the status word (Ibsta), the error function (Iberr), and the count function (Ibcnt). They contain useful information about the last NI-488.2 call that was executed for any given process. Your application should check these functions after each NI-488.2 call. For more information about each status function, refer to the following sections.

If you are writing a multithreaded application, use the thread-specific copies of the status functions in your application. To access the thread-specific copies, use the <u>ThreadIbsta</u>, <u>ThreadIberr</u>, and <u>ThreadIbcnt</u> calls.

Status Word (Ibsta)

Ibsta is the GPIB status word. It contains information about the state of the GPIB and your GPIB hardware. The value stored in Ibsta is the return value of all of the traditional NI-488.2 calls, except <u>ibfind</u> and <u>ibdev</u>. You can examine various status bits in Ibsta and use that information to determine what to do next in your application. For more information about the status bits in Ibsta, refer to the <u>Ibsta Status Bit Values</u> table.

To check for errors after each NI-488.2 call, use the ERR bit in Ibsta, as follows:

if (Ibsta() & ERR)

printf("GPIB error encountered");

Error Function (Iberr)

Iberr is the GPIB error variable. If a call failed with an error, the ERR bit is set in Ibsta. The Iberr value describes the GPIB error that occurred. For more information about the Iberr values, refer to the Error Codes and Solutions table.

Count Function (lbcnt)

Ibent is the count function. It contains information about the number of bytes that went across the GPIB in the most recent I/O operation.

The count function is updated after all I/O operations like <u>ibrd</u>, <u>ibwrt</u>, <u>SendList</u>, and <u>SendCmds</u>. If you are reading data, the count function indicates the number of bytes read. If you are sending data or commands, the count variables reflect the number of data or command bytes sent.

If the data that you are reading contains ASCII characters, you can use Ibcnt to NULL terminate the string and treat it like any other ASCII string. For example, you can use printf to print the result to the screen:

```
char rdbuf[21];
ibrd (ud, rdbuf, 20);
if (!(Ibsta() & ERR)){
  rdbuf[Ibcnt()] = '\0';
  printf ("Read in string: %s\n", rdbuf);
}
else {
  // GPIB Error encountered!
}
In addition, Ibcnt may return information from non-I/O functions, such as
EDVR error codes.
```

Return to Frequently Asked Questions

Iberr() or iberr Error Codes and Solutions

Iberr is the NI-488.2 error function. If a call failed within an error, the Iberr value describes the NI-488.2 error that occurred. When using the GPIB32 API, iberr is the global variable.

The following table lists the NI-488.2 error codes. Remember that the error function is meaningful only when the ERR bit in the status function, <u>Ibsta</u>, is set. For a detailed description of each error and possible solutions, click on the error mnemonic.

Mnemonic Value Meaning

,		
<u>EDVR</u>	0	System error
<u>ECIC</u>	1	Function requires GPIB interface to be CIC
<u>ENOL</u>	2	No Listeners on the GPIB
<u>EADR</u>	3	GPIB interface not addressed correctly
<u>EARG</u>	4	Invalid argument to function call
<u>ESAC</u>	5	GPIB interface not System Controller as required
<u>EABO</u>	6	I/O operation aborted (timeout)
<u>ENEB</u>	7	Nonexistent GPIB interface
<u>EDMA</u>	8	DMA error
EOIP	10	Asynchronous I/O in progress
<u>ECAP</u>	11	No capability for operation
<u>EFSO</u>	12	File system error
<u>EBUS</u>	14	GPIB bus error
<u>ESRQ</u>	16	SRQ stuck in ON position
<u>ETAB</u>	20	Table problem
<u>ELCK</u>	21	Interface is locked
<u>EARM</u>	22	ibnotify callback failed to rearm
<u>EHDL</u>	23	Input handle is invalid
<u>EWIP</u>	26	Wait in progress on specified input handle

- ERST 27 The event notification was cancelled due to a reset of the interface
- **EPWR** 28 The interface lost power

Ibsta() or ibsta Status Bit Values

All calls update a global status function, Ibsta, which contains information about the state of the GPIB and your GPIB hardware. 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. When using the GPIB32 API, ibsta is the global variable.

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). Ibsta is a 32-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.

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

How do I communicate with my instrument?

Refer to the documentation provided by your GPIB instrument manufacturer. The command sequences for GPIB instrument communication are specific to each GPIB instrument. The documentation for your GPIB instrument should include the GPIB commands that you need to use to communicate with your instrument.

In most cases, device-level traditional NI-488.2 calls are sufficient for instrument communication. Typically, you will use <u>ibdev</u> to open a handle to your GPIB instrument. Once you have a handle, you can use <u>ibwrt</u> to send commands to your instrument and <u>ibrd</u> to get data from your instrument. When you are finished, use <u>ibonl</u> to close the handle.

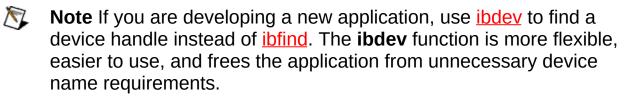
For simple instrument communication, use the NI-488.2 Communicator. For more information, refer to the Using NI-488.2 Communicator topic.

To view an example of the Interactive Control utility communicating with a GPIB instrument, refer to the <u>Getting Started with Interactive Control</u> topic.

For instructions on how to develop your application, refer to the <u>Developing Your NI-488.2 Application</u> topic.

How do I change a GPIB device template?

Some NI-488.2 applications require you to configure a device template before you can successfully run the application. Refer to the documentation provided with your NI-488.2 application to determine whether you need to change a device template.



If you must change a device template, complete the following steps:

1. Start the GPIB Configuration utility.

- 2. Double-click the device template that you want to modify, such as **DEV1**.
- 3. Modify the device template as described in your application documentation.
- 4. Click on the **OK** button twice to save your changes and exit.

If you are using NI-488.2 DOS support to run a GPIB program compiled for MS-DOS, you must also run the updatdos.exe program to update the device templates for the DOS driver. Updatdos.exe is in the DOSWIN16 directory of your NI-488.2 installation.

Note DOS support is not available on Windows Vista or later.

How can I NULL terminate an ASCII response from my instrument?

If your instrument sends you ASCII data, you can convert the string that is returned from a read operation (like <u>ibrd</u> or <u>Receive</u>) to a NULL-terminated string that can be passed to string calls like printf. After a successful read operation, Ibcnt contains the number of bytes read. To create a NULL-terminated string, add a NULL byte onto the end of the string as shown in the following C/C++ code:

char ReadBuffer[101];

```
ibrd (handle, ReadBuffer, 100);
```

if (!(Ibsta() & ERR)) {

```
ReadBuffer[Ibcnt()] = '\0';
```

```
printf ("Read string from instrument: %s",
```

ReadBuffer);

}

Are interrupts required for the NI-488.2 software?

Generally, plug-in interface cards, such as the PCI-GPIB, require interrupt resources in your computer. Remote interfaces, such as the GPIB-USB and GPIB-ENET products, do not require interrupt resources in your computer. There may be exceptions to this statement. Refer to the general readme file, Readme.txt, located on your installation CD or in the installation directory, for the latest interface options supported by the current version of NI-488.2.

Is DMA required for the NI-488.2 software?

No, DMA is not required.

Is my instrument 488.1 or 488.2 compliant?

Refer to the documentation that came with your GPIB instrument to determine whether your instrument is IEEE 488.2 compliant. The NI-488.2 software works with both 488.1 and 488.2 GPIB instruments.

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

Select Start»Programs»National Instruments»Measurement & Automation or click on the following button.

> Start Measurement & Automation Explorer.

Expand the **Devices and Interfaces** directory by clicking on the + next to the folder. Measurement & Automation Explorer lists your installed GPIB hardware under **Devices and Interfaces**.

Function Reference

<u>Traditional NI-488.2 Calls</u> <u>Multi-Device NI-488.2 Calls</u> <u>Supplemental Calls for Multithreaded NI-488.2 Applications</u>

Traditional NI-488.2 Calls

List of Board-Level NI-488.2 Calls List of Device-Level NI-488.2 Calls

Board-Level Traditional NI-488.2 Calls

<u>ibask</u>	Return information about software configuration parameters
<u>ibcac</u>	Become Active Controller
<u>ibcmd</u>	Send GPIB commands
<u>ibcmda</u>	Send GPIB commands asynchronously
ibconfig	Change the software configuration parameters
<u>ibdma</u>	Enable or disable DMA (see the IbcDMA option in <i>ibconfig</i>)
<u>ibeos</u>	Configure the end-of-string (EOS) termination mode or character (see the lbcEOS option in <u>ibconfig</u>)
<u>ibeot</u>	Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations (see the IbcEOT option in ibconfig)
<u>ibfind</u>	Open and initialize an interface or a user-configured instrument descriptor
<u>ibgts</u>	Go from Active Controller to Standby
<u>ibist</u>	Set or clear the board individual status bit for parallel polls (see the lbclst option in <u>ibconfig</u>)
<u>iblck</u>	Acquire or release an exclusive interface lock for the current process
<u>iblines</u>	Return the status of the eight GPIB control lines
<u>ibln</u>	Check for the presence of a device on the bus
<u>ibloc</u>	Go to Local
<u>ibnotify</u>	Notify user of one or more GPIB events by invoking the user callback
<u>ibonl</u>	Place the interface online or offline
<u>ibpad</u>	Change the primary address (see the IbcPAD option in ibconfig)
<u>ibppc</u>	Parallel poll configure
<u>ibrd</u>	Read data from an instrument into a user buffer
<u>ibrda</u>	Read data asynchronously from an instrument into a user buffer

<u>ibrdf</u>	Read data from an instrument into a file
<u>ibrpp</u>	Conduct a parallel poll
<u>ibrsc</u>	Request or release system control (see the lbcSC option in ibconfig)
<u>ibrsv</u>	Request service and change the serial poll status byte (see the lbcRSV option in <u>ibconfig</u>)
<u>ibsad</u>	Change or disable the secondary address (see the lbcSAD option in <u>ibconfig</u>)
<u>ibsic</u>	Assert interface clear
<u>ibsre</u>	Set or clear the Remote Enable (REN) line (see the lbcSRE option in <u>ibconfig</u>)
<u>ibstop</u>	Abort asynchronous I/O operation
<u>ibtmo</u>	Change or disable the I/O timeout period (see the IbcTMO option in ibconfig)
<u>ibwait</u>	Wait for GPIB events
<u>ibwrt</u>	Write data to an instrument from a user buffer
<u>ibwrta</u>	Write data asynchronously to an instrument from a user buffer
<u>ibwrtf</u>	Write data to an instrument from a file

Device-Level Traditional NI-488.2 Calls

<u>ibask</u>	Return information about software configuration parameters
<u>ibclr</u>	Clear a specific instrument
<u>ibconfig</u>	Change the software configuration parameters
<u>ibdev</u>	Open and initialize an instrument
<u>ibeos</u>	Configure the end-of-string (EOS) termination mode or character (see the lbcEOS option in <u>ibconfig</u>)
<u>ibeot</u>	Enable or disable the automatic assertion of GPIB EOI line at the end of write I/O operations (see the IbcEOT option in ibconfig)
<u>ibfind</u>	Open and initialize a board or a user-configured device descriptor.
<u>ibloc</u>	Go to Local
<u>ibnotify</u>	Notify user of one or more GPIB events by invoking the user callback
<u>ibonl</u>	Place the instrument online or offline
<u>ibpad</u>	Change the primary address (see the lbcPAD option in ibconfig)
<u>ibpct</u>	Pass control to another GPIB instrument with Controller capability
<u>ibppc</u>	Parallel poll configure
<u>ibrd</u>	Read data from an instrument into a user buffer
<u>ibrda</u>	Read data asynchronously from an instrument into a user buffer
<u>ibrdf</u>	Read data from an instrument into a file
<u>ibrpp</u>	Conduct a parallel poll
<u>ibrsp</u>	Conduct a serial poll
<u>ibsad</u>	Change or disable the secondary address (see the lbcSAD option in <u>ibconfig</u>)
<u>ibstop</u>	Abort asynchronous I/O operation
<u>ibtmo</u>	Change or disable the I/O timeout period (see the IbcTMO

option in *ibconfig*)

- ibtrg Trigger selected instrument
- ibwait Wait for GPIB events
- ibwrt Write data to an instrument from a user buffer
- ibwrta Write data asynchronously to an instrument from a user buffer
- ibwrtf Write data to an instrument from a file

IBASK Board-Level/Device-Level

Purpose

Return information about software configuration parameters.

Format

С

unsigned long ibask (int ud, int option, int *value)

Visual Basic

CALL ibask (ud%, option%, value%)

or

status% = ilask (ud%, option%, value%)

Interactive Control (Usage Notes)

ibask option

Input

ud Board or device unit descriptor

option Selects the configuration item whose value is being requested

Output

valueCurrent value of the selected configuration itemFunction ReturnThe value of Ibsta.

Description

ibask returns the current value of various configuration parameters for the specified board or device. The current value of the selected configuration item is returned in the integer value. Refer to the table with valid configuration parameter options for ibask.

Possible Errors

- EARG option is not a valid configuration parameter. See <u>ibask Board</u> <u>Configuration Parameter Options</u> and <u>ibask Device Configuration</u> <u>Parameter Options</u>.
- **ECAP** option is not supported by the driver or the interface is not configured correctly.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

ibask Board Configuration Parameter Options

You can use the following options with *ibask* when ud is a board descriptor or a board index.

IbaAUTOPOLL	IbaLON
<u>IbaDMA</u>	IbaPAD
<u>IbaEndBitIsNormal</u>	IbaPP2
<u>IbaEOS</u>	IbaPPC
<u>IbaEOSchar</u>	IbaPPollTime
IbaEOScmp	IbaRsv
IbaEOSrd	IbaSAD
<u>IbaEOSwrt</u>	<u>IbaSC</u>
<u>IbaEOT</u>	IbaSendLLO
IbaHSCableLength	IbaSRE
<u>Ibalst</u>	IbaTIMING
IbalRQ	<u>IbaTMO</u>

Seturn to ibask

IbaAUTOPOLL (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaAUTOPOLL	zero non-zero	Automatic serial polling is disabled. Automatic serial polling is enabled.

IbaDMA (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaDMA	zero	The board will not use DMA for GPIB transfers.
	non-zero	The board will use DMA for GPIB transfers.

IbaEndBitIsNormal (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaEndBitIsNormal	zero	The END bit of Ibsta is set only when EOI or EOI plus the EOS character is received. If the EOS character is received without EOI, the END bit is not set.
	non- zero	The END bit is set whenever EOI, EOS, or EOI plus EOS is received.

IbaEOS (ibask Board Configuration Option)

Using this option makes your application break backwards compatibility with older drivers. Consider using IbaEOSchar, IbaEOScmp, IbaEOSrd, and IbaEOSwrt separately instead.

IbaEOS is the current EOS termination mode and character for the board. The lower byte of the value returned is the EOS character. For the upper byte, the third bit flags, terminating the read when EOS is detected, the fourth bit flags, setting EOI with EOS on write functions, and the fifth bit flags, comparing all 8 bits of the EOS byte instead of comparing the lower 7 bits. The following table describes the different EOS configurations:

		Value	e of v
Bit	Configuration	High Byte	Low Byte
A	Terminate read when EOS is detected.	00000100	EOS character
В	Set EOI with EOS on write function.	00001000	EOS character
C	Compare all 8 bits of EOS byte rather than low 7 bits (all read and write calls).	00010000	EOS character

IbaEOSchar (ibask Board Configuration Option)

Options (Constants)	Description
IbaEOSchar	The current EOS character of the board.

IbaEOScmp (ibask Board Configuration Option)

Options (Constants) Value		Description
IbaEOScmp	zero non- zero	A 7-bit compare is used for all EOS comparisons. An 8-bit compare is used for all EOS comparisons.

IbaEOSrd (ibask Board Configuration Option)

Options (Constants) Value		Description	
IbaEOSrd	zero	The EOS character is ignored during read operations.	
	non- zero	Read operations are terminated by the EOS character.	

IbaEOSwrt (ibask Board Configuration Option)

Options (Constants) Value Description

IbaEOSwrt	zero	The EOI line is not asserted when the EOS character is sent during a write operation.
		The EOI line is asserted when the EOS character is sent during a write operation.

-

IbaEOT (ibask Board Configuration Option)

Options (Constants) Value Description

IbaEOT	zero	The GPIB EOI line is not asserted at the end of write operations.
	non- zero	EOI is asserted at end of writes.

IbaHSCableLength (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaHSCableLength	0 1 to 15	High-speed (HS488) data transfer is disabled. High-speed (HS488) data transfer is enabled. The number returned represents the number of meters of GPIB cable in your system.

IbalRQ (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaIRQ	zero non-zero	Do not use interrupts. Use interrupts.

Ibalst (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaIst	zero non-zero	The board's ist bit is cleared. The board's ist bit is set.

IbaLON (ibask Board Configuration Option)

Options (Constants) Value Description

IbaLON	zero	(Default) Do not enable listen-only mode.
	non- zero	Enable listen-only mode. In this mode, which persists until ibonl is called, the board is forced to listen. While in listen-only mode, the board-level functions ibrd, ibrda, ibrdf, ibwait, and ibtmo function in the usual manner. The behavior of all other functions is undefined.

IbaPAD (ibask Board Configuration Option)

Options (Constants)	Description
IbaPAD	The current primary address of the board.

IbaPP2 (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaPP2	zero	The board is in PP1 moderemote <u>parallel poll</u> configuration.
	non- zero	The board is in PP2 modelocal parallel poll configuration.

IbaPPC (ibask Board Configuration Option)

Options (Constants) Description

IbaPPC The current parallel poll configuration information of the board.

IbaPPollTime (ibask Board Configuration Option)

Options (Constants) Value Description

IbaPPollTime	0	The board uses the standard duration (\geqq 2 μ s) when conducting a parallel poll.
	1 to 17	The board uses a variable length duration when conducting a parallel poll. The duration values correspond to the <i>ibtmo</i> timing values.

IbaRsv (ibask Board Configuration Option)

Options (Constants) Description

IbaRsv The current serial poll status byte of the board.

