NI-488.2™

NI-488.2 Function Reference Manual for Windows

NATIONAL INSTRUMENTS™

June 1999 Edition
Part Number 321038G-01

Worldwide Technical Support and Product Information

www.natinst.com

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, China 0755 3904939, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00,
Singapore 2265886, Spain (Madrid) 91 640 0085, Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@natinst.com.

Important Information

Warranty

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

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

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

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT 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

Trademarks

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

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

Contents

About This Manual

Chapter 1
NI-488.2 Traditional Calls

Chapter 2
NI-488.2 Multi-Device Calls

Chapter 3
Supplemental Calls for Multithreaded Applications

Appendix A
Multiline Interface Messages

Appendix B
Status Word Conditions

Appendix C
Error Codes and Solutions

Appendix D
Technical Support Resources

Glossary

Index

Tables

About This Manual

This manual describes the NI-488.2 traditional and multi-device calls of the NI-488.2 software. You can use the NI-488.2 software for Windows with Windows 95, Windows 98, Windows NT version 4.0, or Windows 2000. This manual assumes that you are already familiar with Windows.

Using the NI-488.2 Documentation

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

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

To view these documents online, insert your *NI-488.2 for Windows* CD. When the **NI-488.2 Software for Windows** screen appears, select the **View Documentation** option. The View Documentation Wizard helps you find the documentation that you want to view. You can also view these documents at `http://www.natinst.com/manuals/`.

Accessing the NI-488.2 Online Help

The *NI-488.2 for Windows Online Help* addresses questions you might have about NI-488.2, includes troubleshooting information, and describes the NI-488.2 API. You can access the NI-488.2 online help as follows:

1. Select **Start»Programs»National Instruments NI-488.2»Explore GPIB**.
2. Expand the **Devices and Interfaces** directory by clicking on the + next to the folder.
3. Right-click on your GPIB interface and select **NI-488.2 Help** from the drop-down menu that appears.

Conventions Used in This Manual

This manual uses the following conventions:

»	The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows tabs, or LEDs.
IEEE 488.1 and IEEE 488.2	*IEEE 488* and *IEEE 488.2* refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that are to be literally 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, constants, variables, filenames, and extensions, and for statements and comments taken from program code.

Related Documentation

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

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*
- Microsoft Platform Software Development Kit (SDK)

1

NI-488.2 Traditional Calls

This chapter lists the traditional calls of the NI-488.2 API and describes the purpose, format, input and output parameters, and possible errors for each function.

For general programming information, refer to the *NI-488.2 for Windows Online Help*, available through Measurement & Automation Explorer. This help file describes how to develop and debug your program. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Table 1-1 describes the sections of each function description in this chapter.

Table 1-1. Sections of Function Descriptions

Section	Description
Function names	The functions in this chapter are listed alphabetically. Each function is designated as board-level, device-level, or both.
Purpose	A brief statement of the purpose of the function.
Format	Describes the format of the function in the following languages—Microsoft Visual C/C++ (version 2.0 or later), Borland C/C++ (version 4.0 or later), and Microsoft Visual Basic (version 4.0 or later).
Input and Output	The input and output parameters for the function. *Function Return* describes the return value of the function.
Description	Describes the purpose and the effect of the function.
Examples	Some function descriptions include sample code showing how to use the function. For more detailed and complete examples, refer to the example programs that are installed with the NI-488.2 software.
Possible Errors	A list of errors that could occur when you invoke the function.

List of Traditional Calls

The following tables list the NI-488.2 traditional calls alphabetically and include a brief statement of the purpose of each function.

Device-Level Calls

Table 1-2 lists the device-level traditional calls and includes a brief statement of the purpose of each function.

Table 1-2. Traditional Calls: Device-Level

Function	Purpose
ibask	Return information about software configuration parameters.
ibbna	Change the access interface of a device.
ibclr	Clear a specific device.
ibconfig	Change the software configuration parameters.
ibdev	Open and initialize a device descriptor.
ibeos	Configure the end-of-string (EOS) termination mode or character.
ibeot	Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations.
ibln	Check for the presence of a device on the bus.
ibloc	Go to Local.
ibnotify	Notify user of one or more GPIB events by invoking the user callback.
ibonl	Place the device online or offline.
ibpad	Change the primary address.
ibpct	Pass control to another GPIB device with Controller capability.
ibppc	Parallel poll configure.
ibrd	Read data from a device into a user buffer.
ibrda	Read data asynchronously from a device into a user buffer.
ibrdf	Read data from a device into a file.

Table 1-2. Traditional Calls: Device-Level (Continued)

Function	Purpose
ibrpp	Conduct a parallel poll.
ibrsp	Conduct a serial poll.
ibsad	Change or disable the secondary address.
ibstop	Abort asynchronous I/O operation.
ibtmo	Change or disable the I/O timeout period.
ibtrg	Trigger selected device.
ibwait	Wait for GPIB events.
ibwrt	Write data to a device from a user buffer.
ibwrta	Write data asynchronously to a device from a user buffer.
ibwrtf	Write data to a device from a file.

Board-Level Calls

Table 1-2 lists the board-level traditional calls and includes a brief statement of the purpose of each function.

Table 1-3. Traditional Calls: Board-Level

Function	Purpose
ibask	Return information about software configuration parameters.
ibcac	Become Active Controller.
ibcmd	Send GPIB commands.
ibcmda	Send GPIB commands asynchronously.
ibconfig	Change the software configuration parameters.
ibdma	Enable or disable DMA.
ibeos	Configure the end-of-string (EOS) termination mode or character.
ibeot	Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations.
ibfind	Open and initialize an interface.

Table 1-3. Traditional Calls: Board-Level (Continued)

Function	Purpose
ibgts	Go from Active Controller to Standby.
ibist	Set or clear the interface individual status bit for parallel polls.
iblines	Return the status of the eight GPIB control lines.
ibln	Check for the presence of a device on the bus.
ibloc	Go to Local.
ibnotify	Notify user of one or more GPIB events by invoking the user callback.
ibonl	Place the interface online or offline.
ibpad	Change the primary address.
ibppc	Parallel poll configure.
ibrd	Read data from a device into a user buffer.
ibrda	Read data asynchronously from a device into a user buffer.
ibrdf	Read data from a device into a file.
ibrpp	Conduct a parallel poll.
ibrsc	Request or release system control.
ibrsv	Request service and change the serial poll status byte.
ibsad	Change or disable the secondary address.
ibsic	Assert interface clear.
ibsre	Set or clear the Remote Enable (REN) line.
ibstop	Abort asynchronous I/O operation.
ibtmo	Change or disable the I/O timeout period.
ibwait	Wait for GPIB events.
ibwrt	Write data to a device from a user buffer.
ibwrta	Write data asynchronously to a device from a user buffer.
ibwrtf	Write data to a device from a file.

IBASK

Board-Level/Device-Level

Purpose

Return information about software configuration parameters.

Format

C

```
int ibask (int ud, int option, int *value)
```

Visual Basic

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

or

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

Input

`ud`	Interface or device unit descriptor
`option`	Selects the configuration item whose value is being requested

Output

`value`	Current value of the selected configuration item
Function Return	The value of `ibsta`

Description

`ibask` returns the current value of various configuration parameters for the specified interface or device. The current value of the selected configuration item is returned in the integer `value`. Tables 1-4 and 1-5 list the valid configuration parameter options for `ibask`.

Possible Errors

EARG	`option` is not a valid configuration parameter. Refer to the `ibask` options listed in Tables 1-4 and 1-5.
ECAP	`option` is not supported by the driver or the interface is not configured correctly.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
EOIP	Asynchronous I/O is in progress.

Table 1-4 lists the options you can use with `ibask` when `ud` is an interface descriptor or an interface index.

Table 1-4. `ibask` Board Configuration Parameter Options

Option (Constant)	Returned Information	
	Value	**Description**
`IbaAUTOPOLL`	zero	Automatic serial polling is disabled.
	non-zero	Automatic serial polling is enabled.
	For more information about serial polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
`IbaCICPROT`	zero	The CIC protocol is disabled.
	non-zero	The CIC protocol is enabled.
	For more information about device-level calls and bus management, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
`IbaDMA`	zero	The interface does not use DMA for GPIB transfers.
	non-zero	The interface uses DMA for GPIB transfers.
	Refer to `ibdma`.	
`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.
`IbaEOSchar`	The current EOS character of the interface. Refer to `ibeos`.	
`IbaEOScmp`	zero	A 7-bit compare is used for all EOS comparisons.
	non-zero	An 8-bit compare is used for all EOS comparisons.
	Refer to `ibeos`.	

