NI-488.2™
Function Reference Manual for DOS/Windows

August 1996 Edition

Part Number 370903A-01

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (512) 794-5678

Branch Offices:
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
Denmark 45 76 26 00, Finland 90 527 2321, France 01 48 14 24 24,
Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 5734815
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico 95 800 010 0793, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70,
Switzerland 056 200 51 51, Taiwan 02 377 1200, U.K. 01635 523545

Limited 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 manual 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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

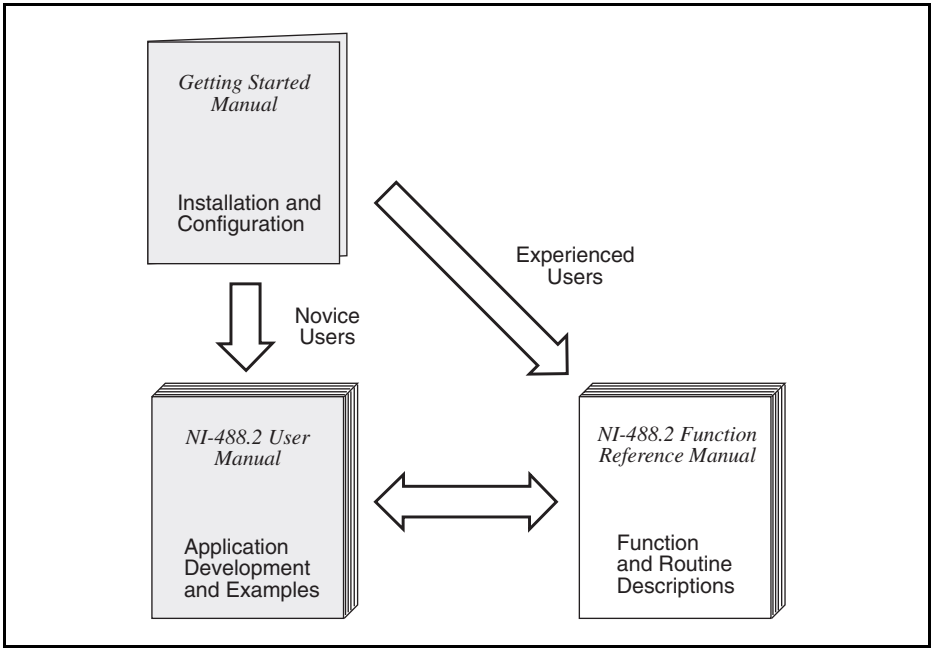
Tables

About This Manual

This manual describes the NI-488 functions and the NI-488.2 routines that comprise the NI-488.2 software package for DOS/Windows. The NI-488.2 software package is meant to be used with Microsoft DOS version 3.0 or higher or with Microsoft Windows version 3.0 or higher. This manual assumes that you are already familiar with the DOS or Windows operating system.

For LabWindows/CVI users, this manual serves as a function reference for the GPIB and GPIB-488.2 libraries, which share the same C syntax as the NI-488.2 routines and NI-488 functions.

How to Use This Manual Set



Use the getting started manual to install and configure your GPIB hardware and NI-488.2 software for DOS or Windows.

Use the *NI-488.2 User Manual for DOS* or *NI-488.2 User Manual for Windows* to learn the basics of GPIB and how to develop an application program. The user manual also contains debugging information and detailed examples.

Use the *NI-488.2 Function Reference Manual for DOS/Windows* for specific NI-488 function and NI-488.2 routine information, such as format, parameters, and possible errors.

If you ordered a kit from National Instruments that includes the GPIB analyzer software, you also received documentation for the GPIB analyzer. You can only use the GPIB analyzer in Windows.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *NI-488 Functions,* includes a listing of the available NI-488 functions and then describes the purpose, format, input and output parameters, and possible errors for each function.
- Chapter 2, *NI-488.2 Routines,* includes a listing of the available NI-488.2 routines and then describes the purpose, format, input and output parameters, and possible errors for each routine.
- Appendix A, *Multiline Interface Messages*, contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions. These multiline interface messages are sent and received with ATN TRUE.
- Appendix B, *Status Word Conditions*, gives a detailed description of the conditions reported in the status word, `ibsta`.
- Appendix C, *Error Codes and Solutions*, lists a description of each error, some conditions under which it might occur, and possible solutions.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
bold italic	Bold italic text denotes a note, caution, or warning.
`monospace`	Text in this font denotes text or characters that are to be literally input 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.
< >	Angle brackets enclose the name of a key on the keyboard–for example, <PageDown>.
IEEE 488 and IEEE 488.2	*IEEE 488* and *IEEE 488.2* are used throughout this manual to refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.
NI-488.2 software	The term *NI-488.2 software* is used throughout this manual to refer to the NI-488.2 software for DOS or Windows unless otherwise noted.

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

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

- *Microsoft MS-DOS User's Guide*
- *Microsoft Windows User's Guide*
- *Microsoft Windows Software Development Kit: Programmer's Reference*
- 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*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix D, *Customer Communication*, at the end of this manual.

Chapter 1
NI-488 Functions

This chapter lists the available NI-488 functions 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 user manual. The user manual explains how to develop and debug your program. It also describes the example programs included with your NI-488.2 software.

Function Names

The functions in this chapter are listed alphabetically. Each function is designated as board level, device level, or both.

Purpose

Each function description includes a brief statement of the purpose of the function.