IbaSAD (ibask Board Configuration Option)

Options	
(Constants)	Description

IbaSAD The current secondary address of the board.

IbaSC (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaSC	zero non-zero	The board is not the System Controller. The board is the System Controller.

IbaSendLLO (ibask Board Configuration Option)

Options (Constants) Value Description

IbaSendLLO	zero	The GPIB LLO command is not sent when a device is put online <u>ibfind</u> or <u>ibdev</u> .
	non- zero	The LLO command is sent.

-

IbaSRE (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaSRE	zero non-zero	The board is not asserting the REN line. The board is asserting the REN line.

IbaTIMING (ibask Board Configuration Option)

Options (Constants)	Value	Description
IbaTIMING		The current bus timing of the board.
	1	Normal timing (T1 delay of 2 μ s).
	2	High-speed timing (T1 delay of 500 ns).
	3	Very high-speed timing (T1 delay of 350 ns).

IbaTMO (ibask Board Configuration Option)

-

Options (Constants) Description

IbaTMO The current I/O timeout period of the board.

ibask Device Configuration Parameter Options

You can use the following options with *ibask* when ud is a device descriptor.

<u>IbaEOS</u>	<u>IbaPAD</u>
<u>IbaEOSchar</u>	<u>IbaREADDR</u>
IbaEOScmp	<u>IbaSAD</u>
IbaEOSrd	<u>IbaSPollTime</u>
<u>IbaEOSwrt</u>	<u>IbaTMO</u>
<u>IbaEOT</u>	<u>IbaUnAddr</u>

Return to ibask

IbaEOS (ibask Device Configuration Option)

Using this option makes your application break backwards compatibility with older drivers. Consider using IbaEOSchar, IbaEOScmp, IbaEOSrd, and IbaEOSwrt separately instead.

IbaEOS is the current EOS termination mode and character for the device. The lower byte of the value returned is the EOS character. For the upper byte, the third bit flags, terminating the read when EOS is detected, the fourth bit flags, setting EOI with EOS on write functions, and the fifth bit flags, comparing all 8 bits of the EOS byte instead of comparing the lower 7 bits. The following table describes the different EOS configurations:

		Value	e of v
Bit	Configuration	High Byte	Low Byte
A	Terminate read when EOS is detected.	00000100	EOS character
В	Set EOI with EOS on write function.	00001000	EOS character
С	Compare all 8 bits of EOS byte rather than low 7 bits (all read and write calls).	00010000	EOS character

IbaEOSchar (ibask Device Configuration Option)

Options (Constants)	Description	
IbaEOSchar	The current EOS character of the device.	

IbaEOScmp (ibask Device Configuration Option)

Options (Constants) Value		Description
IbaEOScmp	zero non- zero	A 7-bit compare is used for all EOS comparisons. An 8-bit compare is used for all EOS comparisons.

IbaEOSrd (ibask Device Configuration Option)

Options (Constants) Value		Description
IbaEOSrd	zero	The EOS character is ignored during read operations.
	non- zero	Read operations are terminated by the EOS character.

IbaEOSwrt (ibask Device Configuration Option)

Options (Constants) Value Description

IbaEOSwrt	zero	The EOI line is not asserted when the EOS character is sent during a write operation.
	-	The EOI line is asserted when the EOS character is sent during a write operation.

-

IbaEOT (ibask Device Configuration Option)

Options (Constants) Value Description

IbaEOT	zero	The GPIB EOI line will not be asserted at the end of a write operation.
	non- zero	EOI will be asserted at the end of a write.

IbaPAD (ibask Device Configuration Option)

Options (Constants)	Description	
•	Description	

IbaPAD The current primary address of the device.

IbaREADDR (ibask Device Configuration Option)

Options (Constants) Value Description

IbaREADDR zero	No unnecessary addressing is performed between device-level read and write operations.
non-	Addressing is always performed before a device-

zero level read or write operation.

IbaSAD (ibask Device Configuration Option)

-

Options (Constants) Description

IbaSAD The current secondary address of the device.

IbaSPollTime (ibask Device Configuration Option)

Options (Constants) Description

IbaSPollTime The length of time the driver waits for a serial poll response when polling the device. The length of time is represented by the <u>ibtmo</u> timing values.

IbaTMO (ibask Device Configuration Option)

-

Options (Constants) Description

IbaTMO The current I/O timeout period of the device.

IbaUnAddr (ibask Device Configuration Option)

Options (Constants) Value Description

IbaUnAddr	zero	The GPIB commands Untalk (UNT) and Unlisten (UNL) are not sent after each device-level read and write operation.
		The UNT and UNL commands are sent after each device-level read and write.

-

Purpose

Become Active Controller.

Format

С

unsigned long ibcac (int ud, int v)

Visual Basic

CALL ibcac (ud%, v%)

or

status% = ilcac (ud%, v%)

Interactive Control (Usage Notes)

ibcac v

Input

- ud A board unit descriptor
- v Determines if control is to be taken asynchronously or synchronously

Output

Function Return

The value of **Ibsta**

Description

Using ibcac, the designated GPIB interface attempts to become the Active Controller by asserting ATN. If v is zero, the GPIB interface takes control asynchronously; if v is non-zero, the GPIB interface takes control synchronously. Before you call ibcac, the GPIB interface must already be CIC. To make the board CIC, use the <u>ibsic</u> function.

To take control synchronously, the GPIB interface attempts to assert the ATN signal without corrupting transferred data. If this is not possible, the board takes control asynchronously.

To take control asynchronously, the GPIB interface asserts ATN immediately without regard for any data transfer currently in progress.

Most applications do not need to use ibcac. Calls that require ATN to be asserted, such as <u>ibcmd</u>, do so automatically.

Possible Errors

- **EARG** ud is valid but does not refer to an interface.
- **<u>ECIC</u>** The interface is not Controller-In-Charge.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBCLR Device-Level

Purpose

Clear a specific device.

Format

С

unsigned long ibclr (int ud)

Visual Basic

CALL ibclr (ud%)

or

status% = ilclr (ud%)

Interactive Control (Usage Notes)

ibclr

Input

ud A device unit descriptor

Output

Function Return

The value of **Ibsta**

Description

ibclr sends the GPIB Selected Device Clear (SDC) message to the device described by ud.

Possible Errors

- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The access board is not CIC. Refer to Device-Level Calls and Bus Management.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBCMD Board-Level

Purpose

Send GPIB commands.

Format

С

unsigned long ibcmd (int ud, const void *cmdbuf, size_t count)

Visual Basic

CALL ibcmd (ud%, cmdbuf\$)

or

status% = ilcmd (ud%, cmdbuf\$, count&)

Interactive Control (Usage Notes)

ibcmd cmdbuf

Input

udA board unit descriptorcmdbufBuffer of command bytes to sendcountNumber of command bytes to send

Output

Function Return

The value of **Ibsta**

Description

ibcmd sends count bytes from cmdbuf over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global function, Ibcnt. Refer to the list of <u>IEEE 488</u> <u>command messages</u> for defined interface messages.

Command bytes are used to configure the state of the GPIB. They are not used to send instructions to GPIB devices. Use <u>ibwrt</u> to send devicespecific instructions.

Possible Errors

- EABO The timeout period expired before all of the command bytes were sent.
- ECIC The interface is not Controller-In-Charge.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners are on the GPIB.
- **EOIP** Asynchronous I/O is in progress.

IBCMDA Board-Level

Purpose

Send GPIB commands asynchronously.

Format

С

unsigned long ibcmda (int ud, const void *cmdbuf, size_t count)

Visual Basic

CALL ibcmda (ud%, cmdbuf\$)

or

status% = ilcmda (ud%, cmdbuf\$, count&)

Interactive Control (Usage Notes)

ibcmda cmdbuf

Input

udA board unit descriptorcmdbufBuffer of command bytes to sendcountNumber of command bytes to send

Output

Function Return

The value of **Ibsta**

Description

ibcmda sends count bytes from cmdbuf over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global function, Ibcnt. Refer to the list of <u>IEEE 488</u> <u>command messages</u> for defined interface messages.

Command bytes are used to configure the state of the GPIB. They are not used to send instructions to GPIB devices. Use <u>ibwrt</u> to send devicespecific instructions.

The asynchronous I/O calls (ibcmda, ibrda, ibwrta) are designed so that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O begins, further NI-488.2 calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed; the driver returns EOIP in this case.

Once the I/O is complete, the application must resynchronize with the NI-488.2 driver. Resynchronization is accomplished by using one of the following calls:

- ibwait If the returned Ibsta contains CMPL, the driver and application are resynchronized.
- ibnotify If the Ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- ibstop 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.

Possible Errors

- **<u>ECIC</u>** The interface is not Controller-In-Charge.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners are on the GPIB.
- **EOIP** Asynchronous I/O is in progress.

IBCONFIG

-

Board-Level/Device-Level

Purpose

Change the software configuration input.

Format

С

unsigned long ibconfig (int ud, int option, int value)

Visual Basic

CALL ibconfig (ud%, option%, value%)

or

status% = ilconfig (ud%, option%, value%)

Interactive Control (Usage Notes)

ibconfig option value

Input

- ud Board or device unit descriptor
- option A parameter that selects the software configuration item
- value The value to which the selected configuration item is to be changed

Output

Function Return

The value of **Ibsta**

Description

ibconfig changes a configuration item to the specified value for the selected board or device. option can be any of the defined options (see ibconfig Board Configuration Parameter Options or ibconfig Device Configuration Parameter Options). value must be valid for the parameter that you are configuring. The previous setting of the configured item is returned in Iberr.

Possible Errors

- EARG Either option or value is not valid. The <u>ibconfig Board</u> <u>Configuration Parameter Options</u> table lists the valid options.
- **ECAP** The driver is not able to make the requested change.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **ESAC** The board does not have System Controller capability.

ibconfig Board Configuration Parameter Options

You can use the following options with *ibconfig* when ud is a board descriptor or a board index.

IbcAUTOPOLL	
	<u>IbcLON</u>
IbcDMA	<u>IbcPAD</u>
<u>IbcEndBitIsNormal</u>	IbcPP2
IbcEOS	<u>IbcPPC</u>
IbcEOSchar	<u>IbcPPollTime</u>
IbcEOScmp	<u>IbcRsv</u>
IbcEOSrd	<u>IbcSAD</u>
<u>IbcEOSwrt</u>	<u>IbcSC</u>
<u>IbcEOT</u>	IbcSendLLO
IbcHSCableLength	<u>IbcSRE</u>
IbcIRQ	IbcTIMING
Ibclst	<u>IbcTMO</u>

Return to ibconfig

IbcAUTOPOLL (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcAUTOPOLL	non-zero	Disable <u>automatic serial polling</u> . Enable automatic serial polling. rmined by <u>Measurement & Automation</u>

IbcDMA (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcDMAzeroThe board will not use DMA for GPIB transfers.non-zeroThe board will use DMA for GPIB transfers.

This option enables or disables DMA transfers for the board, according to v. If v is zero, DMA is not used for GPIB I/O transfers. If v is non-zero, DMA is used for GPIB I/O transfers.

IbcEndBitIsNormal (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcEndBitIsNormal	zero	The END bit of Ibsta is set only when EOI is received. If the EOS character is received with EOI, the END bit is not set.
	non- zero	(Default) The END bit is set whenever EOI or EOS is received.

IbcEOS (ibconfig Board Configuration Option)

Using this option makes your application break backwards compatibility with older drivers. Consider using IbcEOSchar, IbcEOScmp, IbcEOSrd, and IbcEOSwrt separately instead.

IbcEOS is the parameter that describes the new EOS configuration to use. If v is zero, the EOS configuration is disabled. Otherwise, the low byte is the EOS character and the upper byte contains flags that define the EOS mode. For the upper byte, the third bit flags, terminating the read when EOS is detected, the fourth bit flags, setting EOI with EOS on write functions, and the fifth bit flags, comparing all 8 bits of the EOS byte instead of comparing the lower 7 bits. The following table describes the different EOS configurations:

		Value	e of v
Bit	Configuration	High Byte	Low Byte
A	Terminate read when EOS is detected.	00000100	EOS character
В	Set EOI with EOS on write function.	00001000	EOS character
С	Compare all 8 bits of EOS byte rather than low 7 bits (all read and write calls).	00010000	EOS character

Configuration bits A and C determine how to terminate read I/O operations. If bit A is set and bit C is clear, a read ends when a byte that matches the low seven bits of the EOS character is received. If bits A and C are both set, a read ends when a byte that matches all eight bits of the EOS character is received.

Configuration bits B and C determine when a write I/O operation asserts the GPIB EOI line. If bit B is set and bit C is clear, EOI is asserted when the written character matches the low seven bits of the EOS character. If bits B and C are both set, EOI is asserted when the written character matches all eight bits of the EOS character.

Refer to <u>Terminating Data Transfers</u> for more information about EOS and EOI termination methods.

Examples

ibconfig (ud, IbcEOS 0x140A); /* Configure the software to end reads on newline character (hex 0A) for the unit descriptor, ud */ ibconfig (ud, IbcEOS 0x180A); /* Configure the software to assert the GPIB EOI line whenever the newline character (hex 0A) is written out by the unit descriptor, ud */

IbcEOSchar (ibconfig Board Configuration Option)

Options (Constants) Description

IbcEOSchar	Any 8-bit value. This byte becomes the new EOS character.
	Default determined by <u>Measurement & Automation</u> <u>Explorer</u> .

IbcEOScmp (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcEOScmp		Use 7 bits for the EOS character comparison. Use 8 bits for the EOS character comparison. etermined by <u>Measurement & Automation</u>

IbcEOSrd (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcEOSrd	zero non- zero Default o <u>Explorer</u>	Ignore EOS character during read operations. Terminate reads when the EOS character is read. determined by <u>Measurement & Automation</u>

IbcEOSwrt (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcEOSwrt	zero	Do not assert the EOI line when the EOS character is sent during write operations.
	-	Assert the EOI line when the EOS character is sent during write operations.
Default determi <mark>Explorer</mark> .		It determined by <u>Measurement & Automation</u> rer.

IbcEOT (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcEOT	zero non- zero	The GPIB EOI line is not asserted at the end of write operations. EOI is asserted at end of writes.
	2010	Default determined by <u>Measurement & Automation</u> Explorer.

This option enables or disables the assertion of the EOI line at the end of write I/O operations for the board or device described by ud. If v is non-zero, EOI is asserted when the last byte of a GPIB write is sent. If v is zero, nothing occurs when the last byte is sent. If no error occurs during the call, the previous value of EOT is returned in Iberr.

Refer to <u>Terminating Data Transfers</u> for more information about EOS and EOI termination methods.

IbcHSCableLength (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcHSCableLength	0	High-speed (HS488) data transfer is disabled.
	1 to 15	The number of meters of GPIB cable in your system. The NI-488.2 software uses this information to select the appropriate high- speed (HS488) data transfer mode.
	Defau <mark>Explor</mark>	It determined by <u>Measurement & Automation</u> rer.

IbcIRQ (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcIRQ	zero non-zero	Do not use interrupts. Use interrupts.

Ibclst (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcIst	zero non-zero	The board's ist bit is cleared. The board's ist bit is set.

This option sets the interface ist (individual status) bit according to v. If v is zero, the ist bit is cleared; if v is non-zero, the ist bit is set. The previous value of the ist bit is returned in Iberr.

For more information about parallel polling, refer to the <u>Parallel Polling</u> topic.

IbcLON (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcLON	zero	(Default) Do not enable listen-only mode.
	non- zero	Enable listen-only mode. In this mode, which persists until ibonl is called, the board is forced to listen. While in listen-only mode, the board-level functions ibrd, ibrda, ibrdf, ibwait, and ibtmo function in the usual manner. The behavior of all other functions is undefined.

IbcPAD (ibconfig Board Configuration Option)

Options (Constants) Description

IbcPADChanges the primary address of the board.Default determined by Measurement & Automation
Explorer.

This option sets the primary GPIB address of the board or device to v, an integer ranging from 0 to 30. If no error occurs during the call, Iberr contains the previous GPIB primary address.

IbcPP2 (ibconfig Board Configuration Option)

Options (Constants) Value		Description
IbcPP2	zero	(Default) PP1 moderemote <u>parallel poll</u> configuration.
	non- zero	PP2 modelocal parallel poll configuration.

IbcPPC (ibconfig Board Configuration Option)

Options (Constants) Description

IbcPPC Configures the board for parallel polls. Identical to board-level ibppc.

-

IbcPPollTime (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcPPollTime	0	(Default) Use the standard duration (2 μs) when conducting a parallel poll.
	1 to 17	Use a variable length duration when conducting a parallel poll. The duration represented by 1 to 17 corresponds to the IbcTMO values.

IbcRsv (ibconfig Board Configuration Option)

Options (Constants) Description

IbcRsv Sets the serial poll status byte of the board.

This option is used to request service from the Controller and to provide the Controller with an application-dependent status byte when the Controller serial polls the GPIB interface.

The value v is the status byte that the GPIB interface returns when serial polled by the Controller-In-Charge. If bit 6 (hex 40) is set in v, the GPIB interface requests service from the Controller by asserting the GPIB SRQ line. When IbcRsv is called and an error does not occur, the previous status byte is returned in Iberr.

IbcSAD (ibconfig Board Configuration Option)

Options (Constants) Description

IbcSAD Changes the secondary address of the board. Default determined by <u>Measurement & Automation</u> <u>Explorer</u>.

This option changes the <u>secondary GPIB address</u> of the given board or device to v, an integer in the range 96 to 126 (hex 60 to hex 7E) or zero. If v is zero, secondary addressing is disabled. If no error occurs during the call, the previous value of the GPIB secondary address is returned in Iberr.

IbcSC (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcSC	zero non- zero	The board is not the System Controller. The board is the System Controller.
		Default determined by <u>Measurement &</u> <u>Automation Explorer</u> .

This option requests or releases the capability to send Interface Clear (IFC) and Remote Enable (REN) messages to devices. If v is zero, the board releases system control, and calls requiring System Controller capability are not allowed. If v is non-zero, calls requiring System Controller capability are subsequently allowed. If no error occurs during the call, Iberr contains the previous System Controller state of the board.