Table 1-4. `ibask` Board Configuration Parameter Options (Continued)

Option (Constant)	Returned Information	
	Value	**Description**
`IbaEOSrd`	zero	The EOS character is ignored during read operations.
	non-zero	Read operation is terminated by the EOS character.
	Refer to `ibeos`.	
`IbaEOSwrt`	zero	The EOI line is not asserted when the EOS character is sent during a write operation.
	non-zero	The EOI line is asserted when the EOS character is sent during a write operation.
	Refer to `ibeos`.	
`IbaEOT`	zero	The GPIB EOI line is not asserted at the end of a write operation.
	non-zero	EOI is asserted at the end of a write.
	Refer to `ibeot`.	
`IbaHSCableLength`	zero	High-speed (HS488) data transfer is disabled.
	1 to 15	High-speed (HS488) data transfer is enabled. The number returned represents the number of meters of GPIB cable in your system.
	Refer to the NI-488.2 online help for information about high-speed (HS488) data transfer. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
`IbaIst`	The individual status (`ist`) bit of the interface.	
`IbaPAD`	The current primary address of the interface. Refer to `ibpad`.	
`IbaPP2`	zero	The interface is in PP1 mode—remote parallel poll configuration.
	non-zero	The interface is in PP2 mode—local parallel poll configuration.
	For more information about parallel polls, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	

Table 1-4. `ibask` Board Configuration Parameter Options (Continued)

Option (Constant)	Returned Information	
	Value	**Description**
`IbaPPC`	The current parallel poll configuration information of the interface. Refer to `ibppc`.	
`IbaPPollTime`	zero	The interface uses the standard duration (2 µs) when conducting a parallel poll.
	1 to 17	The interface uses a variable length duration when conducting a parallel poll. The duration values correspond to the `ibtmo` timing values.
`IbaReadAdjust`	zero	Read operations do not have pairs of bytes swapped.
	one	Read operations have each pair of bytes swapped.
`IbaRsv`	The current serial poll status byte of the interface.	
`IbaSAD`	The current secondary address of the interface. Refer to `ibsad`.	
`IbaSC`	zero	The interface is not the GPIB System Controller.
	non-zero	The interface is the System Controller.
	Refer to `ibrsc`.	
`IbaSendLLO`	zero	The GPIB LLO command is not sent when a device is put online—`ibfind` or `ibdev`.
	non-zero	The LLO command is sent.
`IbaSRE`	zero	The interface does not automatically assert the GPIB REN line when it becomes the System Controller.
	non-zero	The interface automatically asserts REN when it becomes the System Controller.
	Refer to `ibrsc` and `ibsre`.	
`IbaTIMING`	The current bus timing of the interface.	
	1	Normal timing (T1 delay of 2 µs).
	2	High speed timing (T1 delay of 500 ns).
	3	Very high speed timing (T1 delay of 350 ns).
`IbaTMO`	The current timeout period of the interface. Refer to `ibtmo`.	

Table 1-4. `ibask` Board Configuration Parameter Options (Continued)

Option (Constant)	Returned Information	
	Value	**Description**
`IbaWriteAdjust`	zero	Write operations do not have pairs of bytes swapped.
	one	Write operations have each pair of bytes swapped.

Table 1-5 lists the options you can use with `ibask` when `ud` is a device descriptor or a device index.

Table 1-5. `ibask` Device Configuration Parameter Options

Option (Constant)	Returned Information	
	Value	**Description**
`IbaBNA`	The index of the GPIB access interface used by the given device descriptor.	
`IbaEOSchar`	The current EOS character of the device. Refer to `ibeos`.	
`IbaEOScmp`	zero	A 7-bit compare is used for all EOS comparisons.
	non-zero	An 8-bit compare is used for all EOS comparisons.
	Refer to `ibeos`.	
`IbaEOSrd`	zero	The EOS character is ignored during read operations.
	non-zero	Read operation will be terminated by the EOS character.
	Refer to `ibeos`.	
`IbaEOSwrt`	zero	The EOI line is not asserted when the EOS character is sent during a write operation.
	non-zero	The EOI line is asserted when the EOS character is sent during a write.
	Refer to `ibeos`.	
`IbaEOT`	zero	The GPIB EOI line is not asserted at the end of a write operation.
	non-zero	EOI is asserted at the end of a write.
	Refer to `ibeot`.	

Table 1-5. `ibask` Device Configuration Parameter Options (Continued)

Option (Constant)	Returned Information	
	Value	**Description**
`IbaPAD`	The current primary address of the device. Refer to `ibpad`.	
`IbaReadAdjust`	zero	Read operations do not have pairs of bytes swapped.
	one	Read operations have each pair of bytes swapped.
`IbaREADDR`	zero	No unnecessary addressing is performed between device-level read and write operations.
	non-zero	Addressing is always performed before a device-level read or write operation.
`IbaSAD`	The current secondary address of the device. Refer to `ibsad`.	
`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 `ibtmo` timing values.	
`IbaTMO`	The current timeout period of the device. Refer to `ibtmo`.	
`IbaUnAddr`	zero	The GPIB commands Untalk (UNT) and Unlisten (UNL) are not sent after each device-level read and write operation.
	non-zero	The UNT and UNL commands are sent after each device-level read and write.
`IbaWriteAdjust`	zero	Write operations do not have pairs of bytes swapped.
	one	Write operations have each pair of bytes swapped.

IBBNA

Device-Level

Purpose

Change the access interface of a device.

Format

C

```
int ibbna (int ud, char *bname)
```

Visual Basic

```
CALL ibbna (ud%, bname$)
```

or

```
status% = ilbna (ud%, bname$)
```

Input

ud	A device unit descriptor
bname	An access interface name such as GPIB0

Output

Function Return	The value of ibsta

Description

ibbna assigns the device described by ud to the access interface described by bname. All subsequent bus activity with device ud occurs through the access interface bname. If the call succeeds, iberr contains the previous access interface index.

Possible Errors

EARG	Either ud does not refer to a device or bname does not refer to a valid interface name.
ECIC	The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The access interface is not installed or configured properly.
EOIP	Asynchronous I/O is in progress.

IBCAC

Board-Level

Purpose

Become Active Controller.

Format

C

```
int ibcac (int ud, int v)
```

Visual Basic

```
CALL ibcac (ud%, v%)
```

or

```
status% = ilcac (ud%, v%)
```

Input

ud	An interface 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 interface CIC, use the ibsic function.

To take control synchronously, the GPIB interface attempts to assert the ATN signal without corrupting transferred data. If this is not possible, the interface 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. Functions that require ATN to be asserted, such as ibcmd, do so automatically.

Possible Errors

EARG	ud is valid but does not refer to an interface.
ECIC	The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibclr (int ud)
```

Visual Basic

```
CALL ibclr (ud%)
```

or

```
status% = ilclr (ud%)
```

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

EARG	ud is a valid descriptor but does not refer to a device.
EBUS	No devices are connected to the GPIB.
ECIC	The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibcmd (int ud, void *cmdbuf, long count)
```

Visual Basic

```
CALL ibcmd (ud%, cmdbuf$)
```

or

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

Input

`ud`	An interface unit descriptor
`cmdbuf`	Buffer of command bytes to send
`count`	Number 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 variable, `ibcntl`. Refer to Appendix A, *Multiline Interface Messages*, for a table of the 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 `ibwrt` to send device-specific instructions.

Possible Errors

EABO	The timeout period expired before all of the command bytes were sent.
EARG	ud is valid but does not refer to an interface.
ECIC	The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibcmda (int ud, void *cmdbuf, long count)
```

Visual Basic

```
CALL ibcmda (ud%, cmdbuf$)
```

or

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

Input

`ud`	An interface unit descriptor
`cmdbuf`	Buffer of command bytes to send
`count`	Number 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 variable, `ibcntl`. Refer to Appendix A, *Multiline Interface Messages*, for a table of the 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 `ibwrt` to send device-specific 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 functions:

`ibnotify`	If the `ibsta` value passed to the `ibnotify` callback contains CMPL, the driver and application are resynchronized.
`ibwait`	If the returned `ibsta` 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

EARG	`ud` is valid but does not refer to an interface.
ECIC	The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
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 parameters.

Format

C