IbcSendLLO (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcSendLLO	zero	(Default) Do not send LLO when putting a device online <u>ibfind</u> or <u>ibdev</u> .
		Send LLO when putting a device online <mark>ibfind</mark> or ibdev.
When this option is set to zero, it does not send the CPIR Local Lockout		

When this option is set to zero, it does not send the GPIB Local Lockout (LLO) message when placing a device online. Otherwise, when this option is set to non-zero, it sends the GPIB Lockout (LLO) message when placing a device online. While Local Lockout is in effect, only the Controller-In-Charge can alter the state of the device by sending appropriate GPIB messages. The IbcSendLLO option is reserved for use in unusual local/remote situations.

IbcSRE (ibconfig Board Configuration Option)

Options (Constants)	Value	Description
IbcSRE	zero non-zero	The board is not asserting the REN line. The board is asserting the REN line.

If this option is non-zero, the GPIB Remote Enable (REN) line is asserted. If v is zero, REN is unasserted. The previous value of REN is returned in Iberr.

REN is used by devices to choose between local and remote modes of operation. A device should not actually enter remote mode until it receives its listen address and REN is asserted.

IbcTIMING (ibconfig Board Configuration Option)

Options (Constants) Value Description

IbcTIMING	1	Normal timing (T1 delay of 2 μ s).
	2	High-speed timing (T1 delay of 500 ns).
	3	Very high-speed timing (T1 delay of 350 ns).
	Default determined by <u>Measurement & Automation</u> Explorer.	
The T1 delay is the GPIB Source Handshake tim		L delay is the GPIB Source Handshake timing.

IbcTMO (ibconfig Board Configuration Option)

Options (Constants) Description

IbcTMO Changes the timeout period of the board.

Default determined by <u>Measurement & Automation</u> <u>Explorer</u>.

This option sets the timeout period of the board or device to v. The timeout period is used to select the maximum duration allowed for a synchronous I/O operation (for example, ibrd and ibwrt) or for an ibwait or ibnotify operation with TIMO in the wait mask. If the operation does not complete before the timeout period elapses, the operation is aborted and TIMO is returned in Ibsta. The timeout values listed below represent the minimum timeout period. The actual period could be longer.

Timeout Code Values

Constant	Value of v	Minimum Timeout
TNONE	0	disabled (no timeout)
T10µs	1	10 µs
T30µs	2	30 µs
T100µs	3	100 μs
T300µs	4	300 μs
T1ms	5	1 ms
T3ms	6	3 ms
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms
T300ms	10	300 ms
T1s	11	1 s
T3s	12	3 s
T10s	13	10 s
T30s	14	30 s
T100s	15	100 s
T300s	16	300 s
T1000s	17	1000 s

ibconfig Device Configuration Parameter Options

You can use the following options with *ibconfig* when ud is a device descriptor.

IbcEOS	<u>IbcPAD</u>
<u>IbcEOSchar</u>	IbcREADDR
IbcEOScmp	<u>IbcSAD</u>
IbcEOSrd	<u>IbcSPollTime</u>
<u>IbcEOSwrt</u>	<u>IbcTMO</u>
<u>IbcEOT</u>	<u>IbcUnAddr</u>

Return to ibconfig

IbcEOS (ibconfig Device Configuration Option)

Using this option makes your application break backwards compatibility with older drivers. Consider using IbcEOSchar, IbcEOScmp, IbcEOSrd, and IbcEOSwrt separately instead.

IbcEOS is the parameter that describes the new EOS configuration to use. If v is zero, the EOS configuration is disabled. Otherwise, the low byte is the EOS character and the upper byte contains flags that define the EOS mode. For the upper byte, the third bit flags, terminating the read when EOS is detected, the fourth bit flags, setting EOI with EOS on write functions, and the fifth bit flags, comparing all 8 bits of the EOS byte instead of comparing the lower 7 bits. The following table describes the different EOS configurations:

		Value	e of v
Bit	Configuration	High Byte	Low Byte
A	Terminate read when EOS is detected.	00000100	EOS character
В	Set EOI with EOS on write function.	00001000	EOS character
С	Compare all 8 bits of EOS byte rather than low 7 bits (all read and write calls).	00010000	EOS character

Configuration bits A and C determine how to terminate read I/O operations. If bit A is set and bit C is clear, a read ends when a byte that matches the low seven bits of the EOS character is received. If bits A and C are both set, a read ends when a byte that matches all eight bits of the EOS character is received.

Configuration bits B and C determine when a write I/O operation asserts the GPIB EOI line. If bit B is set and bit C is clear, EOI is asserted when the written character matches the low seven bits of the EOS character. If bits B and C are both set, EOI is asserted when the written character matches all eight bits of the EOS character.

Refer to <u>Terminating Data Transfers</u> for more information about EOS and EOI termination methods.

Examples

ibconfig (ud, IbcEOS 0x140A); /* Configure the software to end reads on newline character (hex 0A) for the unit descriptor, ud */ ibconfig (ud, IbcEOS 0x180A); /* Configure the software to assert the GPIB EOI line whenever the newline character (hex 0A) is written out by the unit descriptor, ud */

IbcEOSchar (ibconfig Device Configuration Option)

Options (Constants) Description

IbcEOSchar Any 8-bit value. This byte becomes the new EOS character.

IbcEOScmp (ibconfig Device Configuration Option)

Options (Constants)	Value	Description
IbcEOScmp	zero non-zero	Use 7 bits for the EOS character comparison. Use 8 bits for the EOS character comparison.

IbcEOSrd (ibconfig Device Configuration Option)

Options (Constants) Value		Description
IbcEOSrd	zero non- zero	Ignore EOS character during read operations. Terminate reads when the EOS character is read.

IbcEOSwrt (ibconfig Device Configuration Option)

Options (Constants) Value Description

IbcEOSwrt	zero	Do not assert the EOI line when the EOS character is sent during write operations.
		Assert the EOI line when the EOS character is sent during write operations.

IbcEOT (ibconfig Device Configuration Option)

Options (Constants) Value Description

IbcEOT	zero	The GPIB EOI line will not be asserted at the end of a write operation.
	non- zero	EOI will be asserted at the end of a write.

This option enables or disables the assertion of the EOI line at the end of write I/O operations for the board or device described by ud. If v is non-zero, EOI is asserted when the last byte of a GPIB write is sent. If v is zero, nothing occurs when the last byte is sent. If no error occurs during the call, the previous value of EOT is returned in Iberr.

Refer to <u>Terminating Data Transfers</u> for more information about EOS and EOI termination methods.

IbcPAD (ibconfig Device Configuration Option)

Options (Constants) Description

IbcPAD Changes the primary address of the device.

This option sets the primary GPIB address of the board or device to v, an integer ranging from 0 to 30. If no error occurs during the call, Iberr contains the previous GPIB primary address.

IbcREADDR (ibconfig Device Configuration Option)

Options (Constants) Value Description

IbcREADDR zero	No unnecessary re-addressing is performed between device-level reads and writes.
	Addressing is always performed before a device- level read or write.

IbcSAD (ibconfig Device Configuration Option)

Options (Constants) Description

IbcSAD Changes the secondary address of the device.

This option changes the <u>secondary GPIB address</u> of the given board or device to v, an integer in the range 96 to 126 (hex 60 to hex 7E) or zero. If v is zero, secondary addressing is disabled. If no error occurs during the call, the previous value of the GPIB secondary address is returned in Iberr.

IbcSPollTime (ibconfig Device Configuration Option)

Options (Constants) Value Description

IbcSPollTime 0 to 17	Sets the length of time the driver waits for a serial poll response byte when polling the given device. The length of time represented by 0 to 17 corresponds to the <u>ibtmo</u> values.
11	Default.

IbcTMO (ibconfig Device Configuration Option)

Options (Constants) Description

IbcTMO Changes the timeout period of the device.

This option sets the timeout period of the board or device to v. The timeout period is used to select the maximum duration allowed for a synchronous I/O operation (for example, ibrd and ibwrt) or for an ibwait or ibnotify operation with TIMO in the wait mask. If the operation does not complete before the timeout period elapses, the operation is aborted and TIMO is returned in Ibsta. The timeout values listed below represent the minimum timeout period. The actual period could be longer.

Timeout Code Values

Constant	Value of v	Minimum Timeout
TNONE	0	disabled (no timeout)
T10µs	1	10 µs
T30µs	2	30 µs
T100µs	3	100 μs
T300µs	4	300 μs
T1ms	5	1 ms
T3ms	6	3 ms
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms
T300ms	10	300 ms
T1s	11	1 s
T3s	12	3 s
T10s	13	10 s
T30s	14	30 s
T100s	15	100 s
T300s	16	300 s
T1000s	17	1000 s

IbcUnAddr (ibconfig Device Configuration Option)

Options (Constants) Value Description

IbcUnAddr	zero	(Default) Do not send Untalk (UNT) and Unlisten (UNL) at the end of device-level reads and writes.
	-	Send UNT and UNL at the end of device-level reads and writes.

IBDEV Device-Level

Purpose

Open and initialize a device descriptor.

Format

С

int ibdev (int BdIndx, int pad, int sad, int tmo, int eot, int eos)

Visual Basic

CALL ibdev (BdIndx%, pad%, sad%, tmo%, eot%, eos%, ud%)

or

ud% = ildev (BdIndx%, pad%, sad%, tmo%, eot%, eos%)

Interactive Control (Usage Notes)

ibdev BdIndx pad sad tmo eot eos

Input

BdIndx	Index of the access board for the device
pad	The primary GPIB address of the device
sad	The secondary GPIB address of the device
tmo	The I/O timeout value
eot	EOI mode of the device
eos	EOS character and modes

Output

Function Return

The device descriptor or a -1

Description

ibdev acquires a device descriptor to use in subsequent device-level traditional NI-488.2 calls. It opens and initializes a device descriptor, and configures it according to the input parameters.

For more details on the meaning and effect of each input parameter, see the <u>IbcPAD</u>, <u>IbcSAD</u>, <u>IbcTMO</u>, <u>IbcEOT</u>, and <u>IbcEOS</u> options in <u>ibconfig</u>.

If ibdev is unable to get a valid device descriptor, a -1 is returned; the ERR bit is set in Ibsta and Iberr contains <u>EDVR</u>.



Note Unit descriptors are allocated on a per process basis, so it is not possible to share them between processes. If you pass a unit descriptor from one process to a second process, all NI-488.2 calls using that descriptor in the second process will return EDVR.

Possible Errors

- EARG pad, sad, tmo, eot, or eos is invalid. See the <u>IbcPAD</u>, <u>IbcSAD</u>, <u>IbcTMO</u>, <u>IbcEOT</u>, and <u>IbcEOS</u> options in <u>ibconfig</u> for details on setting these parameters.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The access board is not CIC. Refer to Device-Level Calls and Bus Management.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBDMA

Board-Level



Note ibdma is deprecated. Use *ibconfig* (IbcDMA) instead.

Purpose

Enable or disable DMA.

Format

С

unsigned long ibdma (int ud, int v)

Visual Basic

CALL ibdma (ud%, v%)

or

status% = ildma (ud%, v%)

Interactive Control (Usage Notes)

ibdma v

Input

- ud Board descriptor
- v Enable or disable the use of DMA

Output

Function Return The value of Ibsta

Description

ibdma enables or disables DMA transfers for the board, according to v. If v is zero, DMA is not used for GPIB I/O transfers. If v is non-zero, DMA is used for GPIB I/O transfers.

Possible Errors

- **ECAP** The interface is not configured to use a DMA channel.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBEOS

Board-Level/Device-Level

Note ibeos is deprecated. Use <u>ibconfig</u> (IbcEOS) instead.

Purpose

Configure the end-of-string (EOS) termination mode or character.

Format

С

unsigned long ibeos (int ud, int v)

Visual Basic

CALL ibeos (ud%, v%)

or

status% = ileos (ud%, v%)

Interactive Control (Usage Notes)

ibeos v

Input

- ud Board or device descriptor
- \mathbf{v} $\;$ EOS mode and character information

Output

Function Return The value of Ibsta

Description

ibeos configures the EOS termination mode or EOS character for the board or device. The parameter v describes the new end-of-string (EOS) configuration to use. If v is zero, the EOS configuration is disabled. Otherwise, the low byte is the EOS character and the upper byte contains flags which define the EOS mode.

 $\overline{\mathbb{N}}$

Note Defining an EOS byte does not cause the driver to automatically send that byte at the end of write I/O operations. Your application is responsible for placing the EOS byte at the end of the data strings that it defines.

The following table describes the different EOS configurations and the corresponding values of v. If no error occurs during the call, the value of the previous EOS setting is returned in Iberr.

		Value of v	
Bit	Configuration	High Byte	Low Byte
A	Terminate read when EOS is detected.	00000100	EOS character
В	Set EOI with EOS on write function.	00001000	EOS character
С	Compare all 8 bits of EOS byte rather than low 7 bits (all read and write calls).	00010000	EOS character

Configuration bits A and C determine how to terminate read I/O operations. If bit A is set and bit C is clear, a read ends when a byte that matches the low seven bits of the EOS character is received. If bits A and C are both set, a read ends when a byte that matches all eight bits of the EOS character is received.

Configuration bits B and C determine when a write I/O operation asserts the GPIB EOI line. If bit B is set and bit C is clear, EOI is asserted when the written character matches the low seven bits of the EOS character. If bits B and C are both set, EOI is asserted when the written character matches all eight bits of the EOS character.

Refer to <u>Terminating Data Transfers</u> for more information about EOS and EOI termination methods.

Examples

ibeos (ud, 0x140A);

/* Configure the software to end reads on
 newline character (hex 0A) for the unit
 descriptor, ud */
ibacs (ud_0u180A);

ibeos (ud, 0x180A);

/* Configure the software to assert the GPIB EOI line whenever the newline character (hex 0A) is written out by the unit descriptor, ud */

Possible Errors

- **EARG** The high byte of v contains invalid bits.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBEOT

Board-Level/Device-Level

Note ibeot is deprecated. Use ibconfig (IbcEOT) instead.

Purpose

Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations.

Format

С

unsigned long ibeot (int ud, int v)

Visual Basic

CALL ibeot (ud%, v%)

or

status% = ileot (ud%, v%)

Interactive Control (Usage Notes)

ibeot v

Input

- ud Board or device descriptor
- \mathbf{v} $\,$ Enables or disables the end of transmission assertion of EOI $\,$

Output

Function Return The value of Ibsta

Description

ibeot enables or disables the assertion of the EOI line at the end of write I/O operations for the board or device described by ud. If v is non-zero, EOI is asserted when the last byte of a GPIB write is sent. If v is zero, nothing occurs when the last byte is sent. If no error occurs during the call, the previous value of EOT is returned in Iberr.

Refer to <u>Terminating Data Transfers</u> for more information about EOS and EOI termination methods.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- ELCK The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBFIND Board-Level/Device-Level

Purpose

Open and initialize a board or a user-configured device descriptor.

Format

С

int ibfind (const char *udname)

Visual Basic

CALL ibfind (udname\$, ud%)

or

ud% = ilfind (udname\$)

Interactive Control (Usage Notes)

ibfind udname

Input

udname A user-configured device or board name

Output

Function Return

The board or device descriptor, or a -1

Description

ibfind is used to acquire a descriptor for a board or user-configured device; this board or device descriptor can be used in subsequent traditional NI-488.2 calls.

ibfind performs the equivalent of an ibonl 1 to initialize the board or device descriptor. The unit descriptor returned by ibfind remains valid until the board or device is put offline using ibonl 0.

If ibfind is unable to get a valid descriptor, a -1 is returned; the ERR bit is set in Ibsta and Iberr contains <u>EDVR</u>.

M

Note Unit descriptors are allocated on a per process basis, so it is not possible to share them between processes. If you pass a unit descriptor from one process to a second process, all NI-488.2 calls using that descriptor in the second process will return EDVR.

Using ibfind to obtain device descriptors is useful only for compatibility with existing applications. New applications should use <u>ibdev</u> instead of ibfind. ibdev is more flexible, easier to use, and frees the application from unnecessary device name requirements.

Possible Errors

- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **ELCK** Another process has an existing lock on the interface.
- **ENEB** The interface is not installed or is not properly configured.

IBGTS Board-Level

Purpose

Go from Active Controller to Standby.

Format

С

unsigned long ibgts (int ud, int v)

Visual Basic

CALL ibgts (ud%, v%)

or

status% = ilgts (ud%, v%)

Interactive Control (Usage Notes)

ibgts v

Input

- ud Board descriptor
- v $\,$ Determines whether to perform acceptor handshaking $\,$

Output

Function Return The value of Ibsta

Description

ibgts causes the GPIB interface at ud to go to Standby Controller and the GPIB ATN line to be unasserted. If v is non-zero, acceptor handshaking or shadow handshaking is performed until END occurs or until ATN is reasserted by a subsequent ibcac call. With this option, the GPIB interface can participate in data handshake as an acceptor without actually reading data. If END is detected, the interface enters a Not Ready For Data (NRFD) handshake holdoff state which results in hold off of subsequent GPIB transfers. If v is 0, no acceptor handshaking or holdoff is performed.

Before performing an ibgts with shadow handshake, call the IbcEOS option in <u>ibconfig</u> to establish proper EOS modes.

For details on the IEEE 488.1 handshake protocol, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation* and ANSI/IEEE Standard 488.1-2003, *IEEE Standard for Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation*.

Possible Errors

- $\ensuremath{\underline{\mathsf{EADR}}}$ v is non-zero, and either ATN is low or the interface is a Talker or a Listener.
- ECIC The interface is not Controller-In-Charge.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBIST

Board-Level



Note ibist is deprecated. Use *ibconfig* (IbcIst) instead.

Purpose

Set or clear the board individual status bit for parallel polls.

Format

С

unsigned long ibist (int ud, int v)

Visual Basic

CALL ibist (ud%, v%)

or

status% = ilist (ud%, v%)

Interactive Control (Usage Notes)

ibist v

Input

- ud Board descriptor
- $\mathbf v$ Indicates whether to set or clear the ist bit

Output

Function Return The value of Ibsta

Description

ibist sets the interface ist (individual status) bit according to v. If v is zero, the ist bit is cleared; if v is non-zero, the ist bit is set. The previous value of the ist bit is returned in Iberr.

For more information about parallel polling, refer to the <u>Parallel Polling</u> topic.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- ELCK The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBLCK Board-Level

Purpose

Acquire or release an exclusive interface lock for the current process.

Format

С

unsigned long iblck (int ud, int v, unsigned int LockWaitTime, void * Reserved)

Visual Basic

CALL iblck (ud%, v%, LockWaitTime&)

or

status% = illck (ud%, v%, LockWaitTime&)

Interactive Control (Usage Notes)

iblck v LockWaitTime

Input

ud	Board unit descriptor or board index descriptor
V	Indicates whether to acquire or release the interface lock
LockWaitTime	Time period (in milliseconds) to wait for an exclusive lock before returning ELCK
Reserved	Reserved for future use; must be NULL

Output

Function Return The value of Ibsta

Description

If v is 1, the driver attempts to acquire an exclusive lock on the interface for the current process. The call fails with ELCK (that is, the call returns with ERR set in Ibsta and Iberr set to ELCK) if the lock could not be acquired within the timeout period specified by LockWaitTime.

If v is 0, a lock previously acquired by the current process is released. Only the process that acquired an interface lock can release a lock. If v is zero, and no lock exists for the process, the call fails with ELCK. LockWaitTime is ignored when using iblck to release interface locks.

If a process has acquired a lock, all GPIB calls by that process for that interface occur normally.

If a process has acquired a lock, all GPIB calls (except iblck and boardlevel ibfind) by other processes for that interface fail immediately with ELCK. iblck calls by other processes attempt to acquire a lock; the call fails with ELCK only after the timeout period has elapsed. Board-level ibfind fails with ELCK but returns a valid unit descriptor.

Interface locks are exclusive and are not shareable among processes. If a process has a lock for an interface, no other process can acquire a lock associated with that interface.

A process may acquire multiple (redundant) locks on an interface. The driver maintains a reference count of the number of outstanding locks per interface and does not unlock the interface until the reference count is 0. If the iblck call is successful, that is, if ERR is not set in Ibsta, Ibcnt contains the number of locks remaining in effect for ud, regardless of the value of v.

An interface lock is associated with a process and a GPIB interface. An acquired lock remains in effect until the lock is released. Each successful call to acquire a lock should have a corresponding call to release the lock. Calling ibonl (0 or 1) releases all interface locks held by that process for the interface described by ud.

A LockWaitTime of 0 specifies a 0 ms wait period. If the interface is locked by another process, then iblck returns immediately with ELCK. Otherwise the process acquires the lock and returns.

Possible Errors

- **EARG** v is not 0 or 1, or Reserved is not NULL.
- **ECAP** Unable to acquire the requested lock because the maximum lock reference count for ud has been reached.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- ELCK Unable to acquire the requested lock within the timeout period because a different process owns a lock on that interface. Or, unable to release a lock because the process currently has no lock for ud.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBLINES Board-Level

Purpose

Return the status of the eight GPIB control lines.

Format

С

unsigned long iblines (int ud, short *clines)

Visual Basic

CALL iblines (ud%, clines%)

or

status% = illines (ud%, clines%)

Interactive Control (Usage Notes)

iblines

Input

ud Board descriptor

Output

clinesReturns GPIB control line state informationFunction ReturnThe value of Ibsta

Description

iblines returns the state of the GPIB control lines in clines. The low-order byte (bits 0 through 7) of clines contains a mask indicating the capability of the GPIB interface to sense the status of each GPIB control line. The upper byte (bits 8 through 15) contains the GPIB control line state information. The following is a pattern of each byte.

7	6	5	4	3	2	1	0
EOI	ATN	SRQ	REN	IFC	NRFD	NDAC	DAV

To determine if a GPIB control line is asserted, first check the appropriate bit in the lower byte to determine if the line can be monitored. If the line can be monitored (indicated by a 1 in the appropriate bit position), check the corresponding bit in the upper byte. If the bit is set (1), the corresponding control line is asserted. If the bit is clear (0), the control line is unasserted.

Example

```
short lines;
iblines (ud, &lines);
if (lines & ValidREN) {
    /* check to see if REN is asserted */
    if (lines & BusREN) {
        printf ("REN is asserted");
    }
}
```

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- ELCK The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBLN Board-Level

F

Purpose

Check for the presence of a device on the bus.

Format

С

unsigned long ibln (int ud, int pad, int sad, short *listen)

Visual Basic

CALL ibln (ud%, pad%, sad%, listen%)

or

status% = illn (ud%, pad%, sad%, listen%)

Interactive Control (Usage Notes)

ibln pad sad

Input

ud Board descriptor

pad The primary GPIB address of the device

sad The secondary GPIB address of the device

Output

listenIndicates if a device is present or notFunction ReturnThe value of Ibsta

Description

ibln determines whether there is a listening device at the GPIB address designated by the pad and sad parameters. The bus associated with the board is tested for <u>Listeners</u>. If a Listener is detected, a non-zero value is returned in listen. If no Listener is found, zero is returned.

The pad parameter can be any valid primary address (a value between 0 and 30). The sad parameter can be any valid secondary address (a value between 96 to 126), or one of the constants NO_SAD or ALL_SAD. The constant NO_SAD designates that no secondary address is to be tested (only a primary address is tested). The constant ALL_SAD designates that all secondary addresses are to be tested.

Possible Errors

- **EARG** Either the pad or sad argument is invalid.
- ECIC The access board is not CIC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBLOC Board-Level/Device-Level

Purpose

Go to Local.

Format

С

unsigned long ibloc (int ud)

Visual Basic

CALL ibloc (ud%) or status% = illoc (ud%)

Interactive Control (Usage Notes)

ibloc

Input

ud Board or device descriptor

Output

Function Return The value of Ibsta

Description

Board-Level

ibloc places the board in local mode if it is not in a lockout state. The board is in a lockout state if LOK appears in the status word <u>lbsta</u>. If the board is in a lockout state, the call has no effect.

The ibloc function is used to simulate a front panel RTL (Return to Local) switch if the computer is used as an instrument.

Device-Level

Unless the REN (Remote Enable) line has been unasserted with the IbcSRE option in <u>ibconfig</u>, all device-level calls automatically place the specified device in remote program mode. ibloc is used to move devices temporarily from a remote program mode to a local mode until the next device function is executed on that device.

Possible Errors

- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> Calls and Bus Management.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBNOTIFY Board-Level/Device-Level

-

Purpose

Notify user of one or more GPIB events by invoking the user callback.

Format for ibnotify

С

unsigned long ibnotify (int ud, int mask, GpibNotifyCallback_t Callback, void *]

Visual Basic (6.0)

CALL ibnotify (ud%, mask%, AddressOf Callback&, RefData\$)

or

status% = ilnotify (ud%, mask%, AddressOf Callback&, RefData\$)

Interactive Control

Not supported

Input for ibnotify

ud Board or device descriptor

mask Bit mask of GPIB events to notice

- Callback Pointer to the callback function (see following prototype)
- RefData User-defined reference data for the callback

Output for ibnotify

Function Return

The value of **Ibsta**

Description of ibnotify

If mask is non-zero, ibnotify monitors the <u>events specified by mask</u>, and when one or more of the events is true, the Callback is invoked. For a board-level ibnotify call, all mask bits are valid except for ERR and RQS. For a device-level ibnotify call, the only valid mask bits are CMPL, TIMO, END, and RQS. If TIMO is set in the notify mask, ibnotify calls the callback function when the timeout period has elapsed, if one or more of the other specified events have not already occurred. If TIMO is not set in the notify mask, the callback is not called until one or more of the specified events occur.



Note Notification is performed when the state of one or more of the **mask** bits is true, so if a request is made to be notified when CMPL is true, and CMPL is currently true, the **Callback** is invoked immediately.

For device-level usage, notification on RQS is not guaranteed to work if automatic serial polling is disabled. By default, automatic serial polling is enabled.

A given ud can have only one outstanding ibnotify call at any one time. If a current ibnotify is in effect for ud, it is replaced by a subsequent ibnotify call. An outstanding ibnotify call for ud can be canceled by a subsequent ibnotify call for ud that has a mask of zero.

If an ibnotify call is outstanding and one or more of the GPIB events it is waiting on become true, the Callback is invoked.

Before placing a board or device (and thus the ud) offline, all outstanding ibnotify calls must be cancelled by a subsequent ibnotify call for ud that has a mask of zero.

Callback Prototype for ibnotify

С

int __stdcall Callback (int LocalUd, unsigned long LocalIbsta, unsigned long LocalIberr, unsigned long LocalIbcnt, void *RefData)

Visual Basic (6.0)

Function Callback (LocalUd%, LocalIbsta%, LocalIberr%, LocalIbcnt&, RefData\$)

Callback Parameters

LocalUd	Board or device descriptor
LocalIbsta	Value of <u>Ibsta</u>
LocalIberr	Value of <u>Iberr</u>
LocalIbcnt	Value of Ibcnt
RefData	User-defined reference data for the callback

Callback Return Value

Bit mask of the GPIB events to notice next.

The Callback function executes in a separate thread in your process. Therefore, it has access to any process global data, but no access to thread local data. If the Callback needs to access global data, you must protect that access using a synchronization primitive (for example, semaphore) because the Callback is running in a different thread context. Alternatively, the issue of data protection can be avoided entirely if the Callback simply posts a message to your application using the Windows PostMessage() function. The Callback function can call any of the NI-488.2 API with the exception of ibnotify. When the Callback is invoked, the values of the NI-488.2 global functions (Ibsta, Iberr, Ibcnt) are undefined. The status variables passed to Callback should be examined, instead of the NI-488.2 globals, to determine why the Callback was invoked. Notice that it is possible that the Callback may be invoked because of an error condition rather than because of the setting of one or more of the requested mask bits.

The return value of the Callback is interpreted as a mask value, which is used to automatically rearm the asynchronous event notification mechanism. If the return value is zero, it is not rearmed. If the return value is non-zero, the asynchronous event notification mechanism is rearmed with the return mask value. If the Callback rearm fails due to an error, the Callback is invoked with ERR set in LocalIbsta and LocalIberr set to EARM.

Like ibwait, ibstop, and ibonl, the invocation of the ibnotify Callback can cause the resynchronization of the handler after an asynchronous I/O operation has completed. In this case, the global variables passed into the Callback after I/O has completed contain the status of the I/O operation.

For an overview of asynchronous event notification in an NI-488.2 application, refer to the <u>Asynchronous Event Notification</u> section. For more information about usage, refer to the <u>ibnotify Usage</u> section.

Possible Errors for ibnotify

- EARG A bit set in mask is invalid.
- **ECAP** ibnotify has been invoked from within an ibnotify Callback function, or the handler cannot perform notification on one or more of the specified mask bits.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed. Ibcnt contains a system-dependent error code.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.

Possible Errors for ibnotify Callback

EARM The Callback return failed to rearm the Callback.

ERST The event notification was cancelled due to a reset of the interface.

Related Topics:

ibnotify Usage ibnotify Usage Example GpibNotify GpibNotify Usage

ibnotify Usage

The ibnotify function is passed a unit descriptor, the bit mask of the desired GPIB events, the address of your callback function, and user-defined reference data. ibnotify has the following prototype:

unsigned long ibnotify (int ud, int mask, GpibNotifyCallback_t Callback, void * RefData)

С

Visual Basic (6.0)

CALL ibnotify (ud%, mask%, AddressOf Callback&, RefData\$)

or

status% = ilnotify (ud%, mask%, AddressOf Callback&, RefData\$)

Input for ibnotify

ud Board or device descriptor

mask Bit mask of GPIB events to notice

Callback Pointer to the Callback function

RefData User-defined reference data for the callback

The ibnotify callback has the following prototype:

С

int __stdcall Callback (int LocalUd, unsigned long LocalIbsta, unsigned long LocalIberr, unsigned long LocalIbcnt, void *RefData)

Visual Basic (6.0)

Function Callback (LocalUd%, LocalIbsta%, LocalIberr%, LocalIbcnt&, RefData\$)

Callback Parameters

LocalUd	Board or device descriptor
LocalIbsta	Value of <u>Ibsta</u>
LocalIberr	Value of <u>Iberr</u>
LocalIbcnt	Value of Ibcnt
RefData	User-defined reference data for the callback

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.

You can view an <u>example written in C of how you can use ibnotify</u> in your application. Assume that your GPIB device is a multimeter that you program to acquire a reading by sending it "SEND DATA". The multimeter requests service when it has a reading ready, and each reading is a floating point value.

In this example, global functions/variables are shared by the Callback thread and the main thread, and the access of the global functions/variables is not protected by synchronization. In this case, synchronization of access to these global functions/variables 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 always just adds information (increases the count or adds another reading to the array of floats).

For overview information about asynchronous event notification within an NI-488.2 application, please refer to the <u>Asynchronous Event Notification</u> <u>Overview</u> section. For more specific information about ibnotify, please refer to the <u>ibnotify</u> function definition.

Note The ibnotify Callback is executed in a separate thread of execution from the rest of your application. If your application might be performing other NI-488.2 operations while it is using ibnotify, you should use the per-thread NI-488.2 global functions/variables that are provided by the **Thread** calls (<u>Threadlbsta</u>, <u>Threadlberr</u>, and <u>Threadlbcnt</u>). In addition, if your application needs to share global functions/variables with the **Callback**, you should use a synchronization primitive (for example, semaphore) to protect access to any global variables. For more information about the use of synchronization primitives, refer to the documentation on using synchronization objects that came with your development tools.

Related Topics:

ibnotify Usage Example ibnotify GpibNotify GpibNotify Usage

ibnotify Usage Example

```
int __stdcall MyCallback (int LocalUd, unsigned long LocalIbsta, unsigned long
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 <u>stdcall MyCallback</u> (int LocalUd, unsigned long LocalIbsta, unsigned long
   unsigned long LocalIbcnt, 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 LocalIbsta, 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 LocalIbsta,
// 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 Ibcnt. 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[LocalIbcnt()] = '\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;
   }
 }
}
```

Related Topics:

<u>ibnotify Usage</u> <u>ibnotify</u> <u>GpibNotify</u> <u>GpibNotify Usage</u>

Notify Mask Layout

Mnemonic	-	Hex Value	Description
ΤΙΜΟ	14	4000	Use the timeout period (see <u>ibconfig</u> , option IbcTMO) to limit the notify period
END	13	2000	END or EOS is detected
SRQI	12	1000	SRQ is asserted (board-level only)
RQS	11	800	Device requested service (device-level only)
CMPL	8	100	I/O is complete
LOK	7	80	GPIB interface is in Lockout State (board-level only)
REM	6	40	GPIB interface is in Remote State (board-level only)
CIC	5	20	GPIB interface is CIC (board-level only)
ATN	4	10	Attention is asserted (board-level only)
TACS	3	8	GPIB interface is Talker (board-level only)
LACS	2	4	GPIB interface is Listener (board-level only)
DTAS	1	2	GPIB interface is in Device Trigger State (board-level only)
DCAS	0	1	GPIB interface is in Device Clear State (board- level only)

Return to ibnotify

■_Return to GpibNotify

GpibNotify Usage

The GpibNotify OLE control is implemented using a method called SetupNotify and an event called Notify. The SetupNotify method is used to enable the NI-488.2 driver to look for one or more GPIB conditions for a particular GPIB handle. After it is set up, the OLE control fires the Notify event when one or more of the GPIB conditions is TRUE. A user-defined callback is invoked when the Notify event is fired.

This section covers the major highlights regarding the sample program that uses the GpibNotify control. The program contains three buttons: **Run**, **Message**, and **Quit**.

Clicking the **Run** button sets into motion a chain of commands that read ten measurements from a Fluke 45 multimeter. First, the program gets a handle to the device. Next, it sends a set of commands that initialize the Fluke 45 multimeter. Then a trigger command is sent. Next the program asks the device to send data. Lastly, it issues a SetupNotify to the GpibNotify OLE control with a mask of the RQS GPIB condition.

When the RQS GPIB condition is TRUE, the Notify event is fired and the user-defined callback is invoked. Each time through the callback, the RearmMask is set to RQS so that the event notification is rearmed for the next RQS GPIB condition. After the callback has read ten measurements from the Fluke 45 multimeter, the RearmMask is set to zero in order to disable the event notification mechanism.

Clicking the **Message** button causes a message to be displayed in a text box every time the button is clicked.

Clicking the **Quit** button closes the program.

The NI-488.2 software includes a sample Visual Basic program. Please refer to the readme.txt file for more detailed information about the project, classes, and code.

For overview information about asynchronous event notification within an NI-488.2 application, please refer to the <u>Asynchronous Event Notification</u> <u>Overview</u> section. For more specific information about the GpibNotify OLE control, please refer to the <u>GpibNotify</u> function definition.

 $\overline{\mathbb{N}}$

Note The GpibNotify OLE control is implemented using the apartment model. Therefore, it only works correctly if your application responds to Windows messages in a timely fashion.

IBONL Board-Level/Device-Level

Purpose

Place the device or interface online or offline.

Format

С

unsigned long ibonl (int ud, int v)

Visual Basic

CALL ibonl (ud%, v%)

or

status% = ilonl (ud%, v%)

Interactive Control (Usage Notes)

ibonl v

Input

- ud Board or device descriptor
- v Indicates whether the board or device is to be placed online or offline

Output

Function Return The value of Ibsta

Description

ibonl resets the board or device and places all its software configuration parameters in their pre-configured state. In addition, if v is zero, the device or interface is placed offline. If v is non-zero, the device or interface is left operational, or online.

If a device or an interface is taken offline, the board or device descriptor (ud) is no longer valid. You must execute an <u>ibdev</u> or <u>ibfind</u> to access the board or device again.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.