```
ibconfig (int ud, int option, int value)
```

Visual Basic

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

or

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

Input

`ud`	Interface 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 interface or device. `option` can be any of the defined constants in Table 1-5 and `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. Table 1-5 lists the valid options.
ECAP	The driver is not able to make the requested change.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
EOIP	Asynchronous I/O is in progress.

Table 1-6 lists the options you can use with `ibconfig` when `ud` is an interface descriptor or an interface index. Default values are in ***bold italics***.

Table 1-6. `ibconfig` Board Configuration Parameter Options

Option (Constant)	Value	Description
`IbcAUTOPOLL`	zero	Disable automatic serial polling.
	non-zero	Enable automatic serial polling.
	Default determined by the NI-488.2 Configuration utility. For more information about automatic serial polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
`IbcCICPROT`	zero	Disable the CIC protocol.
	non-zero	Enable the CIC protocol.
	Default determined by the NI-488.2 Configuration utility. For more information about the CIC protocol, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
`IbcDMA`	Identical to `ibdma`.	
	Default determined by the NI-488.2 Configuration utility.	
`IbcEndBitIsNormal`	zero	Do not set the END bit of `ibsta` when an EOS match occurs during a read.
	non-zero	Set the END bit of `ibsta` when an EOS match occurs during a read.
`IbcEOSchar`	Any 8-bit value.	This byte becomes the new EOS character.
	Default determined by the NI-488.2 Configuration utility.	
`IbcEOScmp`	zero	Use 7 bits for the EOS character comparison.
	non-zero	Use 8 bits for the EOS character comparison.
	Default determined by the NI-488.2 Configuration utility.	
`IbcEOSrd`	zero	Ignore EOS character during read operations.
	non-zero	Terminate reads when the EOS character is read.
	Default determined by the NI-488.2 Configuration utility.	

Table 1-6. `ibconfig` Board Configuration Parameter Options (Continued)

Option (Constant)	Value	Description
IbcEOSwrt	zero	Do not assert EOI with the EOS character during write operations.
	non-zero	Assert EOI with the EOS character during write operations.
	Default determined by the NI-488.2 Configuration utility.	
IbcEOT	Identical to `ibeot`. Changes the data termination mode for write operations.	
	Default determined by the NI-488.2 Configuration utility.	
IbcHSCableLength	zero	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.
	Default determined by the NI-488.2 Configuration utility. Refer to the NI-488.2 online help for information about high-speed (HS488) data transfer. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
IbcIst	***zero***	Changes the individual status (`ist`) bit of the interface.
	Identical to `ibist`.	
IbcPAD	Identical to `ibpad`. Changes the primary address of the interface.	
	Default determined by the NI-488.2 Configuration utility.	
IbcPP2	***zero***	PP1 mode
	non-zero	PP2 mode
	For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.	
IbcPPC	***zero***	Configures the interface for parallel polls.
	Identical to board-level `ibppc`.	

Table 1-6. `ibconfig` Board Configuration Parameter Options (Continued)

Option (Constant)	Value	Description
IbcPPollTime	***zero***	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 `ibtmo` values.
IbcReadAdjust	***zero***	No byte swapping.
	one	Swap pairs of bytes during a read.
IbcRsv	***zero***	Changes the serial poll status byte of the interface.
	Identical to `ibrsv`.	
IbcSAD	Identical to `ibsad`. Changes the secondary address of the interface.	
	Default determined by the NI-488.2 Configuration utility.	
IbcSC	Identical to `ibrsc`. Request or release system control.	
	Default determined by the NI-488.2 Configuration utility.	
IbcSendLLO	***zero***	Do not send LLO when putting a device online—`ibfind` or `ibdev`.
	non-zero	Send LLO when putting a device online—`ibfind` or `ibdev`.
IbcSRE	***zero***	Assert the Remote Enable (REN) line.
	Identical to `ibsre`.	
IbcTIMING	The T1 delay is the GPIB Source Handshake timing.	
	1	Normal timing (T1 delay of 2 µs).
	2	High-speed timing (T1 delay of 500 ns).
	3	Very high-speed timing (T1 delay of 350 ns).
	Default determined by the NI-488.2 Configuration utility.	
IbcTMO	Identical to `ibtmo`. Changes the timeout period of the interface.	
	Default determined by the NI-488.2 Configuration utility.	

Table 1-6. `ibconfig` Board Configuration Parameter Options (Continued)

Option (Constant)	Value	Description
`IbcWriteAdjust`	***zero***	No byte swapping.
	one	Swap pairs of bytes during a write.

Table 1-7 lists the options you can use with `ibconfig` when `ud` is a device descriptor or a device index. Default values are in ***bold italics***.

Table 1-7. `ibconfig` Device Configuration Parameter Options

Option (Constant)	Value	Description
`IbcEOSchar`	Any 8-bit value.	This byte becomes the new EOS character.
	Default determined by the NI-488.2 Configuration utility.	
`IbcEOScmp`	zero	Use seven bits for the EOS character comparison.
	non-zero	Use eight bits for the EOS character comparison.
	Default determined by the NI-488.2 Configuration utility.	
`IbcEOSrd`	zero	Ignore EOS character during read operations.
	non-zero	Terminate reads when the EOS character is read.
	Default determined by the NI-488.2 Configuration utility.	
`IbcEOSwrt`	zero	Do not send EOI with the EOS character during write operations.
	non-zero	Send EOI with the EOS character during writes.
	Default determined by the NI-488.2 Configuration utility.	
`IbcEOT`	Identical to `ibeot`. Changes the data termination method for writes.	
	Default determined by the NI-488.2 Configuration utility.	
`IbcPAD`	Identical to `ibpad`. Changes the primary address of the device.	
	Default determined by the NI-488.2 Configuration utility.	
`IbcReadAdjust`	***zero***	No byte swapping.
	one	Swap pairs of bytes during a read.

Table 1-7. `ibconfig` Device Configuration Parameter Options (Continued)

Option (Constant)	Value	Description
`IbcREADDR`	zero	No unnecessary readdressing is performed between device-level reads and writes.
	non-zero	Addressing is always performed before a device-level read or write.
	Default determined by the NI-488.2 Configuration utility.	
`IbcSAD`	Identical to `ibsad`. Changes the secondary address of the device.	
	Default determined by the NI-488.2 Configuration utility.	
`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 `ibtmo` values.
	Default is ***11***.	
`IbcTMO`	Identical to `ibtmo`. Changes the device timeout period.	
	Default determined by the NI-488.2 Configuration utility.	
`IbcUnAddr`	***zero***	Do not send Untalk (UNT) and Unlisten (UNL) at the end of device-level reads and writes.
	non-zero	Send UNT and UNL at the end of device-level reads and writes.
`IbcWriteAdjust`	***zero***	No byte swapping.
	one	Swap pairs of bytes during a write.

IBDEV

Device-Level

Purpose

Open and initialize a device descriptor.

Format

C

```
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%)
```

Input

`BdIndx`	Index of the access interface 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 corresponding NI-488.2 calls for `ibbna`, `ibpad`, `ibsad`, `ibtmo`, `ibeot`, and `ibeos`.

If ibdev is unable to get a valid device descriptor, a –1 is returned; the ERR bit is set in ibsta and iberr contains EDVR.

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. Refer to ibpad, ibsad, ibtmo, ibeot, and ibeos for details on setting these parameters.
EDVR	Either no device descriptors are available or BdIndx refers to a GPIB interface that is not installed.
ENEB	The interface is not installed or is not properly configured.

IBDMA

Board-Level

Purpose

Enable or disable DMA.

Format

C

```
int ibdma (int ud, int v)
```

Visual Basic

```
CALL ibdma (ud%, v%)
```

or

```
status% = ildma (ud%, v%)
```

Input

ud	Interface 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 interface, 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

EARG	ud is valid but does not refer to an interface.
ECAP	The interface is not configured to use a DMA channel. Use the NI-488.2 Configuration utility to configure a DMA channel.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBEOS

Board-Level/Device-Level

Purpose

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

Format

C

```
int ibeos (int ud, int v)
```

Visual Basic

```
CALL ibeos (ud%, v%)
```

or

```
status% = ileos (ud%, v%)
```

Input

ud	Interface or device descriptor
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 interface 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.

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.

Table 1-8 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.

Table 1-8. EOS Configurations

Bit	Configuration	Value of v	
		High Byte	Low Byte
A	Terminate read when EOS is detected	00000100	EOS character
B	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 functions)	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.

For more information about the termination of I/O operations, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Examples

```
ibeos (ud, 0x140A);  /* Configure the software to end reads on
                        newline character (hex 0A) for the unit
                        descriptor, ud */
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 Either ud is invalid or the NI-488.2 driver is not installed.

ENEB The interface is not installed or is not properly configured.

EOIP Asynchronous I/O is in progress.

IBEOT

Board-Level/Device-Level

Purpose

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

Format

C

```
int ibeot (int ud, int v)
```

Visual Basic

```
CALL ibeot (ud%, v%)
```

or

```
status% = ileot (ud%, v%)
```

Input

ud	Interface or device descriptor
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 interface 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.

For more information about the termination of I/O operations, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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 an interface or a user-configured device descriptor.

Format

C

```
int ibfind (char *udname)
```

Visual Basic

```
CALL ibfind (udname$, ud%)
```

or

```
ud% = ilfind (udname$)
```

Input

udname	A user-configured device or interface name

Output

Function Return	The interface or device descriptor, or a –1

Description

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

`ibfind` performs the equivalent of an `ibonl 1` to initialize the interface or device descriptor. The unit descriptor returned by `ibfind` remains valid until the interface 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 EDVR.

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.

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

Possible Errors

EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `udname` is not recognized as an interface or device name or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.

IBGTS

Board-Level

Purpose

Go from Active Controller to Standby.

Format

C

```
int ibgts (int ud, int v)
```

Visual Basic

```
CALL ibgts (ud%, v%)
```

or

```
status% = ilgts (ud%, v%)
```

Input

ud	Interface 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 ibeos function 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*.

Possible Errors

EADR	v is non-zero, and either ATN is low or the interface is a Talker or a Listener.
EARG	ud is valid but does not refer to an interface.
ECIC	The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBIST

Board-Level

Purpose

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

Format

C

```
int ibist (int ud, int v)
```

Visual Basic

```
CALL ibist (ud%, v%)
```

or

```
status% = ilist (ud%, v%)
```

Input

ud	Interface descriptor
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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EARG	ud is valid but does not refer to an interface.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int iblines (int ud, short *clines)
```

Visual Basic

```
CALL iblines (ud%, clines%)
```

or

```
status% = illines (ud%, clines%)
```

Input

`ud`	Interface descriptor

Output

`clines`	Returns GPIB control line state information
Function Return	The 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

EARG	ud is valid but does not refer to an interface.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBLN

Board-Level/Device-Level

Purpose

Check for the presence of a device on the bus.

Format

C

```
int ibln (int ud, int pad, int sad, short *listen)
```

Visual Basic

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

or

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

Input

ud	Interface or device descriptor
pad	The primary GPIB address of the device
sad	The secondary GPIB address of the device

Output

listen	Indicates if a device is present or not
Function Return	The value of ibsta

Description

ibln determines whether there is a listening device at the GPIB address designated by the pad and sad parameters. If ud is an interface descriptor, the bus associated with that interface is tested for Listeners. If ud is a device descriptor, ibln uses the access interface associated with that device to test for Listeners. 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	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibloc (int ud)
```

Visual Basic

```
CALL ibloc (ud%)
```

or

```
status% = illoc (ud%)
```

Input

ud	Interface or device descriptor

Output

Function Return	The value of ibsta

Description

Board-Level

ibloc places the interface in local mode if it is not in a lockout state. The interface is in a lockout state if LOK does not appear in the status word ibsta. If the interface 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 ibsre function, all device-level functions 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

EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

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

Visual Basic

Not supported

Input

`ud`	Interface 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

Function Return	The value of `ibsta`

Description

If `mask` is non-zero, `ibnotify` monitors the events specified by `mask`, and when one or more of the events is true, your `Callback` is invoked. The `ibnotify` mask bits are identical to the `ibsta` bits, and are defined in Table 1-9. 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.

Table 1-9. Notify Mask Layout

Mnemonic	Bit Position	Hex Value	Description
TIMO	14	4000	Use the timeout period (see `ibtmo`) 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 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 Controller-In-Charge (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)

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.

Note For device-level usage, notification on RQS cannot be 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 0.

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

Callback Prototype

```
int __stdcall Callback
              (int ud, int ibsta, int iberr, long ibcntl,
               void *RefData)
```

Callback Parameters

ud	Interface or device descriptor
ibsta	Value of ibsta
iberr	Value of iberr
ibcntl	Value of ibcntl
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 GPIB global variables (ibsta, iberr, ibcntl) are undefined. The status variables passed to Callback should be examined, instead of the GPIB 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 ibsta set to ERR, iberr set to EDVR, and ibcntl set to IBNOTIFY_REARM_FAILED, which is defined in decl-32.h.

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 more information about the usage of ibnotify and a detailed example, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

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 Either `ud` is invalid or the NI-488.2 driver is not installed. `ibcntl` contains a system-dependent error code.

ENEB The interface is not installed or is not properly configured.

Possible Error for the Callback

EDVR The `Callback` return failed to rearm the `Callback`.

IBONL

Board-Level/Device-Level

Purpose

Place the device or interface online or offline.

Format

C

```
int ibonl (int ud, int v)
```

Visual Basic

```
CALL ibonl (ud%, v%)
```

or

```
status% = ilonl (ud%, v%)
```

Input

ud	Interface or device descriptor
v	Indicates whether the interface or device is to be taken online or offline

Output

Function Return	The value of ibsta

Description

ibonl resets the interface 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 taken 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 interface or device descriptor (ud) is no longer valid. You must execute an ibdev or ibfind to access the interface or device again.

Possible Errors

EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.

IBPAD

Board-Level/Device-Level

Purpose

Change the primary address.

Format

C

```
int ibpad (int ud, int v)
```

Visual Basic

```
CALL ibpad (ud%, v%)
```

or

```
status% = ilpad (ud%, v%)
```

Input

ud	Interface or device descriptor
v	GPIB primary address

Output

Function Return	The value of ibsta

Description

ibpad sets the primary GPIB address of the interface 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	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBPCT

Device-Level

Purpose

Pass control to another GPIB device with Controller capability.

Format

C

```
int ibpct (int ud)
```

Visual Basic

```
CALL ibpct (ud%)
```

or

```
status% = ilpct (ud%)
```

Input

ud	Device descriptor

Output

Function Return	The value of ibsta

Description

ibpct passes Controller-in-Charge status to the device indicated by ud. The access interface 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.
EBUS	No devices are connected to the GPIB.
ECIC	The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibppc (int ud, int v)
```

Visual Basic

```
CALL ibppc (ud%, v%)
```

or

```
status% = ilppc (ud%, v%)
```

Input

ud	Interface 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 message—Parallel 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 an interface 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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EARG	v does not contain a valid PPE or PPD message.
EBUS	Device-level: No devices are connected to the GPIB.
ECAP	Board-level: The interface is not configured to perform local parallel poll configuration. Refer to ibconfig, option IbcPP2.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibrd (int ud, void *rdbuf, long count)
```

Visual Basic

```
CALL ibrd (ud%, rdbuf$)
```

or

```
status% = ilrd (ud%, rdbuf$, count&)
```

Input

`ud`	Interface 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 `ibsta`

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 variable `ibcntl`.

Board-Level

If `ud` is an interface 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 interface is not CIC, the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

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 interface. Use `ibpad` and `ibsad`.
EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibrda (int ud, int *rdbuf, long count)
```

Visual Basic

```
CALL ibrda (ud%, rdbuf$)
```

or

```
status% = ilrda (ud%, rdbuf$, count&)
```

Input

`ud`	Interface 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 `ibsta`

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 variable `ibcntl`.

Board-Level

If `ud` is an interface 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 interface 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 variable `ibcntl`.

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 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 functions:

`ibnotify`	If the `ibsta` value passed to the `ibnotify` callback contains CMPL, the driver and application are resynchronized.
`ibwait`	If the returned `ibsta` 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 `ibcmd` to address the GPIB. Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access interface. Use `ibpad` and `ibsad`.
EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibrdf (int ud, char *flname)
```

Visual Basic

```
CALL ibrdf (ud%, flname$)
```

or

```
status% = ilrdf (ud%, flname$)
```

Input

`ud`	Interface or device descriptor
`flname`	Name 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 variable `ibcntl`.

Board-Level

If `ud` is an interface 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 interface is not CIC, the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Possible Errors

EABO	END was not received within the timeout period, or `ud` is an interface 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 interface. Use `ibpad` and `ibsad`.
EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
EFSO	`ibrdf` could not access `flname`.
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

C

```
int ibrpp (int ud, char *ppr)
```

Visual Basic

```
CALL ibrpp (ud%, ppr%)
```

or

```
status% = ilrpp (ud%, ppr%)
```

Input

ud	Interface or device descriptor

Output

ppr	Parallel poll response byte
Function Return	The 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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRSC

Board-Level

Purpose

Request or release system control.

Format

C

```
int ibrsc (int ud, int v)
```

Visual Basic

```
CALL ibrsc (ud%, v%)
```

or

```
status% = ilrsc (ud%, v%)
```

Input

ud	Interface 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 interface releases system control, and functions requiring System Controller capability are not allowed. If v is non-zero, functions requiring System Controller capability are subsequently allowed. If no error occurs during the call, iberr contains the previous System Controller state of the interface.

Possible Errors

EARG	ud is a valid descriptor but does not refer to an interface.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRSP

Device-Level

Purpose

Conduct a serial poll.

Format

C

```
int ibrsp (int ud, char *spr)
```

Visual Basic

```
CALL ibrsp (ud%, spr%)
```

or

```
status% = ilrsp (ud%, spr%)
```

Input

`ud`	Device descriptor

Output

`spr`	Serial poll response byte
Function Return	The 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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EABO	The serial poll response could not be read within the serial poll timeout period.
EARG	`ud` is a valid descriptor but does not refer to a device.
EBUS	No devices are connected to the GPIB.
ECIC	The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESTB	Autopolling is enabled and the serial poll queue of the device has overflowed. Call `ibrsp` more often to keep the queue from overflowing.