IBPAD

Board-Level/Device-Level

Note ibpad is deprecated. Use ibconfig (IbcPAD) instead.

Purpose

Change the primary address.

Format

С

unsigned long ibpad (int ud, int v)

Visual Basic

CALL ibpad (ud%, v%)

or

status% = ilpad (ud%, v%)

Interactive Control (Usage Notes)

 $ibpad \ v$

Input

ud Board or device descriptor

v GPIB primary address

Output

Function Return The value of Ibsta

Description

ibpad sets the primary GPIB address of the board or device to v, an integer ranging from 0 to 30. If no error occurs during the call, Iberr contains the previous GPIB primary address.

Possible Errors

- EARG v is not a valid primary GPIB address; it must be in the range 0 to 30.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBPCT Device-Level

E

Purpose

Pass control to another GPIB device with Controller capability.

Format

С

unsigned long ibpct (int ud)

Visual Basic

CALL ibpct (ud%) or status% = ilpct (ud%)

Interactive Control (Usage Notes)

ibpct

Input

ud Device descriptor

Output

Function Return The value of Ibsta

Description

ibpct passes Controller-in-Charge (CIC) status to the device indicated by ud. The access board automatically unasserts the ATN line and goes to Controller Idle State (CIDS). This function assumes that the device has Controller capability.

Possible Errors

- **EARG** ud is valid but does not refer to a device.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The access board is not CIC. See <u>Device-Level Calls and Bus</u> <u>Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBPPC Board-Level/Device-Level

-

Purpose

Parallel poll configure.

Format

С

unsigned long ibppc (int ud, int v)

Visual Basic

CALL ibppc (ud%, v%)

or

status% = ilppc (ud%, v%)

Interactive Control (Usage Notes)

ibppc v

Input

- ud Board or device descriptor
- v Parallel poll enable/disable value

Output

Function Return The value of Ibsta

Description

Device-Level

If ud is a device descriptor, ibppc enables or disables the device from responding to parallel polls. The device is addressed and sent the appropriate parallel poll messageParallel Poll Enable (PPE) or Disable (PPD). Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero to send PPD.

Board-Level

If ud is a board descriptor, ibppc performs a local parallel poll configuration using the parallel poll configuration value v. Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero to send PPD. If no error occurs during the call, Iberr contains the previous value of the local parallel poll configuration.

For more information about parallel polling, refer to the <u>Parallel Polling</u> section. See <u>IEEE 488 command messages</u> for parallel poll messages.

Possible Errors

- **EARG** v does not contain a valid PPE or PPD message.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- **ECAP** Board-level: The board is not configured to perform local parallel poll configuration. (See <u>ibconfig</u>, option IbcPP2.)
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management</u>.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBRD

Board-Level/Device-Level

Purpose

Read data from a device into a user buffer.

Format

С

unsigned long ibrd (int ud, void *rdbuf, size_t count)

Visual Basic

CALL ibrd (ud%, rdbuf\$)

or

status% = ilrd (ud%, rdbuf\$, count&)

Interactive Control (Usage Notes)

ibrd count

Input

ud Board or device descriptor

count Number of bytes to be read from the GPIB

Output

rdbuf Address of buffer into which data is read Function Return The value of <u>Ibsta</u>

Description

Device-Level

If ud is a device descriptor, ibrd addresses the GPIB, reads up to count bytes of data, and places the data into the buffer specified by rdbuf. The operation terminates normally when count bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period. The actual number of bytes transferred is returned in the global function Ibcnt.

Board-Level

If ud is a board descriptor, ibrd reads up to count bytes of data and places the data into the buffer specified by rdbuf. A board-level ibrd assumes that the GPIB is already properly addressed. The operation terminates normally when count bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not CIC, the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global function Ibcnt.

Possible Errors

- EABO Either count bytes or END was not received within the timeout period or a Device Clear message was received after the read operation began.
- EADR Board-level: The GPIB is not correctly addressed; use ibcmd to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use <u>ibconfig</u>, options IbcPAD and IbcSAD.

- **EARG** Either the buffer or the count is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management.</u>
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- EOIP Asynchronous I/O is in progress.

IBRDA Board-Level/Device-Level

-

Purpose

Read data asynchronously from a device into a user buffer.

Format

С

unsigned long ibrda (int ud, void *rdbuf, size_t count)

Visual Basic

CALL ibrda (ud%, rdbuf\$)

or

status% = ilrda (ud%, rdbuf\$, count&)

Interactive Control (Usage Notes)

ibrda count

Input

ud Board or device unit descriptor

count Number of bytes to be read from the GPIB

Output

rdbuf Address of buffer into which data is read Function Return The value of <u>Ibsta</u>

Description

Device-Level

If ud is a device descriptor, ibrda addresses the GPIB, begins an asynchronous read of up to count bytes of data from a GPIB device, and places the data into the buffer specified by rdbuf. The operation terminates normally when count bytes have been received or END is received. The actual number of bytes transferred is returned in the global function Ibcnt.

Board-Level

If ud is a board descriptor, ibrda reads up to count bytes of data from a GPIB device and places the data into the buffer specified by rdbuf. A board-level ibrda assumes that the GPIB is already properly addressed. The operation terminates normally when count bytes have been received or END is received. The operation terminates with an error if the board is not the CIC, and the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global function Ibcnt.

Board- and Device-Level

The asynchronous I/O calls (ibcmda, ibrda, ibwrta) are designed so that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O has begun, further NI-488.2 calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed; the driver returns <u>EOIP</u> in this case.

Once the I/O is complete, the application must resynchronize with the NI-488.2 driver. Resynchronization is accomplished by using one of the following calls:

- ibwait If the returned Ibsta contains CMPL, the driver and application are resynchronized.
- ibnotify If the Ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- ibstop 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.

Possible Errors

- EABO Board-level: a Device Clear message was received from the CIC.
- EADR Board-level: The GPIB is not correctly addressed; use <u>ibcmd</u> to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use <u>ibconfig</u>, options IbcPAD and IbcSAD.

- **EARG** Either the buffer or the count is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> Calls and Bus Management.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- EOIP Asynchronous I/O is in progress.

IBRDF Board-Level/Device-Level

Purpose

Read data from a device into a file.

Format

С

unsigned long ibrdf (int ud, const char *flname)

Visual Basic

CALL ibrdf (ud%, flname\$)

or

status% = ilrdf (ud%, flname\$)

Interactive Control (Usage Notes)

ibrdf flname

Input

udBoard or device descriptorflnameName of file into which data is read

Output

Function Return The value of Ibsta

Description

Device-Level

If ud is a device descriptor, ibrdf addresses the GPIB, reads data from a GPIB device, and places the data into the file specified by flname. The operation terminates normally when END is received. The operation terminates with an error if the transfer could not complete within the timeout period. The actual number of bytes transferred is returned in the global function Ibcnt.

Board-Level

If ud is a board descriptor, ibrdf reads data from a GPIB device and places the data into the file specified by flname. A board-level ibrdf assumes that the GPIB is already properly addressed. The operation terminates normally when END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not CIC, the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global function Ibcnt.

Possible Errors

- **EABO** END was not received within the timeout period, or ud is a board descriptor and Device Clear was received after the read operation began.
- EADR Board-level: The GPIB is not correctly addressed; use ibcmd to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use <u>ibconfig</u>, options IbcPAD and IbcSAD.

- **EARG** flname is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management.</u>
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EFSO</u>** ibrdf could not access flname.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBRPP

-

Board-Level/Device-Level

Purpose

Conduct a parallel poll.

Format

С

unsigned long ibrpp (int ud, char *ppr)

Visual Basic

CALL ibrpp (ud%, ppr%)

or

status% = ilrpp (ud%, ppr%)

Interactive Control (Usage Notes)

ibrpp

Input

ud Board or device descriptor

Output

pprParallel poll response byteFunction ReturnThe value of Ibsta

Description

ibrpp parallel polls all the devices on the GPIB. The result of this poll is returned in ppr.

For more information about parallel polling, refer to the <u>Parallel Polling</u> <u>Overview</u>.

Possible Errors

- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> Calls and Bus Management.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBRSC

Board-Level



Note ibrsc is deprecated. Use *ibconfig* (IbcSC) instead.

Purpose

Request or release system control.

Format

С

unsigned long ibrsc (int ud, int v)

Visual Basic

CALL ibrsc (ud%, v%)

or

status% = ilrsc (ud%, v%)

Interactive Control (Usage Notes)

ibrsc v

Input

- ud Board or device descriptor
- v Determines if system control is to be requested or released

Output

Function Return The value of Ibsta

Description

ibrsc requests or releases the capability to send Interface Clear (IFC) and Remote Enable (REN) messages to devices. If v is zero, the board releases system control, and calls requiring System Controller capability are not allowed. If v is non-zero, calls requiring System Controller capability are subsequently allowed. If no error occurs during the call, Iberr contains the previous System Controller state of the board.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- ELCK The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBRSP

E

Device-Level

Purpose

Conduct a serial poll.

Format

С

unsigned long ibrsp (int ud, char *spr)

Visual Basic

CALL ibrsp (ud%, spr%)

or

status% = ilrsp (ud%, spr%)

Interactive Control (Usage Notes)

ibrsp

Input

ud Device descriptor

Output

sprSerial poll response byteFunction ReturnThe value of Ibsta

Description

The ibrsp function is used to serial poll the device ud. The serial poll response byte is returned in spr. If bit 6 (hex 40) of the response is set, the device is requesting service. When the automatic serial polling feature is enabled, the device might have already been polled. In this case, ibrsp returns the previously acquired status byte.

For more information about serial polling, refer to the <u>Serial Polling</u> <u>Overview</u>.

Possible Errors

- EABO The serial poll response could not be read within the serial poll timeout period.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The access board is not CIC. See <u>Device-Level Calls and Bus</u> <u>Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBRSV

Board-Level



Note ibrsv is deprecated. Use ibconfig (IbcRSV) instead.

Purpose

Request service and change the serial poll status byte.

Format

С

unsigned long ibrsv (int ud, int v)

Visual Basic

CALL ibrsv (ud%, v%)

or

status% = ilrsv (ud%, v%)

Interactive Control (Usage Notes)

ibrsv v

Input

ud Board descriptor

v Serial poll status byte

Output

Function Return The value of Ibsta

Description

ibrsv is used to request service from the Controller and to provide the Controller with an application-dependent status byte when the Controller serial polls the GPIB interface.

The value v is the status byte that the GPIB interface returns when serial polled by the Controller-In-Charge. If bit 6 (hex 40) is set in v, the GPIB interface requests service from the Controller by asserting the GPIB SRQ line. When ibrsv is called and an error does not occur, the previous status byte is returned in Iberr.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- ELCK The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBSAD

Board-Level/Device-Level

Note ibsad is deprecated. Use <u>ibconfig</u> (IbcSAD) instead.

Purpose

Change or disable the secondary address.

Format

С

unsigned long ibsad (int ud, int v)

Visual Basic

CALL ibsad (ud%, v%)

or

status% = ilsad (ud%, v%)

Interactive Control (Usage Notes)

ibsad v

Input

- ud Board or device descriptor
- v GPIB secondary address

Output

Function Return The value of Ibsta

Description

ibsad changes the <u>secondary GPIB address</u> of the given board or device to v, an integer in the range 96 to 126 (hex 60 to hex 7E) or zero. If v is zero, secondary addressing is disabled. If no error occurs during the call, the previous value of the GPIB secondary address is returned in Iberr.

Possible Errors

- **EARG** v is non-zero and outside the legal range 96 to 126.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBSIC Board-Level

Purpose

Assert interface clear.

Format

С

unsigned long ibsic (int ud)

Visual Basic

CALL ibsic (ud%) or status% = ilsic (ud%)

Interactive Control (Usage Notes)

ibsic

Input

ud Board descriptor

Output

Function Return The value of Ibsta

Description

ibsic asserts the GPIB interface clear (IFC) line for at least 100 μ s, then unasserts IFC. Asserting IFC unaddresses all devices on the bus and makes the interface board CIC.

The IFC signal resets only the GPIB interface calls of bus devices and not the internal device calls. Consult your device documentation to determine how to reset the internal calls of your device.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **ESAC** The board does not have System Controller capability.

IBSRE

Board-Level



Note ibsre is deprecated. Use ibconfig (IbcSRE) instead.

Purpose

Set or clear the Remote Enable (REN) line.

Format

С

unsigned long ibsre (int ud, int v)

Visual Basic

CALL ibsre (ud%, v%)

or

status% = ilsre (ud%, v%)

Interactive Control (Usage Notes)

ibsre v

Input

- ud Board descriptor
- $\mathbf v$ $\;$ Indicates whether to set or clear the REN line $\;$

Output

Function Return The value of Ibsta

Description

If v is non-zero, the GPIB Remote Enable (REN) line is asserted. If v is zero, REN is unasserted. The previous value of REN is returned in ${\rm Iberr.}$

REN is used by devices to choose between local and remote modes of operation. A device should not actually enter remote mode until it receives its listen address and REN is asserted.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **ESAC** The board does not have System Controller capability.

IBSTOP Board-Level/Device-Level

-

Purpose

Abort asynchronous I/O operation.

Format

С

unsigned long ibstop (int ud)

Visual Basic

CALL ibstop (ud%)

or

status% = ilstop (ud%)

Interactive Control (Usage Notes)

ibstop

Input

ud Board or device descriptor

Output

Function Return The value of Ibsta

Description

The ibstop function aborts any asynchronous read, write, or command operation that is in progress and resynchronizes the application with the driver. If asynchronous I/O is in progress, the error bit is set in the status word, Ibsta, and <u>EABO</u> is returned, indicating that the I/O was successfully stopped.

Possible Errors

- EABO Asynchronous I/O was successfully stopped.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.

IBTMO

Board-Level/Device-Level

Note ibtmo is deprecated. Use <u>ibconfig</u> (IbcTMO) instead.

Purpose

Change or disable the timeout period.

Format

С

unsigned long ibtmo (int ud, int v)

Visual Basic

CALL ibtmo (ud%, v%)

or

status% = iltmo (ud%, v%)

Interactive Control (Usage Notes)

ibtmo v

Input

ud Board or device descriptor

v <u>Timeout duration code</u>

Output

Function Return The value of Ibsta

Description

ibtmo sets the timeout period of the board or device to v. The timeout period is used to select the maximum duration allowed for a synchronous I/O operation (for example, ibrd and ibwrt) or for an ibwait or ibnotify operation with TIMO in the wait mask. If the operation does not complete before the timeout period elapses, the operation is aborted and TIMO is returned in Ibsta. The timeout values represent the minimum timeout period. The actual period could be longer.

Possible Errors

EARG v is invalid.

- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

Timeout Code Values

Constant	Value of v	Minimum Timeout
TNONE	0	disabled (no timeout)
T10µs	1	10 µs
T30µs	2	30 μs
T100µs	3	100 µs
T300µs	4	300 µs
T1ms	5	1 ms
T3ms	6	3 ms
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms
T300ms	10	300 ms
T1s	11	1 s
T3s	12	3 s
T10s	13	10 s
T30s	14	30 s
T100s	15	100 s
T300s	16	300 s
T1000s	17	1000 s

Return to ibtmo

IBTRG Device-Level

Purpose

Trigger selected device.

Format

С

unsigned long ibtrg (int ud)

Visual Basic

CALL ibtrg (ud%) or status% = iltrg (ud%)

Interactive Control (Usage Notes)

ibtrg

Input

ud Device descriptor

Output

Function Return The value of Ibsta

Description

ibtrg sends the Group Execute Trigger (GET) message to the device described by ud.

Possible Errors

- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The access board is not CIC. See <u>Device-Level Calls and Bus</u> <u>Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

IBWAIT Board-Level/Device-Level

Purpose

Wait for GPIB events.

Format

С

unsigned long ibwait (int ud, int mask)

Visual Basic

CALL ibwait (ud%, mask%)

or

status% = ilwait (ud%, mask%)

Interactive Control (Usage Notes)

ibwait mask

Input

ud Board or device descriptor

mask Bit mask of GPIB events to wait for

Output

Function Return The value of Ibsta

Description

ibwait monitors the <u>events specified by mask</u> and delays processing until one or more of the events occurs. If the wait mask is zero, ibwait returns immediately with the updated Ibsta status word. If TIMO is set in the wait mask, ibwait returns when the timeout period has elapsed, if one or more of the other specified events have not already occurred. If TIMO is not set in the wait mask, the function waits indefinitely for one or more of the specified events to occur. The existing ibwait mask bits are identical to the Ibsta bits. If ud is a device descriptor, the only valid wait mask bits are TIMO, END, RQS, and CMPL. If ud is a board descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period using the <u>ibconfig</u> function (option IbcTMO).

Possible Errors

- **EARG** The bit set in mask is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- ESRQ Device-level: If RQS is set in the wait mask, ESRQ indicates that the <u>Stuck SRQ condition</u> exists. For more information about serial polling, refer to the <u>Serial Polling Overview</u>.
- **EWIP** An ibwait call is already in progress on the specified unit descriptor.
- **ERST** The event notification was cancelled due to a reset of the interface.

Wait Mask Layout

Mnemonic		Hex Value	Description
ΤΙΜΟ	14	4000	Use the timeout period (see <u>ibconfig</u> , option IbcTMO) to limit the notify period
END	13	2000	END or EOS is detected
SRQI	12	1000	SRQ is asserted (board-level only)
RQS	11	800	Device requested service (device-level only)
CMPL	8	100	I/O is completed
LOK	7	80	GPIB interface is in Lockout State (board-level only)
REM	6	40	GPIB interface is in Remote State (board-level only)
CIC	5	20	GPIB interface is CIC (board-level only)
ATN	4	10	Attention is asserted (board-level only)
TACS	3	8	GPIB interface is Talker (board-level only)
LACS	2	4	GPIB interface is Listener (board-level only)
DTAS	1	2	GPIB interface is in Device Trigger State (board-level only)
DCAS	0	1	GPIB interface is in Device Clear State (board- level only)

Return to ibwait

IBWRT Board-Level/Device-Level

Purpose

Write data to a device from a user buffer.

Format

С

unsigned long ibwrt (int ud, const void *wrtbuf, size_t count)

Visual Basic

CALL ibwrt (ud%, wrtbuf\$)

or

status% = ilwrt (ud%, wrtbuf\$, count&)

Interactive Control (Usage Notes)

ibwrt wrtbuf

Input

ud Board or device descriptor

wrtbuf Address of the buffer containing the bytes to write

count Number of bytes to be written

Output

Function Return The value of Ibsta

Description

Device-Level

If ud is a device descriptor, ibwrt addresses the GPIB and writes count bytes from the memory location specified by wrtbuf to a GPIB device. The operation terminates normally when count bytes have been sent. The operation terminates with an error if count bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the global function Ibcnt.

Board-Level

If ud is a board descriptor, ibwrt writes count bytes of data from the buffer specified by wrtbuf to a GPIB device; a board-level ibwrt assumes that the GPIB is already properly addressed. The operation terminates normally when count bytes have been sent. The operation terminates with an error if count bytes could not be sent within the timeout period or, if the board is not CIC, the CIC sends Device Clear on the GPIB. The actual number of bytes transferred is returned in the global function Ibcnt.

Possible Errors

- EABO Either count bytes were not sent within the timeout period, or a Device Clear message was received after the write operation began.
- EADR Board-level: The GPIB is not correctly addressed; use ibcmd to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use the IbcPAD and IbcSAD options in <u>ibconfig</u>.

- **EARG** Either the buffer or the count is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> Calls and Bus Management.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners were detected on the bus.
- **EOIP** Asynchronous I/O is in progress.

IBWRTA Board-Level/Device-Level

Purpose

Write data asynchronously to a device from a user buffer.

Format

С

unsigned long ibwrta (int ud, const void *wrtbuf, size_t count)

Visual Basic

CALL ibwrta (ud%, wrtbuf\$)

or

status% = ilwrta (ud%, wrtbuf\$, count&)

Interactive Control (Usage Notes)

ibwrta wrtbuf

Input

ud Board or device descriptor

wrtbuf Address of the buffer containing the bytes to write

count Number of bytes to be written

Output

Function Return

The value of **Ibsta**

Description

Device-Level

If ud is a device descriptor, ibwrta addresses the GPIB properly and writes count bytes from wrtbuf to a GPIB device. The operation terminates normally when count bytes have been sent. The actual number of bytes transferred is returned in the global function Ibcnt.

Board-Level

If ud is a board descriptor, ibwrta begins an asynchronous write of count bytes of data from wrtbuf to a GPIB device. A board-level ibwrta assumes that the GPIB is already properly addressed. The operation terminates normally when count bytes have been sent. The operation terminates with an error if the board is not the CIC, and the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global function Ibcnt.

Board- and Device-Level

The asynchronous I/O calls (ibcmda, ibrda, ibwrta) are designed so that applications can perform other non-GPIB operations with the I/O in progress. Once the asynchronous I/O begins, further NI-488.2 calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed; the driver returns EOIP in this case.

Once the I/O is complete, the application must resynchronize with the NI-488.2 driver. Resynchronization is accomplished by using one of the following calls:

- ibwait If the returned Ibsta contains CMPL, the driver and application are resynchronized.
- ibnotify If the Ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- ibstop 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.

Possible Errors

- EABO Board-level: A Device Clear message was received from the CIC.
- EADR Board-level: The GPIB is not correctly addressed; use <u>ibcmd</u> to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use the IbcPAD and IbcSAD options in <u>ibconfig</u>.

- **EARG** Either the buffer or the count is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- EHDL ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners were detected on the bus.
- **EOIP** Asynchronous I/O is in progress.

IBWRTF Board-Level/Device-Level

-

Purpose

Write data to a device from a file.

Format

С

unsigned long ibwrtf (int ud, const char *flname)

Visual Basic

CALL ibwrtf (ud%, flname\$)

or

status% = ilwrtf (ud%, flname\$)

Interactive Control (Usage Notes)

ibwrtf flname

Input

ud Board or device descriptor

flname Name of file containing the data to be written

Output

Function Return The value of Ibsta

Description

Device-Level

If ud is a device descriptor, ibwrtf addresses the GPIB and writes all of the bytes from the file flname to a GPIB device. The operation terminates normally when all of the bytes have been sent. The operation terminates with an error if all of the bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the global function Ibcnt.

Board-Level

If ud is a board descriptor, ibwrtf writes all of the bytes of data from the file flname to a GPIB device. A board-level ibwrtf assumes that the GPIB is already properly addressed. The operation terminates normally when all of the bytes have been sent. The operation terminates with an error if all of the bytes could not be sent within the timeout period, or if the board is not CIC, the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global function Ibcnt.

Possible Errors

- EABO Either the file could not be transferred within the timeout period, or a Device Clear message was received after the write operation began.
- EADR Board-level: The GPIB is not correctly addressed; use ibcmd to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use the IbcPAD and IbcSAD options in <u>ibconfig</u>.

- **EARG** flname is invalid.
- **<u>EBUS</u>** Device-level: No devices are connected to the GPIB.
- ECIC Device-level: The access board is not CIC. See <u>Device-Level</u> <u>Calls and Bus Management</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EFSO</u>** ibwrtf could not access flname.
- **<u>EHDL</u>** ud is invalid or out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

Multi-Device NI-488.2 Calls

AllSpoll	Sorial poll all instruments
AllSpoll	Serial poll all instruments
<u>DevClear</u>	Clear a single instrument
<u>DevClearList</u>	Clear multiple instruments
<u>EnableLocal</u>	Enable operations from the front panel of instruments (leave remote programming mode)
EnableRemote	Enable remote GPIB programming for instruments
<u>FindLstn</u>	Find listening instruments on the GPIB
<u>FindRQS</u>	Determine which instrument is requesting service
PassControl	Pass control to another instrument with Controller capability
<u>PPoll</u>	Perform a parallel poll on the GPIB
PPollConfig	Configure an instrument to respond to parallel polls
<u>PPollUnconfig</u>	Unconfigure instruments for parallel polls
<u>RcvRespMsg</u>	Read data bytes from an instrument that is already addressed to talk
ReadStatusByte	Serial poll a single instrument
<u>Receive</u>	Read data bytes from an instrument
<u>ReceiveSetup</u>	Address an instrument to be a Talker and the interface to be a Listener in preparation for RcvRespMsg
<u>ResetSys</u>	Reset and initialize IEEE 488.2-compliant instruments
<u>Send</u>	Send data bytes to an instrument
SendCmds	Send GPIB command bytes
<u>SendDataBytes</u>	Send data bytes to instruments that are already addressed to listen
SendIFC	Reset the GPIB by sending interface clear
<u>SendList</u>	Send data bytes to multiple GPIB instruments
<u>SendLLO</u>	Send the Local Lockout (LLO) message to all instruments

<u>SendSetup</u>	Set up instruments to receive data in preparation for SendDataBytes
<u>SetRWLS</u>	Place instruments in Remote With Lockout State
TestSRQ	Determine the current state of the GPIB Service Request (SRQ) line
<u>TestSys</u>	Cause IEEE 488.2-compliant instruments to conduct self tests
<u>Trigger</u>	Trigger an instrument
<u>TriggerList</u>	Trigger multiple instruments
<u>WaitSRQ</u>	Wait until an instrument asserts the GPIB Service Request (SRQ)

Supplemental Calls for Multithreaded 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 NI-488.2 global functions (Ibsta, Iberr, Ibcnt). The NI-488.2 global functions are defined on a per-process basis, so each process accesses its own private copy of the NI-488.2 global functions.

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 NI-488.2 global functions without some form of synchronization (for example, a semaphore). To understand why this is true, take a look at 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 global functions, 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 global functions. 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.

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

Use synchronization to protect access to process-global functions.

Do not use process-global functions.

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

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

unsigned long <u>ThreadIbsta();</u> // return thread-specific Ibsta() unsigned long <u>ThreadIberr();</u> // return thread-specific Iberr() unsigned long <u>ThreadIbcnt();</u> // return thread-specific Ibcnt()

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

if (Ibsta() & ERR)

could be replaced by

if (ThreadIbsta() & ERR)

Note If you are using ibnotify in your application, 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 per-thread NI-488.2 global functions through ThreadIbsta , ThreadIberr , and ThreadIbcnt , instead of the per-process NI-488.2 global functions.

IEEE 488 Command Messages

These multiline interface messages are sent and received with ATN asserted.

For more information about these messages, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*, and ANSI/IEEE Standard 488.1-2003, *IEEE Standard for Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation*.

Multiline Interface Message Definitions

Hex Decimal ASCII Message

0	NUL	
1	SOH	GTL
2	STX	
3	ETX	
4	EOT	SDC
5	ENQ	PPC
6	ACK	
7	BEL	
8	BS	GET
9	HT	ТСТ
10	LF	
11	VT	
12	FF	
13	CR	
14	SO	
15	SI	
16	DLE	
17	DC1	LLO
18	DC2	
	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17	1SOH2STX3ETX4EOT5ENQ6ACK7BEL8BS9HT10LF11VT12FF13CR14SO15SI16DLE17DC1

13	19	DC3	
14	20	DC4	DCL
15	21	NAK	PPU
16	22	SYN	
17	23	ETB	
18	24	CAN	SPE
19	25	EM	SPD
1A	26	SUB	
1B	27	ESC	
1C	28	FS	
1D	29	GS	
1E	30	RS	
1F	31	US	CFE
20	32	SP	MLA0
21	33	!	MLA1
22	34		MLA2
23	35	#	MLA3
24	36	\$	MLA4
25	37	%	MLA5
26	38	&	MLA6
27	39	ı	MLA7
28	40	(MLA8
29	41)	MLA9
2A	42	*	MLA10
2B	43	+	MLA11
2C	44	,	MLA12
2D	45	-	MLA13
2E	46		MLA14
2F	47	/	MLA15

30	48	0	MLA16
31	49	1	MLA17
32	50	2	MLA18
33	51	3	MLA19
34	52	4	MLA20
35	53	5	MLA21
36	54	6	MLA22
37	55	7	MLA23
38	56	8	MLA24
39	57	9	MLA25
ЗA	58	:	MLA26
3B	59	;	MLA27
3C	60	<	MLA28
3D	61	=	MLA29
3E	62	>	MLA30
3F	63	?	UNL
40	64	@	MTA0
41	65	А	MTA1
42	66	В	MTA2
43	67	С	MTA3
44	68	D	MTA4
45	69	Е	MTA5
46	70	F	MTA6
47	71	G	MTA7
48	72	Н	MTA8
49	73	I	MTA9
4A	74	J	MTA10
4B	75	К	MTA11
4C	76	L	MTA12

4D	77	М	MTA13
4E	78	Ν	MTA14
4F	79	0	MTA15
50	80	Ρ	MTA16
51	81	Q	MTA17
52	82	R	MTA18
53	83	S	MTA19
54	84	Т	MTA20
55	85	U	MTA21
56	86	V	MTA22
57	87	W	MTA23
58	88	Х	MTA24
59	89	Y	MTA25
5A	90	Z	MTA26
5B	91	[MTA27
5C	92	١	MTA28
5D	93]	MTA29
5E	94	Λ	MTA30
5F	95	_	UNT
60	96	`	MSA0, PPE
61	97	а	MSA1, PPE, CFG1
62	98	b	MSA2, PPE, CFG2
63	99	С	MSA3, PPE, CFG3
64	100	d	MSA4, PPE, CFG4
65	101	е	MSA5, PPE, CFG5
66	102	f	MSA6, PPE, CFG6
67	103	g	MSA7, PPE, CFG7
68	104	h	MSA8, PPE, CFG8
69	105	i	MSA9, PPE, CFG9

6A	106	j	MSA10, PPE, CFG10
6B	107	, k	MSA11, PPE, CFG11
6C	108	I	MSA12, PPE, CFG12
		•	, ,
6D	109	m	MSA13, PPE, CFG13
6E	110	n	MSA14, PPE, CFG14
6F	111	0	MSA15, PPE, CFG15
70	112	р	MSA16, PPD
71	113	q	MSA17, PPD
72	114	r	MSA18, PPD
73	115	S	MSA19, PPD
74	116	t	MSA20, PPD
75	117	u	MSA21, PPD
76	118	V	MSA22, PPD
77	119	W	MSA23, PPD
78	120	Х	MSA24, PPD
79	121	у	MSA25, PPD
7A	122	Z	MSA26, PPD
7B	123	{	MSA27, PPD
7C	124		MSA28, PPD
7D	125	}	MSA29, PPD
7E	126	~	MSA30, PPD
7F	127	DEL	

Multiline Interface Message Definitions

- CFE Configuration Enable (This multiline interface message is part of the IEEE 488.1-2003 specification and supports the HS488 highspeed protocol.)
- CFG Configure (This multiline interface message is part of the IEEE 488.1-2003 specification and supports the HS488 high-speed protocol.)
- DCL Device Clear
- GET Group Execute Trigger
- GTL Go To Local
- LLO Local Lockout
- MLA My Listen Address
- MSA My Secondary Address
- MTA My Talk Address
- PPC Parallel Poll Configure
- PPD Parallel Poll Disable
- PPE Parallel Poll Enable
- PPU Parallel Poll Unconfigure
- SDC Selected Device Clear
- SPD Serial Poll Disable
- SPE Serial Poll Enable
- TCT Take Control
- **UNL** Unlisten
- UNT Untalk



Prefixes

Symbol	Prefix	Value
р	pico	10 -12
n	nano	10 ⁻⁹
μ	micro	10 -6
m	milli	10 ⁻³
k	kilo	10 ³
М	mega	10 6
G	giga	10 ⁹
Т	tera	10 12

Α

acceptor handshake	Listeners use this GPIB interface function to receive data, and all devices use it to receive commands. See <u>source handshake</u> and <u>handshake</u> .
access board	The GPIB interface that controls and communicates with the devices on the bus that are attached to it.
ANSI	American National Standards Institute.
API	Application Programming Interface.
application interface	Formerly called <i>language interface</i> . Code that enables an application program that uses NI-488.2 calls to access the driver.
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 NI-488.2 software in which serial polls are executed automatically by the driver whenever a device asserts the GPIB SRQ line.

В

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

С

- 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 that 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. See <u>handshake</u>.
- DCL Device Clear. The GPIB command used to reset the device or internal functions of all devices. See <u>SDC</u>.

device- A function that combines several rudimentary board operations level into one function so that the user does not have to be

function concerned with bus management or other GPIB protocol matters.

DIO1 The GPIB lines that are used to transmit command or data through bytes from one device to another.

- DIO8
- DLL Dynamic link library.
- DMA Direct memory access. High-speed data transfer between the GPIB interface and memory that is not handled directly by the CPU. Not available on some systems. See <u>programmed I/O</u>.
- driver Device driver software installed within the operating system.

Ε

END or A message that signals the end of a data string. END is sentby asserting the GPIB End or Identify (EOI) line with the lastdata byte.

EOI A GPIB line that is used to signal either the last byte of a data message (END) or the parallel poll Identify (IDY) message.

EOS or A 7- or 8-bit end-of-string character that is sent as the last EOS Byte byte of a data message.

- EOT End of transmission.
- ESB The Event Status bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.

G

- GET Group Execute Trigger. It is 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, ANSI/IEEE Standard 488.1-2003, ANSI/IEEE Standard 488.2-1987, and ANSI/IEEE Standard 488.2-1992.

GPIB The address of a device on the GPIB, composed of a primary address (MLA and MTA) and perhaps a secondary address (MSA). The GPIB interface has both a GPIB address and an I/O address.

GPIB Refers to the National Instruments family of GPIB interfaces. interface

GTL Go To Local. It is the GPIB command used to place an addressed Listener in local (front panel) control mode.

Н

- handshake The mechanism used to transfer bytes from the Source Handshake function of one device to the Acceptor Handshake function of another device. The three GPIB lines DAV, NRFD, and NDAC 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. For more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.1-2003.
- hex Hexadecimal; a number represented in base 16. For example, decimal 16 = hex 10.

high-level See <u>device-level function</u>.

function

HS488 High-speed data transfer protocol, part of the IEEE 488.1-2003 specification.

Hz Hertz.

I

- ibcnt After each NI-488.2 I/O call, this global variable contains the actual number of bytes transmitted. On systems with a 16-bit integer, such as MS-DOS, ibcnt is a 16-bit integer, and ibcntl is a 32-bit integer. For cross-platform compatibility, use ibcntl, unless using the newer NI4882 API. For accessing the newer NI4882 API, use the global function, Ibcnt, instead.
- Ibcnt After each NI-488.2 call, this global function contains the actual number of bytes transmitted. The Ibcnt function returns a 32-bit integer. For accessing the newer NI4882 API, this function is recommended instead of the global variables, ibcnt and ibcntl.
- ibcntl After each NI-488.2 I/O call, this global variable contains the actual number of bytes transmitted. On systems with a 16-bit integer, such as MS-DOS, ibcnt is a 16-bit integer, and ibcntl is a 32-bit integer. For cross-platform compatibility, use ibcntl, unless using the newer NI4882 API. For accessing the newer NI4882 API, use the global function, Ibcnt, instead.
- iberr A global variable that contains the specific error code associated with a function call that failed. For accessing the newer NI4882 API, use the global function, Iberr, instead.
- Iberr A global function that contains the specific error code associated with a function call that failed. For accessing the newer NI4882 API, this function is recommended instead of the global variable, iberr.
- ibsta At the end of each function call, this global variable (status word) contains status information. For accessing the newer NI4882 API, use the global function, Ibsta, instead.
- Ibsta At the end of each function call, this global function contains status information. For accessing the newer NI4882 API, this function is recommended instead of the global variable, ibsta.
- IEEE Institute of Electrical and Electronic Engineers.

interface A broadcast message sent from the Controller to all devices message and used to manage the GPIB.

- I/O Input/output. In the context of this documentation, the transmission of commands or messages between the computer through the GPIB interface and other devices on the GPIB.
- I/O The address of the GPIB interface from the point of view of
- address the CPU, as opposed to the GPIB address of the GPIB interface. Also called port address or board address.
- ist An Individual Status bit of the status byte used in the Parallel Poll Configure function.