IBRSV

Board-Level

Purpose

Request service and change the serial poll status byte.

Format

C

```
int ibrsv (int ud, int v)
```

Visual Basic

```
CALL ibrsv (ud%, v%)
```

or

```
status% = ilrsv (ud%, v%)
```

Input

ud	Interface 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

EARG	ud is a valid descriptor but does not refer to an interface.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBSAD

Board-Level/Device-Level

Purpose

Change or disable the secondary address.

Format

C

```
int ibsad (int ud, int v)
```

Visual Basic

```
CALL ibsad (ud%, v%)
```

or

```
status% = ilsad (ud%, v%)
```

Input

ud	Interface or device descriptor
v	GPIB secondary address

Output

Function Return	The value of ibsta

Description

ibsad changes the secondary GPIB address of the given interface 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	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibsic (int ud)
```

Visual Basic

```
CALL ibsic (ud%)
```

or

```
status% = ilsic (ud%)
```

Input

ud	Interface descriptor

Output

Function Return	The value of ibsta

Description

ibsic asserts the GPIB interfaces clear (IFC) line for at least 100 µs if the GPIB interface is System Controller. This initializes the GPIB and makes the interface CIC and Active Controller with ATN asserted.

The IFC signal resets only the GPIB interface functions of bus devices and not the internal device functions. Consult your device documentation to determine how to reset the internal functions of your device.

Possible Errors

EARG	ud is a valid descriptor but does not refer to an interface.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface does not have System Controller capability.

IBSRE

Board-Level

Purpose

Set or clear the Remote Enable (REN) line.

Format

C

```
int ibsre (int ud, int v)
```

Visual Basic

```
CALL ibsre (ud%, v%)
```

or

```
status% = ilsre (ud%, v%)
```

Input

ud	Interface descriptor
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 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.

Possible Errors

EARG	ud is a valid descriptor but does not refer to an interface.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface does not have System Controller capability.

IBSTOP

Board-Level/Device-Level

Purpose

Abort asynchronous I/O operation.

Format

C

```
int ibstop (int ud)
```

Visual Basic

```
CALL ibstop (ud%)
```

or

```
status% = ilstop (ud%)
```

Input

ud	Interface 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 EABO is returned, indicating that the I/O was successfully stopped.

Possible Errors

EABO	Asynchronous I/O was successfully stopped.
EBUS	Device-level: No devices are connected to the GPIB.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.

IBTMO

Board-Level/Device-Level

Purpose

Change or disable the I/O timeout period.

Format

C

```
int ibtmo (int ud, int v)
```

Visual Basic

```
CALL ibtmo (ud%, v%)
```

or

```
status% = iltmo (ud%, v%)
```

Input

ud	Interface or device descriptor
v	Timeout duration code

Output

Function Return	The value of ibsta

Description

ibtmo sets the timeout period of the interface 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. Refer to Table 1-10 for a list of valid timeout values. These timeout values represent the minimum timeout period. The actual period could be longer.

Table 1-10. Timeout Code Values

Constant	Value of v	Minimum Timeout
TNONE	0	disable (no timeout)
T10us	1	10 µs
T30us	2	30 µs
T100us	3	100 µs

Table 1-10. Timeout Code Values (Continued)

Constant	Value of v	Minimum Timeout
T300us	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

Possible Errors

EARG v is invalid.

EDVR Either ud is invalid or the NI-488.2 driver is not installed.

ENEB The interface is not installed or is not properly configured.

EOIP Asynchronous I/O is in progress.

IBTRG

Device-Level

Purpose

Trigger selected device.

Format

C

```
int ibtrg (int ud)
```

Visual Basic

```
CALL ibtrg (ud%)
```

or

```
status% = iltrg (ud%)
```

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

EARG	ud is a valid descriptor but does not refer to a device.
EBUS	No devices are connected to the GPIB.
ECIC	The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either ud is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibwait (int ud, int mask)
```

Visual Basic

```
CALL ibwait (ud%, mask%)
```

or

```
status% = ilwait (ud%, mask%)
```

Input

`ud`	Interface or device descriptor
`mask`	Bit mask of GPIB events to wait for

Output

Function Return	The value of `ibsta`

Description

`ibwait` monitors the events specified by `mask` 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; they are described in Table 1-11. If `ud` is a device descriptor, the only valid wait mask bits are TIMO, END, RQS, and CMPL. If `ud` is an interface descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period using the `ibtmo` function.

Table 1-11. Wait Mask Layout

Mnemonic	Bit Position	Hex Value	Description
TIMO	14	4000	Use the timeout period (see ibtmo) 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 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 Controller-In-Charge (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)

Possible Errors

EARG The bit set in mask is invalid.

EBUS Device-level: No devices are connected to the GPIB.

ECIC Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

EDVR Either ud is invalid or the NI-488.2 driver is not installed.

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 *Stuck SRQ* condition exists. For more information about serial polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

IBWRT

Board-Level/Device-Level

Purpose

Write data to a device from a user buffer.

Format

C

```
int ibwrt (int ud, void *wrtbuf, long count)
```

Visual Basic

```
CALL ibwrt (ud%, wrtbuf$)
```

or

```
status% = ilwrt (ud%, wrtbuf$, count&)
```

Input

`ud`	Interface 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 variable `ibcntl`.

Board-Level

If `ud` is an interface 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 interface is not CIC, the CIC sends Device Clear on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

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 interface. Use `ibpad` and `ibsad`.
EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibwrta (int ud, int *wrtbuf, long count)
```

Visual Basic

```
CALL ibwrta (ud%, wrtbuf$)
```

or

```
status% = ilwrta (ud%, wrtbuf$, count&)
```

Input

`ud`	Interface 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 variable `ibcntl`.

Board-Level

If `ud` is an interface 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 interface 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 variable `ibcntl`.

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 functions:

`ibnotify`	If the `ibsta` value passed to the `ibnotify` callback contains CMPL, the driver and application are resynchronized.
`ibwait`	If the returned `ibsta` 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 NI-488.2 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 interface. Use `ibpad` and `ibsad`.
EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
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

C

```
int ibwrtf (int ud, char *flname)
```

Visual Basic

```
CALL ibwrtf (ud%, flname$)
```

or

```
status% = ilwrtf (ud%, flname$)
```

Input

`ud`	Interface 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 variable `ibcntl`.

Board-Level

If `ud` is an interface 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 interface is not CIC, the CIC sends a Device Clear on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

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 interface. Use `ibpad` and `ibsad`.
EBUS	Device-level: No devices are connected to the GPIB.
ECIC	Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
EFSO	`ibwrtf` could not access `flname`.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

2

NI-488.2 Multi-Device Calls

This chapter lists the NI-488.2 multi-device calls and describes the purpose, format, input and output parameters, and possible errors for each call.

For general programming information, refer to the *NI-488.2 for Windows Online Help*, available through Measurement & Automation Explorer. This help file describes how to develop and debug your program. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Table 2-1 describes the sections of each function description in this chapter.

Table 2-1. Sections of Function Descriptions

Section	Description
Function names	The functions in this chapter are listed alphabetically.
Purpose	A brief statement of the purpose of the function.
Format	Describes the format of the function in the following languages—Microsoft Visual C/C++ (version 2.0 or later), Borland C/C++ (version 4.0 or later), and Microsoft Visual Basic (version 4.0 or later).
Input and Output	The input and output parameters for the function. Most of the NI-488.2 multi-device calls have an input parameter which is either a single address or a list of addresses. The address parameter is a 16-bit integer that has two components—the low byte is a valid primary address (0 to 30), and the high byte is a valid secondary address (`NO_SAD(0)` or 96 to 126). A list of addresses is an array of single addresses. You must mark the end of the list with the constant `NOADDR`. An empty address list is either an array with only the `NOADDR` constant in it, or a `NULL` pointer. The C language interface header file includes the definition of a type (`typedef`) called `Addr4882_t`. Use the `Addr4882_t` type when declaring addresses or address lists.
Description	Describes the purpose and the effect of the function.

Table 2-1. Sections of Function Descriptions (Continued)

Section	Description
Examples	For more detailed and complete examples, refer to the example programs that are installed with the NI-488.2 software.
Possible Errors	A list of errors that could occur when you invoke the function.

List of Multi-Device Calls

Table 2-2 lists the NI-488.2 multi-device calls alphabetically and includes a brief statement of the purpose of each function.

Table 2-2. NI-488.2 API: Multi-Device Calls

Call	Purpose
`AllSpoll`	Serial poll all devices.
`DevClear`	Clear a single device.
`DevClearList`	Clear multiple devices.
`EnableLocal`	Enable operations from the front panel of devices (leave remote programming mode).
`EnableRemote`	Enable remote GPIB programming for devices.
`FindLstn`	Find listening devices on the GPIB.
`FindRQS`	Determine which device is requesting service.
`PassControl`	Pass control to another device with Controller capability.
`PPoll`	Perform a parallel poll on the GPIB.
`PPollConfig`	Configure a device to respond to parallel polls.
`PPollUnconfig`	Unconfigure devices for parallel polls.
`RcvRespMsg`	Read data bytes from a device that is already addressed to talk.
`ReadStatusByte`	Serial poll a single device.
`Receive`	Read data bytes from a device.
`ReceiveSetup`	Address a device to be a Talker and the interface to be a Listener in preparation for `RcvRespMsg`.
`ResetSys`	Reset and initialize IEEE 488.2-compliant devices.

Table 2-2. NI-488.2 API: Multi-Device Calls (Continued)

Call	Purpose
`Send`	Send data bytes to a device.
`SendCmds`	Send GPIB command bytes.
`SendDataBytes`	Send data bytes to devices that are already addressed to listen.
`SendIFC`	Reset the GPIB by sending interface clear.
`SendList`	Send data bytes to multiple GPIB devices.
`SendLLO`	Send the Local Lockout (LLO) message to all devices.
`SendSetup`	Set up devices to receive data in preparation for `SendDataBytes`.
`SetRWLS`	Place devices in Remote With Lockout State.
`TestSRQ`	Determine the current state of the GPIB Service Request (SRQ) line.
`TestSys`	Cause IEEE 488.2-compliant devices to conduct self tests.
`Trigger`	Trigger a device.
`TriggerList`	Trigger multiple devices.
`WaitSRQ`	Wait until a device asserts the GPIB Service Request (SRQ) line.

AllSpoll

Purpose

Serial poll all devices.

Format

C