L

LAD Listen address. See <u>MLA</u>.

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- A rudimentary board or device function that performs a single operation.

function

Μ	
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.
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.

Ν

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

Ρ

- parallel poll The process of polling all configured devices at once and reading a composite poll response. See <u>serial poll</u>.
- PIO See 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.

programmedLow-speed data transfer between the GPIB interface andI/Omemory in which the CPU moves each data byte
according to program instructions. See DMA.

R

resynchronize The NI-488.2 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. See <u>DCL</u>.
- 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. See <u>parallel poll</u>.

service See <u>SRQ</u>.

request

- source The GPIB interface function that transmits data and commands. Talkers use this function to send data, and the Controller uses it to send commands. See <u>acceptor</u> <u>handshake</u> and <u>handshake</u>.
- 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. See <u>SPD</u>.
- 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 See <u>lbsta</u>.

- synchronous Refers to the relationship between the NI-488.2 calls and a process when executing driver calls is predictable; the process is blocked until the driver completes the function.
- System The single designated Controller that can assert control Controller (become CIC of the GPIB) by sending the Interface Clear (IFC) message. Other devices can become CIC only by having control passed to them.

Т

TAD Talk Address. See <u>MTA</u>.

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 NI-488.2 driver that prevents I/O calls 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.

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 THERETOFORE 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.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about <u>National Instruments trademarks</u>.

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **HelpPatents** in your software, the patents.txt file on your media, or <u>ni.com/patents</u>.

Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- <u>Support</u>—Technical support resources at ni.com/support include the following:
 - Self-Help Resources—For answers and solutions, visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-bystep troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
 - Standard Service Program Membership—This program entitles members to direct access to NI Applications Engineers via phone and email for one-toone technical support, as well as exclusive access to on demand training modules via the <u>Services Resource</u> <u>Center</u>. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

For information about other <u>technical support options</u> in your area, visit ni.com/services or <u>contact</u> your local office at ni.com/contact.

- <u>Training and Certification</u>—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- <u>System Integration</u>—If you have time constraints, limited inhouse technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your <u>local office</u> or NI corporate headquarters. You also can visit the <u>Worldwide Offices</u> section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

FindLstn

Purpose

Find listening devices on the GPIB.

Format

С

void FindLstn (int boardID, const Addr4882_t *padlist, short *resultlist, size_t
limit)

Visual Basic

CALL FindLstn (boardID%, padlist%(), resultlist%(), limit%)

Interactive Control (Usage Notes)

FindLstn padlist limit

Input

boardID The interface number

- padlist A list of primary addresses that is terminated by NOADDR
- limit Total number of entries that can be placed in resultlist

Output

resultlist Addresses of all listening devices found by FindLstn are placed in this array

Description

FindLstn tests all of the primary addresses in padlist as follows: If a device is present at a primary address given in padlist, the primary address is stored in resultlist. Otherwise, all <u>secondary addresses</u> of the primary address are tested, and the addresses of any devices found are stored in resultlist. No more than limit addresses are stored in resultlist. Ibcnt contains the actual number of addresses stored in resultlist.

Possible Errors

- **EARG** An invalid primary address appears in padlist; Ibcnt is the index of the first invalid address in the padlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **ETAB** The number of devices found on the GPIB exceed limit.

AllSpoll

Purpose

Serial poll all devices.

Format

С

void AllSpoll (int boardID, const Addr4882_t *addrlist, short *resultlist)

Visual Basic

CALL AllSpoll (boardID%, addrlist%(), resultlist%())

Interactive Control (Usage Notes)

AllSpoll addrlist

Input

boardID The interface number

addrlist A list of device addresses terminated by NOADDR

Output

resultlist A list of serial poll response bytes corresponding to device addresses in addrlist

Description

AllSpoll serial polls all of the devices described by addrlist. It stores the poll responses in resultlist and the number of responses in Ibcnt. For more information about serial polling, refer to the <u>Serial Polling Overview</u>.

Possible Errors

- **EABO** One of the devices timed out instead of responding to the serial poll; Ibcnt contains the index of the timed-out device.
- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

DevClear

-

Purpose

Clear a single device.

Format

С

void DevClear (int boardID, Addr4882_t address)

Visual Basic

CALL DevClear (boardID%, address%)

Interactive Control (Usage Notes)

DevClear address

Input

boardIDThe interface numberaddressAddress of the device you want to clear

Description

DevClear sends the Selected Device Clear (SDC) GPIB message to the device described by address. If address is the constant NOADDR, the Universal Device Clear (DCL) message is sent to all devices.

Possible Errors

- **EARG** The address parameter does not contain a valid address.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

DevClearList

Purpose

Clear multiple devices.

Format

С

void DevClearList (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL DevClearList (boardID%, addrlist%())

Interactive Control (Usage Notes)

DevClearList addrlist

Input

boardID The interface number

addrlist A list of device addresses terminated by NOADDR that you want to clear

Description

DevClearList sends the Selected Device Clear (SDC) GPIB message to all the device addresses described by addrlist. If addrlist contains only the constant NOADDR, the Universal Device Clear (DCL) message is sent to all the devices on the bus.

Possible Errors

- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

EnableLocal

E

Purpose

Enable operations from the front panel of devices (leave remote programming mode).

Format

С

void EnableLocal (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL EnableLocal (boardID%, addrlist%())

Interactive Control (Usage Notes)

EnableLocal addrlist

Input

boardID The interface number

addrlist A list of device addresses that is terminated by NOADDR

Description

EnableLocal sends the Go To Local (GTL) GPIB message to all the devices described by addrlist. This places the devices into local mode. If addrlist contains only the constant NOADDR, the Remote Enable (REN) GPIB line is unasserted.

Possible Errors

- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **<u>ESAC</u>** The interface is not configured as System Controller.

EnableRemote

E

Purpose

Enable remote GPIB programming for devices.

Format

С

void EnableRemote (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL EnableRemote (boardID%, addrlist%())

Interactive Control (Usage Notes)

EnableRemote addrlist

Input

boardID The interface number

addrlist A list of device addresses that is terminated by NOADDR

Description

EnableRemote asserts the Remote Enable (REN) GPIB line. All devices described by addrlist are put into a listen-active state.

Possible Errors

- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **<u>ESAC</u>** The interface is not configured as System Controller.

FindRQS

E

Purpose

Determine which device is requesting service.

Format

С

void FindRQS (int boardID, const Addr4882_t *addrlist, short *result)

Visual Basic

CALL FindRQS (boardID%, addrlist%(), result%)

Interactive Control (Usage Notes)

FindRQS addrlist

Input

boardID The interface number

addrlist List of device addresses that is terminated by NOADDR

Output

result Serial poll response byte of the device that is requesting service

Description

FindRQS serial polls the devices described by addrlist, in order, until it finds a device which is requesting service. The serial poll response byte is then placed in result. Ibcnt contains the index of the device requesting service in addrlist. If none of the devices are requesting service, the index corresponding to NOADDR in addrlist is returned in Ibcnt and ETAB is returned in Iberr. For more information about serial polling, refer to the <u>Serial Polling Overview</u>.

Possible Errors

- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- EOIP Asynchronous I/O is in progress.
- **ETAB** None of the devices in addrlist are requesting service or addrlist contains only NOADDR. Ibcnt contains the index of NOADDR in addrlist.

PassControl

Purpose

Pass control to another device with Controller capability.

Format

С

void PassControl (int boardID, Addr4882_t address)

Visual Basic

CALL PassControl (boardID%, address%)

Interactive Control (Usage Notes)

PassControl address

Input

boardID The interface number

address Address of the device to which you want to pass control

Description

PassControl sends the Take Control (TCT) GPIB message to the device described by address. The device becomes Controller-In-Charge (CIC) and the interface is no longer CIC.

Possible Errors

- **EARG** The address parameter is invalid. It must be a valid primary/secondary address pair. It cannot be the constant NOADDR.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

PPoll

Purpose

Perform a parallel poll on the GPIB.

Format

С

void PPoll (int boardID, short *result)

Visual Basic

CALL PPoll (boardID%, result%)

Interactive Control (Usage Notes)

PPoll

Input

boardID

The interface number

Output

result The parallel poll result

Description

PPoll conducts a parallel poll and the result is placed in result. Each of the eight bits of result represents the status information for each device configured for a parallel poll. The interpretation of the status information is based on the latest parallel poll configuration command sent to each device (see <u>PPollConfig</u> and <u>PPollUnconfig</u>). The Controller can use parallel polling to obtain one-bit, device-dependent status messages from up to eight devices simultaneously.

For more information about parallel polling, refer to the <u>Parallel Polling</u> <u>Overview</u>.

Possible Errors

- **EARG** result does not point to a valid memory location.
- **<u>ECIC</u>** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

PPollConfig

Purpose

Configure a device to respond to parallel polls.

Format

С

void PPollConfig (int boardID, Addr4882_t address, int dataline, int lineSense)

Visual Basic

CALL PPollConfig (boardID%, address%, dataline%, lineSense%)

Interactive Control (Usage Notes)

PPollConfig address dataline lineSense

Input

boardID The interface number

- address Address of the device to be configured
- dataline Data line (a value in the range of 1 to 8) on which the device responds to parallel polls

lineSense Sense (either 0 or 1) of the parallel poll response

Description

PPollConfig configures the device described by address to respond to parallel polls by asserting or not asserting the GPIB data line, dataline. If lineSense equals the individual status (ist) bit of the device, the assigned GPIB data line is asserted during a parallel poll. Otherwise, the data line is not asserted during a parallel poll. The Controller can use parallel polling to obtain 1-bit, device-dependent status messages from up to eight devices simultaneously.

For more information about parallel polling, refer to the <u>Parallel Polling</u> <u>Overview</u>.

Possible Errors

- **EARG** Either the address parameter is invalid, dataline is not in the range 1 to 8, or lineSense is not 0 or 1. The address must be a valid primary/secondary address pair. It cannot be the constant NOADDR.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- EOIP Asynchronous I/O is in progress.

PPollUnconfig

Purpose

Unconfigure devices for parallel polls.

Format

С

void PPollUnconfig (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL PPollUnconfig (boardID%, addrlist%())

Interactive Control (Usage Notes)

PPollUnconfig addrlist

Input

boardID The interface number

addrlist A list of device addresses that is terminated by NOADDR

Description

PPollUnconfig unconfigures all the devices described by addrlist for parallel polls. If addrlist contains only the constant NOADDR, the Parallel Poll Unconfigure (PPU) GPIB message is sent to all GPIB devices. The devices unconfigured by this function do not participate in subsequent parallel polls.

For more information about parallel polling, refer to the <u>Parallel Polling</u> <u>Overview</u>.

Possible Errors

- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

RcvRespMsg

Purpose

Read data bytes from a device that is already addressed to talk.

Format

С

void RcvRespMsg (int boardID, void *buffer, size_t count, int termination)

Visual Basic

CALL RcvRespMsg (boardID%, buffer\$, termination%)

Interactive Control (Usage Notes)

RcvRespMsg count termination

Input

boardID The interface number

count Number of bytes read

termination Description of the data termination mode (STOPend or an 8bit EOS character)

Output

buffer Stores the received data bytes

Description

RcvRespMsg reads up to count bytes from the GPIB and places these bytes into buffer. Data bytes are read until either count data bytes have been read or the termination condition is detected. If the termination condition is STOPend, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when the 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global function, Ibcnt.

RcvRespMsg assumes that the interface is already in its listen-active state and a device is already addressed to be a Talker (see <u>ReceiveSetup</u> or <u>Receive</u>).

Possible Errors

- EABO The I/O timeout period elapsed before all the bytes were received.
- **EADR** The interface is not in the listen-active state; use **ReceiveSetup** to address the GPIB properly.
- EARG The termination parameter is invalid. It must be either STOPend or an 8-bit EOS character.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

ReadStatusByte

E

Purpose

Serial poll a single device.

Format

С

void ReadStatusByte (int boardID, Addr4882_t address, short *result)

Visual Basic

CALL ReadStatusByte (boardID%, address%, result%)

Interactive Control (Usage Notes)

ReadStatusByte address

Input

boardID address

The interface number A device address

Output

result Serial poll response byte

Description

ReadStatusByte serial polls the device described by address. The response byte is stored in result.

Possible Errors

- **EABO** The device times out instead of responding to the serial poll.
- **EARG** The address parameter is invalid.
- **EBUS** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

Receive

Purpose

Read data bytes from a device.

Format

С

void Receive (int boardID, Addr4882_t address, void *buffer, size_t count, int termination)

Visual Basic

CALL Receive (boardID%, address%, buffer\$, termination%)

Interactive Control (Usage Notes)

Receive address count termination

Input

boardID The interface number

address Address of a device to receive data

count Number of bytes to read

termination Description of the data termination mode (STOPend or an EOS character)

Output

buffer Stores the received data bytes

Description

Receive addresses the device described by address to talk and the interface to listen. Then up to count bytes are read and placed into the buffer. Data bytes are read until either count bytes have been read or the termination condition is detected. If the termination condition is STOPend, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when an 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global function, Ibcnt.

Possible Errors

- EABO The I/O timeout period elapsed before all the bytes were received.
- EARG The address or termination parameter is invalid. The address must be a valid primary/secondary address pair. It cannot be the constant NOADDR.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

ReceiveSetup

Purpose

Address a device to be a Talker and the interface to be a Listener in preparation for RcvRespMsg.

Format

С

void ReceiveSetup (int boardID, Addr4882_t address)

Visual Basic

CALL ReceiveSetup (boardID%, address%)

Interactive Control (Usage Notes)

ReceiveSetup address

Input

boardIDThe interface numberaddressAddress of a device to be talk addressed

Description

ReceiveSetup makes the device described by address talk-active, and makes the interface listen-active. This call is usually followed by a call to RcvRespMsg to transfer data from the device to the interface. This call is particularly useful to make multiple calls to RcvRspMsg; it eliminates the need to readdress the device to receive every block of data.

Possible Errors

- **EARG** The address parameter is invalid.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

ResetSys

Purpose

Reset and initialize IEEE 488.2-compliant devices.

Format

С

void ResetSys (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL ResetSys (boardID%, addrlist%())

Interactive Control (Usage Notes)

ResetSys addrlist

Input

boardID The interface number

addrlist A list of device addresses that is terminated by NOADDR

Description

The reset and initialization take place in three steps. The first step resets the GPIB by asserting the Remote Enable (REN) line and then the Interface Clear (IFC) line. The second step clears all of the devices by sending the Universal Device Clear (DCL) GPIB message. The final step causes IEEE 488.2-compliant devices to perform device-specific reset and initialization. This step is accomplished by sending the message "*RST\n" to the devices described by addrlist.

Possible Errors

- EABO I/O operation is aborted
- **EARG** Either an invalid address appears in addrlist or addrlist is empty; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners are on the GPIB.
- **EOIP** Asynchronous I/O is in progress.
- **ESAC** The board is not System Controller.

Send

Purpose

Send data bytes to a device.

Format

С

void Send (int boardID, Addr4882_t address, const void *buffer, size_t count, int eotmode)

Visual Basic

CALL Send (boardID%, address%, buffer\$, eotmode%)

Interactive Control (Usage Notes)

Send address buffer eotmode

Input

- boardID The interface number
- address Address of a device to which data is sent
- buffer The data bytes to be sent
- count Number of bytes to be sent
- eotmode The data termination mode: DABend, NULLend, or NLend

Description

Send addresses the device described by address to listen and the interface to talk. Then count bytes from buffer are sent to the device. The last byte is sent with the EOI line asserted if eotmode is DABend. The last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend then a new line character ('\n') is sent with the EOI line asserted after the last byte of buffer. The actual number of bytes transferred is returned in the global function, Ibcnt.

Possible Errors

- EABO The I/O timeout period has expired before all of the bytes were sent.
- **EARG** Either the address parameter or eotmode parameter is invalid, or the buffer is empty and eotmode is DABend. The address must be a valid primary/secondary address pair; it cannot be the constant NOADDR. The eotmode parameter can only be DABend, NULLend, or NLend.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners are on the GPIB to accept the data bytes.
- **EOIP** Asynchronous I/O is in progress.

SendCmds

Purpose

Send GPIB command bytes.

Format

С

void SendCmds (int boardID, const void *buffer, size_t count)

Visual Basic

CALL SendCmds (boardID%, buffer\$)

Interactive Control (Usage Notes)

SendCmds buffer

Input

boardIDThe interface numberbufferCommand bytes to be sentcountNumber of bytes to be sent

Description

SendCmds sends count command bytes from buffer over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global function Ibcnt. Refer to the <u>IEEE 488</u> <u>command messages</u> for defined interface messages.

Use command bytes to configure the state of the GPIB, not to send instructions to GPIB devices. Use <u>Send</u> or <u>SendList</u> to send device-specific instructions.

Possible Errors

- EABO The I/O timeout period expired before all of the command bytes were sent.
- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **EHDL** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No devices are connected to the GPIB.
- **EOIP** Asynchronous I/O is in progress.

SendDataBytes

Purpose

Send data bytes to devices that are already addressed to listen.

Format

С

void SendDataBytes (int boardID, const void *buffer, size_t count, int eotmode)

Visual Basic

CALL SendDataBytes (boardID%, buffer\$, eotmode%)

Interactive Control (Usage Notes)

SendDataBytes buffer eotmode

Input

- boardID The interface number
- buffer The data bytes to be sent
- count Number of bytes to be sent
- eotmode The data termination mode: DABend, NULLend, or NLend

Description

SendDataBytes sends count number of bytes from the buffer to devices which are already addressed to listen. The last byte is sent with the EOI line asserted if eotmode is DABend; the last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend then a new line character ('\n') is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global function, Ibcnt.

SendDataBytes assumes that the interface is in talk-active state and that devices are already addressed as Listeners on the GPIB (see <u>SendSetup</u>, <u>Send</u>, or <u>SendList</u>).