```
void AllSpoll (int boardID, Addr4882_t *addrlist, short *resultlist)
```

Visual Basic

```
CALL AllSpoll (boardID%, addrlist%(), resultlist%())
```

Input

`boardID`	The interface number
`addrlist`	A list of device addresses that is 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 `ibcntl`.

Possible Errors

EABO	One of the devices timed out instead of responding to the serial poll; `ibcntl` contains the index of the timed-out device.
EARG	An invalid address appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void DevClear (int boardID, Addr4882_t address)
```

Visual Basic

```
CALL DevClear (boardID%, address%)
```

Input

`boardID`	The interface number
`address`	Address 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 is invalid.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

DevClearList

Purpose

Clear multiple devices.

Format

C

```
void DevClearList (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL DevClearList (boardID%, 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`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

EnableLocal

Purpose

Enable operations from the front panel of devices (leave remote programming mode).

Format

C

```
void EnableLocal (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL EnableLocal (boardID%, 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`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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.

EnableRemote

Purpose

Enable remote GPIB programming for devices.

Format

C

```
void EnableRemote (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL EnableRemote (boardID%, 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`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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.

FindLstn

Purpose

Find listening devices on the GPIB.

Format

C

```
void FindLstn (int boardID, Addr4882_t *padlist,
Addr4882_t *resultlist, int limit)
```

Visual Basic

```
CALL FindLstn (boardID%, padlist%(), resultlist%(), 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 secondary addresses 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`. `ibcntl` contains the actual number of addresses stored in `resultlist`.

Possible Errors

EARG	An invalid primary address appears in `padlist`; `ibcntl` is the index of the invalid address in the `padlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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`.

FindRQS

Purpose

Determine which device is requesting service.

Format

C

```
void FindRQS (int boardID, Addr4882_t *addrlist, short *result)
```

Visual Basic

```
CALL FindRQS (boardID%, addrlist%(), result%)
```

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`. `ibcntl` 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 `ibcntl` and ETAB is returned in `iberr`.

Possible Errors

EARG	An invalid address appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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`. `ibcntl` contains the index of `NOADDR` in `addrlist`.

PassControl

Purpose

Pass control to another device with Controller capability.

Format

C

```
void PassControl (int boardID, Addr4882_t address)
```

Visual Basic

```
CALL PassControl (boardID%, 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 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`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void PPoll (int boardID, short *result)
```

Visual Basic

```
CALL PPoll (boardID%, result%)
```

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 `PPollConfig` and `PPollUnconfig`). 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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void PPollConfig (int boardID, Addr4882_t address, int dataline,
int lineSense)
```

Visual Basic

```
CALL PPollConfig (boardID%, 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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

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`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void PPollUnconfig (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL PPollUnconfig (boardID%, 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 NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Possible Errors

EARG	An invalid address appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `interfaceID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void RcvRespMsg (int boardID, void *buffer, long count,
int termination)
```

Visual Basic

```
CALL RcvRespMsg (boardID%, buffer$, termination%)
```

Input

`boardID`	The interface number
`count`	Number of bytes read
`termination`	Description of the data termination mode (`STOPend` or an 8-bit 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 variable, `ibcntl`.

`RcvRespMsg` assumes that the interface is already in its listen-active state and a device is already addressed to be a Talker (see `ReceiveSetup` or `Receive`).

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.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

ReadStatusByte

Purpose

Serial poll a single device.

Format

C

```
void ReadStatusByte (int boardID, Addr4882_t address, short *result)
```

Visual Basic

```
CALL ReadStatusByte (boardID%, address%, result%)
```

Input

`boardID`	The interface number
`address`	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	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void Receive (int boardID, Addr4882_t address, void *buffer,
long count, int termination)
```

Visual Basic

```
CALL Receive (boardID%, address%, buffer$, 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 variable, `ibcntl`.

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`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void ReceiveSetup (int boardID, Addr4882_t address)
```

Visual Basic

```
CALL ReceiveSetup (boardID%, address%)
```

Input

`boardID`	The interface number
`address`	Address 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 `RcvRespMsg`; it eliminates the need to readdress the device to receive every block of data.

Possible Errors

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	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void ResetSys (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL ResetSys (boardID%, 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; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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 interface is not System Controller.

Send

Purpose

Send data bytes to a device.

Format

C

```
void Send (int boardID, Addr4882_t address, void *buffer,
long count, int eotmode)
```

Visual Basic

```
CALL Send (boardID%, 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`, 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 variable, `ibcntl`.

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`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void SendCmds (int boardID, void *buffer, long count)
```

Visual Basic

```
CALL SendCmds (boardID%, buffer$)
```

Input

`boardID`	The interface number
`buffer`	Command bytes to be sent
`count`	Number 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 variable `ibcntl`. Refer to Appendix A, *Multiline Interface Messages*, for a listing of the defined interface messages.

Use command bytes to configure the state of the GPIB, not to send instructions to GPIB devices. Use `Send` or `SendList` 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 `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void SendDataBytes (int boardID, void *buffer, long count,
int eotmode)
```

Visual Basic

```
CALL SendDataBytes (boardID%, 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 variable, ibcntl.

SendDataBytes assumes that the interface is in talk-active state and that devices are already addressed as Listeners on the GPIB (see SendSetup, Send, or SendList).

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`.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void SendIFC (int boardID)
```

Visual Basic

```
CALL SendIFC (boardID%)
```

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 unaddressed and that the interface functions of the devices are in their idle states.

Possible Errors

EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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 `ibrsc`.

SendList

Purpose

Send data bytes to multiple GPIB devices.

Format

C

```
void SendList (int boardID, Addr4882_t *addrlist, void *buffer,
long count, int eotmode)
```

Visual Basic

```
CALL SendList (boardID%, 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 variable, `ibcntl`.

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 (`ibcntl` is the index of the 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.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void SendLLO (int boardID)
```

Visual Basic

```
CALL SendLLO (boardID%)
```

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 `SetRWLS`.

Possible Errors

EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void SendSetup (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL SendSetup (boardID%, 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 SendDataBytes 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; ibcntl is the index of the invalid address in the addrlist array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see SendIFC.
EDVR	Either boardID is invalid or the NI-488.2 driver is not installed.
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

C

```
void SetRWLS (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL SetRWLS (boardID%, addrlist%())
```

Input

`boardID`	The interface number
`addrlist`	A 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; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void TestSRQ (int boardID, short *result)
```

Visual Basic

```
CALL TestSRQ (boardID%, result%)
```

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 `WaitSRQ` to wait until SRQ is asserted.

Possible Errors

EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

TestSys

Purpose

Cause IEEE 488.2-compliant devices to conduct self tests.