Possible Errors

- **EABO** The I/O timeout period expired before all of the bytes were sent.
- EADR The interface is not talk-active; use SendSetup to address the GPIB properly.
- EARG Either the eotmode parameter is invalid (it can only be DABend, NULLend, or NLend), or the buffer is empty and the eotmode is DABend.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners are on the GPIB to accept the data bytes; use SendSetup to address the GPIB properly.
- **EOIP** Asynchronous I/O is in progress.

SendIFC

-

Purpose

Reset the GPIB by sending interface clear.

Format

С

void SendIFC (int boardID)

Visual Basic

CALL SendIFC (boardID%)

Interactive Control (Usage Notes)

SendIFC

Input

boardID

The interface number

Description

SendIFC is used as part of GPIB initialization. It forces the interface to be Controller-In-Charge of the GPIB. It also ensures that the connected devices are all un-addressed and that the interface calls of the devices are in their idle states.

Possible Errors

- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- ESAC The interface is not configured as the System Controller; see <u>ibconfig</u>, option IbcSC.

SendList

Purpose

Send data bytes to multiple GPIB devices.

Format

С

void SendList (int boardID, const Addr4882_t *addrlist, const void *buffer, size_t count, int eotmode)

Visual Basic

CALL SendList (boardID%, addrlist%(), buffer\$, eotmode%)

Interactive Control (Usage Notes)

SendList addrlist buffer eotmode

Input

- boardID The interface number
- addrlist A list of device addresses to send data
- buffer The data bytes to be sent
- count Number of bytes transmitted
- eotmode The data termination mode: DABend, NULLend, or NLend

Description

SendList addresses the devices described by addrlist to listen and the interface to talk. Then, count bytes from buffer are sent to the devices. The last byte is sent with the EOI line asserted if eotmode is DABend. The last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend, a new line character ('\n') is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global function, Ibcnt.

Possible Errors

- **EABO** The I/O timeout period expired before all of the bytes were sent.
- **EARG** Either an invalid address appears in addrlist or the addrlist is empty (Ibcnt is the index of the first invalid address), or the eotmode parameter is invalid. The eotmode parameter can only be DABend, NULLend, or NLend. If the buffer is empty, an eotmode of DABend is disallowed.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **<u>ECIC</u>** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

SendLLO

-

Purpose

Send the Local Lockout (LLO) message to all devices.

Format

С

void SendLLO (int boardID)

Visual Basic

CALL SendLLO (boardID%)

Interactive Control (Usage Notes)

SendLLO

Input

boardID

The interface number

Description

SendLLO sends the GPIB Local Lockout (LLO) message to all devices. While Local Lockout is in effect, only the Controller-In-Charge can alter the state of the devices by sending appropriate GPIB messages. SendLLO is reserved for use in unusual local/remote situations. In the typical case of placing the devices in Remote With Local Lockout, use <u>SetRWLS</u>.

Possible Errors

- **EBUS** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **ESAC** The interface is not configured as System Controller.

SendSetup

Purpose

Set up devices to receive data in preparation for SendDataBytes.

Format

С

void SendSetup (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL SendSetup (boardID%, addrlist%())

Interactive Control (Usage Notes)

Send addrlist

Input

boardID The interface number

addrlist A list of device addresses that is terminated by NOADDR

Description

SendSetup makes the devices described by addrlist listen-active and makes the interface talk-active. This call is usually followed by <u>SendDataBytes</u> to actually transfer data from the interface to the devices. SendSetup is particularly useful to set up the addressing before making multiple calls to SendDataBytes; it eliminates the need to readdress the devices for every block of data.

Possible Errors

- **EARG** Either an invalid address appears in addrlist or the addrlist is empty; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

SetRWLS

Purpose

Place devices in Remote With Lockout State.

Format

С

void SetRWLS (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL SetRWLS (boardID%, addrlist%())

Interactive Control (Usage Notes)

SetRWLS addrlist

Input

boardID The interface number

addrlist List of device addresses that is terminated by NOADDR

Description

SetRWLS places the devices described by addrlist in remote mode by asserting the Remote Enable (REN) GPIB line. Then those devices are placed in lockout state by the Local Lockout (LLO) GPIB message. You cannot program those devices locally until the Controller-In-Charge releases the Local Lockout by way of the EnableLocal call.

Possible Errors

- **EARG** Either an invalid address appears in addrlist or the addrlist is empty; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
- **ESAC** The interface is not configured as System Controller.

TestSRQ

Purpose

Determine the current state of the GPIB Service Request (SRQ) line.

Format

С

void TestSRQ (int boardID, short *result)

Visual Basic

CALL TestSRQ (boardID%, result%)

Interactive Control (Usage Notes)

TestSRQ

Input

boardID

The interface number

Output

result State of the SRQ line: non-zero if the line is asserted, zero if the line is not asserted

Description

TestSRQ returns the current state of the GPIB SRQ line in result. If SRQ is asserted, result contains a non-zero value. Otherwise, result contains a zero. Use TestSRQ to get the current state of the GPIB SRQ line. Use <u>WaitSRQ</u> to wait until SRQ is asserted.

Possible Errors

- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- EDVR The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

TestSys

E -

Purpose

Cause IEEE 488.2-compliant devices to conduct self tests.

Format

С

void TestSys (int boardID, const Addr4882_t *addrlist, short *resultlist)

Visual Basic

CALL TestSys (boardID%, addrlist%(), resultlist%())

Interactive Control (Usage Notes)

TestSys addrlist

Input

boardID The interface number

addrlist A list of device addresses terminated by NOADDR

Output

resultlist A list of test results; each entry corresponds to an address in addrlist

Description

TestSys sends the "*TST?" message to the IEEE 488.2-compliant devices described by addrlist. The "*TST?" message instructs them to conduct their self-test procedures. A 16-bit test result code is read from each device and stored in resultlist. A test result of $0\n$ indicates that the device passed its self test. Refer to the documentation that came with the device to determine the meaning of the failure code. Any other value indicates that the device failed its self test. If the function returns without an error (that is, the ERR bit is not set in Ibsta), Ibcnt contains the number of services that failed. Otherwise, the meaning of Ibcnt depends on the error returned. If a device fails to send a response before the timeout period expires, a test result of 1 is reported for it, and the error EABO is returned.

Possible Errors

- **EABO** The interface timed out before receiving a result from a device; Ibcnt contains the index of the timed-out device. -1 is stored as the test result for the timed-out device.
- EARG Either an invalid address appears in addrlist or the addrlist is empty; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **ENOL** No Listeners are on the GPIB.
- **EOIP** Asynchronous I/O is in progress.

Trigger

Purpose

Trigger a device.

Format

С

void Trigger (int boardID, Addr4882_t address)

Visual Basic

CALL Trigger (boardID%, address%)

Interactive Control (Usage Notes)

Trigger address

Input

boardIDThe interface numberaddressAddress of a device to be triggered

Description

Trigger sends the Group Execute Trigger (GET) GPIB message to the device described by address. If address is the constant NOADDR, the GET message is sent to all devices that are currently listen-active on the GPIB.

Possible Errors

- **EARG** The address parameter is invalid.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see <u>SendIFC</u>.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

TriggerList

Purpose

Trigger multiple devices.

Format

С

void TriggerList (int boardID, const Addr4882_t *addrlist)

Visual Basic

CALL TriggerList (boardID%, addrlist%())

Interactive Control (Usage Notes)

TriggerList addrlist

Input

boardID The interface number

addrlist A list of device addresses terminated by NOADDR

Description

TriggerList sends the Group Execute Trigger (GET) GPIB message to the devices described by addrlist. If the only address in addrlist is the constant NOADDR, no addressing is performed and the GET message is sent to all devices that are currently listen-active on the GPIB.

Possible Errors

- **EARG** An invalid address appears in addrlist; Ibcnt is the index of the first invalid address in the addrlist array.
- **<u>EBUS</u>** No devices are connected to the GPIB.
- ECIC The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

WaitSRQ

Purpose

Wait until a device asserts the GPIB Service Request (SRQ) line.

Format

С

void WaitSRQ (int boardID, short *result)

Visual Basic

CALL WaitSRQ (boardID%, result%)

Interactive Control (Usage Notes)

WaitSRQ

Input

boardID

The interface number

Output

result State of the SRQ line: non-zero if line is asserted, zero if line is not asserted

Description

WaitSRQ waits until either the GPIB SRQ line is asserted or the timeout period has expired (see <u>ibconfig</u>, option IbcTMO). When WaitSRQ returns, result contains a non-zero if SRQ is asserted. Otherwise, result contains a zero. Use <u>TestSRQ</u> to get the current state of the GPIB SRQ line. Use WaitSRQ to wait until SRQ is asserted.

Possible Errors

- **ECIC** The interface is not able to detect the state of the SRQ line because it is not the Controller-In-Charge.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed.
- **<u>EHDL</u>** boardID is out of range.
- **ELCK** The requested operation could not be performed because of an existing lock by another process.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.

ThreadIbsta

Purpose

Return the value of the thread-specific <u>Ibsta</u>.

Format

С

unsigned long ThreadIbsta ()

Visual Basic

rc% = ThreadIbsta ()

Input

none No input parameters

Output

Function ReturnValue of Ibsta for the calling thread

Description

ThreadIbsta returns the current value of Ibsta for a particular thread of execution. The global NI-488.2 status functions (Ibsta, Iberr, Ibcnt) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status variables are maintained on a per thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status functions using ThreadIbsta, ThreadIberr, and ThreadIbcnt instead of the global NI-488.2 status functions.

Return to Supplemental Calls for Multithreaded NI-488.2 Applications

Related Topics: ibnotify GpibNotify Threadlbcnt

ThreadIberr

ThreadIberr

1

Purpose

Return the value of the thread-specific <u>Iberr</u>.

Format

С

unsigned long ThreadIberr ()

Visual Basic

rc% = ThreadIberr ()

Input

none No input parameters

Output

Function ReturnValue of Iberr for the calling thread

Description

ThreadIberr returns the current value of Iberr for a particular thread of execution. The global NI-488.2 status functions (Ibsta, Iberr, Ibcnt) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status variables are maintained on a per thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status functions using ThreadIbsta, ThreadIberr, and ThreadIbcnt instead of the global NI-488.2 status functions.

Return to Supplemental Calls for Multithreaded NI-488.2 Applications

Related Topics: ibnotify GpibNotify

<u>ThreadIbcnt</u>

<u>ThreadIbsta</u>

ThreadIbcnt

-

Purpose

Return the value of the thread-specific Ibcnt.

Format

С

unsigned long ThreadIbcnt ()

Visual Basic

rc% = ThreadIbcnt ()

Input

none No input parameters

Output

Function ReturnValue of Ibcnt for the calling thread

Description

ThreadIbcnt returns the current value of Ibcnt for a particular thread of execution. The global NI-488.2 status functions (Ibsta, Iberr, Ibcnt) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status functions are maintained on a per-thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status functions using <u>ThreadIbsta</u>, <u>ThreadIberr</u>, and ThreadIbcnt instead of the global NI-488.2 status functions.

Return to Supplemental Calls for Multithreaded NI-488.2 Applications

Related Topics: ibnotify GpibNotify ThreadIberr

<u>ThreadIbsta</u>

Multithreaded NI-488.2 Applications

Multithreading issues are beyond the scope of this help file. For information on this topic, please refer to the application note, *Developing Multithreaded GPIB Applications Using NI-488.2*, on the <u>NI Developer</u> <u>Zone</u> at zone.ni.com.

GpibNotify Board-Level/Device-Level

Purpose

Notify user of one or more GPIB events by invoking the user callback. GpibNotify is an OLE control.

Format for the GpibNotify OLE Control

Visual Basic

status& = GpibNotify<x>.SetupNotify ud%, mask%

where $\langle x \rangle$ is the instance of the GpibNotify OLE control.

Interactive Control

Not supported

Input for the GpibNotify OLE Control

ud Board or device descriptor

mask Bit mask of GPIB events to notice

The mask parameter is optional. Alternative ways to set the mask value include using the SetSetupMask method or changing its value on the SetupMask page of the GpibNotify control properties.

Output for the GpibNotify OLE Control

Function Return

The value of <u>lbsta</u>

Description of the GpibNotify OLE Control

If mask is non-zero, GpibNotify monitors the <u>events specified by mask</u>, and when one or more of the events is true, the Callback is invoked. For a board-level GpibNotify call, all mask bits are valid except for ERR and RQS. For a device-level GpibNotify call, the only valid mask bits are CMPL, TIMO, END, and RQS.



Note Notification is performed when the state of one or more of the **mask** bits is true, so if a request is made to be notified when CMPL is true, and CMPL is currently true, the **Callback** is invoked immediately.

For device-level usage, notification on RQS is not guaranteed to work if automatic serial polling is disabled. By default, automatic serial polling is enabled.

A given ud can have only one outstanding GpibNotify call at any one time. If a current GpibNotify is in effect for ud, it is replaced by a subsequent GpibNotify call. An outstanding GpibNotify call for ud can be canceled by a subsequent GpibNotify call for ud that has a mask of zero.

If a GpibNotify call is outstanding and one or more of the GPIB events it is waiting on become true, the Callback is invoked.

Note After you make a SetupNotify call, the global NI-488.2 status functions (Ibsta, Iberr, and Ibcnt) are undefined. Instead, use the thread-specific NI-488.2 status variable calls (ThreadIbsta, ThreadIberr, and ThreadIbcnt), to examine the NI-488.2 status variables returned by the SetupNotify call. This restriction applies only to the SetupNotify call; for the rest of the NI-488.2 calls, you can continue to examine Ibsta, Iberr, and Ibcnt.

Visual Basic exhibits odd behavior if the **GpibNotify** control is destroyed before the **Callback** has been executed. For this reason, cancel any outstanding Callbacks by calling **SetupNotify** with a mask of zero before the control is destroyed. In addition, your application should give any blocked **Callback** threads an opportunity to run before destroying the control by executing a **Sleep 0** call.

Callback Prototype for the GpibNotify OLE Control

Private Sub GpibNotify<x>_Notify(ByVal LocalUd As Long, ByVal LocalIbsta As Long, ByVal LocalIberr As Long, ByVal LocalIbcnt As Long, RearmMask As Long)

where $\langle x \rangle$ is the instance of the Notify callback routine. Each GpibNotify call has its own Callback routine.

Callback Parameters

LocalUd	Board or device descriptor	
LocalIbsta	Value of Ibsta	
LocalIberr	Value of Iberr	
LocalIbcnt	Value of Ibcnt	
RearmMask	Bit mask of the GPIB events to notice	

The Callback function can call any of the NI-488.2 calls with the exception of GpibNotify. When the Callback is invoked, the values of the <u>NI-488.2</u> global functions (Ibsta, Iberr, and Ibcnt) are undefined. The status functions passed to Callback should be examined, instead of the NI-488.2 globals, to determine why the Callback was invoked. Notice that it is possible that the Callback may be invoked because of an error condition rather than because of the setting of one or more of the requested mask bits.

next

The RearmMask is interpreted as a mask value that the NI-488.2 software uses to automatically rearm the asynchronous event notification mechanism. If RearmMask is set to zero, the Callback is not rearmed. If RearmMask is set to non-zero, the Callback is rearmed with the RearmMask value. If the Callback rearm fails due to an error, the Callback is invoked with ERR set in LocalIbsta and LocalIberr set to EARM.

Like ibwait, ibstop, and ibonl, the invocation of the GpibNotify Callback can cause the resynchronization of the handler after an asynchronous I/O operation has completed. In this case, the global variables passed into the Callback after I/O has completed contain the status of the I/O operation.

For an overview of asynchronous event notification in an NI-488.2 application, refer to the <u>Asynchronous Event Notification</u> section. For more information about usage, refer to the <u>GpibNotify Usage</u> section.

Possible Errors for the GpibNotify OLE Control

- **EARG** A bit set in mask is invalid.
- **ECAP** GpibNotify has been invoked from within an GpibNotify Callback function, or the handler cannot perform notification on one or more of the specified mask bits.
- **EDVR** The NI-488.2 driver is either configured incorrectly or is not properly installed. Ibcnt contains a system-dependent error code.
- **ENEB** The interface is not installed or is not properly configured.

Possible Errors for the GpibNotify OLE Control Callback

EARM The GpibNotify OLE control was unable to rearm the Callback.

Related Topics: ibnotify ibnotify Usage ibnotify Usage Example

Using NI Spy

NI Spy monitors, records, and displays the NI-488.2 calls made from applications. You can use it to troubleshoot errors in your application and to verify the communication with your GPIB instrument. NI Spy shows which NI-488.2 calls are being used to communicate with your instrument. If your application is not working properly, you can use NI Spy to search for failed NI-488.2 calls. For more information, refer to its help.

To start NI Spy, select **Start»Programs»National Instruments»NI Spy**. Or click on the following button.

Start NI Spy.

Branch Offices

Office	Telephone Number
Australia	1800 300 800
Austria	43 662 457990-0
Belgium	32 (0) 2 757 0020
Brazil	55 11 3262 3599
Canada	800 433 3488
China	86 21 5050 9800
Czech Republic	420 224 235 774
Denmark	45 45 76 26 00
Finland	358 (0) 9 725 72511
France	33 (0) 1 57 66 24 24
Germany	49 89 7413130
India	91 80 41190000
Israel	972 0 3 6393737
Italy	39 02 41309277
Japan	0120-527196 / 81 3 5472 2970
Korea	82 02 3451 3400
Lebanon	961 (0) 1 33 28 28
Malaysia	1800 887710
Mexico	01 800 010 0793
Netherlands	31 (0) 348 433 466
New Zealand	0800 553 322
Norway	47 (0) 66 90 76 60
Poland	48 22 3390150
Portugal	351 210 311 210
Russia	7 495 783 6851
Singapore	1800 226 5886
Slovenia	386 3 425 42 00

South Africa	27 0 11 805 8197	
Spain	34 91 640 0085	
Sweden	46 (0) 8 587 895 00	
Switzerland	41 56 2005151	
Taiwan	886 02 2377 2222	
Thailand	662 278 6777	
Turkey	90 212 279 3031	
United Kingdom	44 (0) 1635 523545	
United States (Corporate) 512 683 0100		