Format

C

```
void TestSys (int boardID, Addr4882_t *addrlist, short *resultlist)
```

Visual Basic

```
CALL TestSys (boardID%, addrlist%(), resultlist%())
```

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`), `ibcntl` contains the number of devices that failed. Otherwise, the meaning of `ibcntl` 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; `ibcntl` 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; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void Trigger (int boardID, Addr4882_t address)
```

Visual Basic

```
CALL Trigger (boardID%, address%)
```

Input

`boardID`	The interface number
`address`	Address 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.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

TriggerList

Purpose

Trigger multiple devices.

Format

C

```
void TriggerList (int boardID, Addr4882_t *addrlist)
```

Visual Basic

```
CALL TriggerList (boardID%, 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`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
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

C

```
void WaitSRQ (int boardID, short *result)
```

Visual Basic

```
CALL WaitSRQ (boardID%, result%)
```

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 `ibtmo`). When `WaitSRQ` returns, `result` contains a non-zero if SRQ is asserted. Otherwise, `result` contains a zero. Use `TestSRQ` to get the current state of the GPIB SRQ line. Use `WaitSRQ` to wait until SRQ is asserted.

Possible Errors

EDVR	Either `boardID` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

3

Supplemental Calls for Multithreaded Applications

This chapter lists the supplemental functions designed for multithreaded applications and describes the purpose, format, and input and output parameters for each function.

For general programming information, refer to the *NI-488.2 for Windows Online Help*, available through Measurement & Automation Explorer. This help file describes how to develop and debug your program. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Table 3-1 describes the sections of each function description in this chapter.

Table 3-1. Sections of Function Descriptions

Section	Description
Function names	The functions in this chapter are listed alphabetically.
Purpose	A brief statement of the purpose of the function.
Format	Describes the format of the function in the following languages—Microsoft Visual C/C++ (version 2.0 or later), Borland C/C++ (version 4.0 or later), and Microsoft Visual Basic (version 4.0 or later).
Input and Output	The input and output parameters for the function. *Function Return* describes the return value of the function.
Description	Describes the purpose and the effect of the function.

List of Supplemental Calls

Table 3-2 lists the NI-488.2 supplemental calls alphabetically and includes a brief statement of the purpose of each function.

Table 3-2. Supplemental Calls for Multithreaded Applications

Function	Purpose
`ThreadIbcnt`	Return the value of the thread-specific `ibcnt`.
`ThreadIbcntl`	Return the value of the thread-specific `ibcntl`.
`ThreadIberr`	Return the value of the thread-specific `iberr`.
`ThreadIbsta`	Return the value of the thread-specific `ibsta`.

ThreadIbcnt

Purpose

Return the value of the thread-specific `ibcnt`.

Format

C

```
int ThreadIbcnt ()
```

Visual Basic

```
rc% = ThreadIbcnt ()
```

Input

none	No input parameters

Output

Function Return	Value 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 variables (`ibsta`, `iberr`, `ibcnt`, `ibcntl`) 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 variables using `ThreadIbsta`, `ThreadIberr`, `ThreadIbcnt`, and `ThreadIbcntl` instead of the global NI-488.2 status variables.

ThreadIbcntl

Purpose

Return the value of the thread-specific `ibcntl`.

Format

C

```
long ThreadIbcntl ()
```

Visual Basic

```
rc& = ThreadIbcntl ()
```

Input

none	No input parameters

Output

Function Return	Value of `ibcntl` for the calling thread

Description

`ThreadIbcntl` returns the current value of `ibcntl` for a particular thread of execution. The global NI-488.2 status variables (`ibsta`, `iberr`, `ibcnt`, `ibcntl`) 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 variables using `ThreadIbsta`, `ThreadIberr`, `ThreadIbcnt`, and `ThreadIbcntl` instead of the global NI-488.2 status variables.

ThreadIberr

Purpose

Return the value of the thread-specific `iberr`.

Format

C

```
int ThreadIberr ()
```

Visual Basic

```
rc% = ThreadIberr ()
```

Input

none	No input parameters

Output

Function Return	Value 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 variables (`ibsta`, `iberr`, `ibcnt`, `ibcntl`) 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 variables using `ThreadIbsta`, `ThreadIberr`, `ThreadIbcnt`, and `ThreadIbcntl` instead of the global NI-488.2 status variables.

ThreadIbsta

Purpose

Return the value of the thread-specific `ibsta`.

Format

C

```
int ThreadIbsta ()
```

Visual Basic

```
rc% = ThreadIbsta ()
```

Input

none	No input parameters

Output

Function Return	Value 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 variables (`ibsta`, `iberr`, `ibcnt`, `ibcntl`) 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 variables using `ThreadIbsta`, `ThreadIberr`, `ThreadIbcnt`, and `ThreadIbcntl` instead of the global NI-488.2 status variables.

A

Multiline Interface Messages

This appendix lists the multiline interface messages and describes the mnemonics and messages that correspond to the interface functions.

The multiline interface messages are commands defined by the IEEE 488 standard. The messages are sent and received with ATN asserted. The interface functions include initializing the bus, addressing and unaddressing devices, and setting device modes for local or remote programming. For more information about these messages, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*.

Table A-1. Multiline Interface Messages

Hex	Dec	ASCII	Message
00	0	NUL	—
01	1	SOH	GTL
02	2	STX	—
03	3	ETX	—
04	4	EOT	SDC
05	5	ENQ	PPC
06	6	ACK	—
07	7	BEL	—
08	8	BS	GET
09	9	HT	TCT
0A	10	LF	—
0B	11	VT	—
0C	12	FF	—
0D	13	CR	—
0E	14	SO	—
0F	15	SI	—
10	16	DLE	—
11	17	DC1	LLO
12	18	DC2	—
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

Hex	Dec	ASCII	Message
20	32	SP	MLA0
21	33	!	MLA1
22	34	"	MLA2
23	35	#	MLA3
24	36	$	MLA4
25	37	%	MLA5
26	38	&	MLA6
27	39	'	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
3A	58	:	MLA26
3B	59	;	MLA27
3C	60	<	MLA28
3D	61	=	MLA29
3E	62	>	MLA30
3F	63	?	UNL

Table A-1. Multiline Interface Messages (Continued)

Hex	Dec	ASCII	Message
40	64	@	MTA0
41	65	A	MTA1
42	66	B	MTA2
43	67	C	MTA3
44	68	D	MTA4
45	69	E	MTA5
46	70	F	MTA6
47	71	G	MTA7
48	72	H	MTA8
49	73	I	MTA9
4A	74	J	MTA10
4B	75	K	MTA11
4C	76	L	MTA12
4D	77	M	MTA13
4E	78	N	MTA14
4F	79	O	MTA15
50	80	P	MTA16
51	81	Q	MTA17
52	82	R	MTA18
53	83	S	MTA19
54	84	T	MTA20
55	85	U	MTA21
56	86	V	MTA22
57	87	W	MTA23
58	88	X	MTA24
59	89	Y	MTA25
5A	90	Z	MTA26
5B	91	[	MTA27
5C	92	\	MTA28
5D	93	]	MTA29
5E	94	^	MTA30
5F	95	_	UNT

Hex	Dec	ASCII	Message
60	96	`	MSA0, PPE
61	97	a	MSA1, PPE, CFG1
62	98	b	MSA2, PPE, CFG2
63	99	c	MSA3, PPE, CFG3
64	100	d	MSA4, PPE, CFG4
65	101	e	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	l	MSA12, PPE, CFG12
6D	109	m	MSA13, PPE, CFG13
6E	110	n	MSA14, PPE, CFG14
6F	111	o	MSA15, PPE, CFG15
70	112	p	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	x	MSA24, PPD
79	121	y	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	PPD	Parallel Poll Disable
CFG †	Configure	PPE	Parallel Poll Enable
DCL	Device Clear	PPU	Parallel Poll Unconfigure
GET	Group Execute Trigger	SDC	Selected Device Clear
GTL	Go To Local	SPD	Serial Poll Disable
LLO	Local Lockout	SPE	Serial Poll Enable
MLA	My Listen Address	TCT	Take Control
MSA	My Secondary Address	UNL	Unlisten
MTA	My Talk Address	UNT	Untalk
PPC	Parallel Poll Configure		

† This multiline interface message is a proposed extension to the IEEE 488 specification to support the HS488 protocol.

B

Status Word Conditions

This appendix describes the conditions reported in the status word, `ibsta`.

For information about using `ibsta` in your application, refer to the *NI-488.2 for Windows Online Help*. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

Each bit in `ibsta` can be set for device calls (dev), board calls (brd), or both (dev, brd). Table B-1 shows the status word layout.

Table B-1. Status Word Layout

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

ERR (dev, brd)

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

TIMO (dev, brd)

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

END (dev, brd)

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

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

SRQI (brd)

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

RQS (dev)

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

CMPL (dev, brd)

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

LOK (brd)

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

REM (brd)

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

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

CIC (brd)

CIC indicates whether the GPIB interface is the Controller-In-Charge. CIC is set when the `SendIFC` or `ibsic` 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.

ATN (brd)

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

TACS (brd)

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

LACS (brd)

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

DTAS (brd)

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

DCAS (brd)

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

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

C

Error Codes and Solutions

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

Table C-1 lists the GPIB error codes.

Table C-1. GPIB Error Codes

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

EDVR (0)

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

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

Solutions

Possible solutions for this error are as follows:

- Use `ibdev` to open a device without specifying its symbolic name.
- Use only device or interface names that are configured in the NI-488.2 Configuration utility as parameters to the `ibfind` function.
- Use the NI-488.2 Troubleshooting Wizard to ensure that each interface you want to access is working properly. To start the NI-488.2 Troubleshooting Wizard, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**, select **Measurement & Automation** in the left window frame, then choose **Help» Troubleshooting»NI-488.2 Troubleshooting Wizard**.
- Use the unit descriptor returned from `ibdev` or `ibfind` as the first parameter in subsequent traditional NI-488.2 calls. Examine the variable before the failing function to make sure its value has not been corrupted.
- Refer to the *NI-488.2 for Windows Online Help*. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.

ECIC (1)

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

- Any device-level traditional NI-488.2 calls that affect the GPIB.
- Any board-level traditional NI-488.2 calls that issue GPIB command bytes: `ibcmd`, `ibcmda`, `ibln`, and `ibrpp`.
- `ibcac` and `ibgts`.
- Any NI-488.2 multi-device calls that issue GPIB command bytes: SendCmds, PPoll, Send, and `Receive`.

Solutions

Possible solutions for this error are as follows:

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

ENOL (2)

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

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

Solutions

Possible solutions for this error are as follows:

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

EADR (3)

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

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

Solutions

Possible solutions for this error are as follows:

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

EARG (4)

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

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

Solutions

Possible solutions for this error are as follows:

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

ESAC (5)

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

Solutions

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

EABO (6)

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

Solutions

Possible solutions for this error are as follows:

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

ENEB (7)

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

Solutions

Make sure there is a GPIB interface in your computer that is properly configured both in hardware and software using a valid base I/O address by running the NI-488.2 Troubleshooting Wizard. To run the NI-488.2 Troubleshooting Wizard, select **Start»Programs»National Instruments NI-488.2»Explore GPIB**, select **Measurement & Automation** in the left window frame, then choose **Help»Troubleshooting»NI-488.2 Troubleshooting Wizard**.

EDMA (8)

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

Solutions

Possible solutions for this error are as follows:

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

EOIP (10)

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

Solutions

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

`ibnotify`	If the `ibsta` value passed to the `ibnotify` callback contains CMPL, the driver and application are resynchronized.
`ibwait`	If the returned `ibsta` 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.

ECAP (11)

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

Solutions

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

EFSO (12)

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

Solutions

Possible solutions for this error are as follows:

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

EBUS (14)

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

Solutions

Possible solutions for this error are as follows:

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

ESTB (15)

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

Solutions

Possible solutions for this error are as follows:

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

ESRQ (16)

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

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

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

Solutions

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

ETAB (20)

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

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

Solutions

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

D

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.natinst.com/support`.

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.natinst.com/worldwide`.

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Ontario) 905 785 0085,
Canada (Québec) 514 694 8521, China 0755 3904939,
Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466,
Norway 32 27 73 00, Singapore 2265886, Spain (Madrid) 91 640 0085,
Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200,
United Kingdom 01635 523545

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}

A

acceptor handshake — Listeners use this GPIB interface function to receive data, and all devices use it to receive commands. *See* source handshake and handshake.

access interface — 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 Programmer Interface

ASCII — American Standard Code for Information Interchange.

asynchronous — An action or event that occurs at an unpredictable time with respect to the execution of a program.

automatic serial polling — Autopolling. 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.

B

base I/O address — *See* I/O address.

BIOS — Basic Input/Output System.

board-level function — A rudimentary function that performs a single operation.

C

CFE	Configuration Enable. The GPIB command which precedes CFGn and is used to place devices into their configuration mode.
CFGn	These GPIB commands (CFG1 through CFG15) follow CFE and are used to configure all devices for the number of meters of cable in the system so 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* handshake.
DCL	Device Clear. The GPIB command used to reset the device or internal functions of all devices. *See* SDC.
device-level function	A function that combines several rudimentary board operations into one function so that the user does not have to be concerned with bus management or other GPIB protocol matters.
DIO1 through DIO8	The GPIB lines that are used to transmit command or data bytes from one device to another.
DLL	Dynamic link library.
DMA	Direct memory access. High-speed data transfer between the GPIB interface and memory that is not handled directly by the CPU. Not available on some systems. *See* programmed I/O.
driver	Device driver software installed within the operating system.

E

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

EOS or EOS Byte	A 7- or 8-bit end-of-string character that is sent as the last byte of a data message.
EOT	End of transmission
ESB	The Event Status bit 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. The GPIB command used to trigger a device or internal function of an addressed Listener.
GPIB	General Purpose Interface Bus is the common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1992.
GPIB address	The address of a device on the GPIB, composed of a primary address (MLA and MTA) and perhaps a secondary address (MSA). The GPIB interface has both a GPIB address and an I/O address.
GPIB interface	Refers to the National Instruments family of GPIB interfaces.
GTL	Go To Local. The GPIB command used to place an addressed Listener in local (front panel) control mode.

H

handshake	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, *IEEE Standard Digital Interface for Programmable Instrumentation*.
hex	Hexadecimal; a number represented in base 16. For example, decimal 16 = hex 10.
high-level function	*See* device-level function.
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`.
`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`.
`iberr`	A global variable that contains the specific error code associated with a function call that failed.
`ibsta`	At the end of each function call, this global variable (status word) contains status information.
IEEE	Institute of Electrical and Electronic Engineers.
interface message	A broadcast message sent from the Controller to all devices and used to manage the GPIB.
I/O	Input/output. In this manual, it is the transmission of commands or messages between the system via the GPIB board and other devices on the GPIB.
I/O address	The address of the GPIB interface from the point of view of the CPU, as opposed to the GPIB address of the GPIB interface. Also called port address or interface address.
`ist`	An Individual Status bit of the status byte used in the Parallel Poll Configure function.

L

LAD	Listen Address. *See* MLA.
language interface	Code that enables an application program that uses NI-488.2 calls to access the driver.
Listener	A GPIB device that receives data messages from a Talker.

LLO	Local Lockout. The GPIB command used to tell all devices that they may or should ignore remote (GPIB) data messages or local (front panel) controls, depending on whether the device is in local or remote program mode.
low-level function	A rudimentary Controller-In-Charge or device function that performs a single operation.

M

m	Meters.
MAV	The Message Available bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.
MLA	My Listen Address. The 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. The GPIB command used to address a device to be a Talker. It can be any one of the 31 primary addresses.
multitasking	The concurrent processing of more than one program or task.

N

NDAC	Not Data Accepted. One of the three GPIB handshake lines. *See* handshake.
NRFD	Not Ready For Data. One of the three GPIB handshake lines. *See* handshake.

P

parallel poll	The process of polling all configured devices at once and reading a composite poll response. *See* serial poll.
PIO	*See* programmed I/O.

PPC	Parallel Poll Configure. The GPIB command used to configure an addressed Listener to participate in polls.
PPD	Parallel Poll Disable. The GPIB command used to disable a configured device from participating in polls. There are 16 PPD commands.
PPE	Parallel Poll Enable. 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. The GPIB command used to disable used to disable any device from participating in polls.
programmed I/O	Low-speed data transfer between the GPIB interface and memory 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* DCL.
serial poll	The process of polling and reading the status byte of one device at a time. *See* parallel poll.
service request	*See* SRQ.
source handshake	The GPIB interface function that transmits data and commands. Talkers use this function to send data, and the Controller uses it to send commands. *See* acceptor handshake and handshake.
SPD	Serial Poll Disable. The GPIB command used to cancel an SPE command.
SPE	Serial Poll Enable. The GPIB command used to enable a specific device to be polled. That device must also be addressed to talk. *See* SPD.

SRQ	Service Request. The GPIB line that a device asserts to notify the CIC that the device needs servicing.
status byte	The IEEE 488.2-defined data byte sent by a device when it is serially polled.
status word	*See* `ibsta`.
synchronous	Refers to the relationship between the NI-488.2 driver functions and a process when executing driver functions is predictable; the process is blocked until the driver completes the function.
System Controller	The single designated Controller that can assert control (become CIC of the GPIB) by sending the Interface Clear (IFC) message. Other devices can become CIC only by having control passed to them.

T

TAD	Talk Address. *See* MTA.
Talker	A GPIB device that sends data messages to Listeners.
TCT	Take Control. The GPIB command used to pass control of the bus from the current Controller to an addressed Talker.
timeout	A feature of the NI-488.2 driver that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.
TLC	An integrated circuit that implements most of the GPIB Talker, Listener, and Controller functions in hardware.

U

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

Index

E

F

G

I

L

M

N

O

P

R

S

T

W