DOS Format

The DOS format is given for each of the languages supported by the NI-488.2 software:

- Microsoft C (version 5.1 or higher) and Borland C++ (version 2.0 or higher)

  Note: ***The C language interface does not support the Borland C++ huge memory model. Contact National Instruments for the Borland C++ huge memory model language interface.***

- Microsoft Professional BASIC version 7.0 or higher and Microsoft Visual Basic for DOS version 1.0 or higher
- Microsoft QuickBASIC version 4.0 or higher
- BASICA and GWBASIC

Windows Format

The Windows format is given for the following:

- Microsoft C (version 5.1 or higher), LabWindows/CVI for Windows, and Borland C++ (version 2.0 or higher)
- Microsoft Visual Basic version 1.0 or higher
- Direct entry into the Windows Dynamic Link Library `gpib.dll`
  - Direct entry for Microsoft C and Borland C++
  - Direct entry for Microsoft Visual Basic

Input and Output

The input and output parameters for each function are listed. Function Return describes the return value of the function. The return value of the NI-488 functions is usually the value of `ibsta`.

Description

The description section gives details about the purpose and effect of each 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 included with your NI-488.2 software. The example programs are described in Chapter 2 of the NI-488.2 user manual.

Possible Errors

Each function description includes a list of errors that could occur when the function is invoked.

List of NI-488 Functions

The following tables contain alphabetical lists of each NI-488 function along with its purpose. Table 1-1 lists the device-level functions. Table 1-2 lists the board-level functions.

Table 1-1. List of NI-488 Device-Level Functions

Function	Purpose
ibask	Return information about software configuration parameters
ibbna	Change the access board of a device
ibclr	Clear a specific device
ibconfig	Change the software configuration parameters
ibdev	Open and initialize a device
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
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
ibrdi	Read data from a device into a user integer buffer
ibrdia	Read data asynchronously from a device into a user integer buffer
ibrpp	Conduct a parallel poll
ibrsp	Conduct a serial poll
ibsad	Change or disable the secondary address
ibstop	Abort asynchronous I/O operation

(continues)

Table 1-1. List of NI-488 Device-Level Functions (Continued)

Function	Purpose
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
ibwrti	Write data to a device from a user integer buffer
ibwrtia	Write data asynchronously to a device from a user integer buffer

Table 1-2. List of NI-488 Board-Level Functions

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
ibevent	Return the oldest event
ibfind	Open and initialize a GPIB board
ibgts	Go from Active Controller to Standby
ibist	Set or clear the board 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
ibonl	Place the interface board 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
ibrdi	Read data from a device into a user integer buffer
ibrdia	Read data asynchronously from a device into a user integer buffer
ibrpp	Conduct a parallel poll

(continues)

Table 1-2. List of NI-488 Board-Level Functions (Continued)

Function	Purpose
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
ibsrq	Request an SRQ "interrupt routine"
ibstop	Abort asynchronous I/O operation
ibtmo	Change or disable the I/O timeout period
ibtrap	Configure the Applications Monitor
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
ibwrti	Write data to a device from a user integer buffer
ibwrtia	Write data asynchronously to a device from a user integer buffer

IBASK

Board Level
Device Level

IBASK

Purpose

Return information about software configuration parameters.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibask (ud%, option%, value%)
     or
status% = ilask (ud%, option%, value%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibask (ud%, option%, value%)
     or
status% = ilask (ud%, option%, value%)
```

Direct Entry with C

```
DLLibask (int ud, int option, int *value, int _far *ibsta,
          int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibask Lib "gpib.dll"
    (byVal ud%, ByVal option%, value%, ibsta%, iberr%, ibcntl&)
     As Integer
```

IBASK

Board Level
Device Level

IBASK
(Continued)

Input

`ud`	Board or device unit descriptor
`option`	Selects the configuration item whose value is being returned

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 board or device. The current value of the selected configuration item is returned in the integer specified by `value`. Table 1-3 and Table 1-4 list the valid configuration parameter options for `ibask`.

Possible Errors

EARG	`option` is not a valid configuration parameter. See the `ibask` options listed in Table 1-3 and Table 1-4.
ECAP	`option` is not supported by the driver in its current configuration.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.

IBASK

Board Level
Device Level

IBASK
(Continued)

Table 1-3 lists the options you can use with `ibask` when `ud` is a board descriptor or a board index. The following is an alphabetical list of the option constants included in Table 1-3.

Constants	Values	Constants	Values
• IbaAUTOPOLL	0x0007	• IbaPP2	0x0010
• IbaCICPROT	0x0008	• IbaPPC	0x0005
• IbaDMA	0x0012	• IbaPPollTime	0x0019
• IbaEndBitIsNormal	0x001A	• IbaReadAdjust	0x0013
• IbaEOSchar	0x000F	• IbaRsv	0x0021
• IbaEOScmp	0x000E	• IbaSAD	0x0002
• IbaEOSrd	0x000C	• IbaSC	0x000A
• IbaEOSwrt	0x000D	• IbaSendLLO	0x0017
• IbaEOT	0x0004	• IbaSpollBit	0x0016
• IbaEventQueue	0x0015	• IbaSRE	0x000B
• IbaHSCableLength	0x001F	• IbaTIMING	0x0011
• IbaIRQ	0x0009	• IbaTMO	0x0003
• IbaIst	0x0020	• IbaWriteAdjust	0x0014
• IbaPAD	0x0001		

IBASK

Board Level
Device Level

IBASK
(Continued)

Table 1-3. ibask Board Configuration Parameter Options

Options (Constants)	Options (Values)	Returned Information
IbaPAD	0x0001	The current primary address of the board. See `ibpad`.
IbaSAD	0x0002	The current secondary address of the board. See `ibsad`.
IbaTMO	0x0003	The current I/O timeout of the board. See `ibtmo`.
IbaEOT	0x0004	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. See `ibeot`.
IbaPPC	0x0005	The current parallel poll configuration information of the board. See `ibppc`.
IbaAUTOPOLL	0x0007	zero = Automatic serial polling is disabled. non-zero = Automatic serial polling is enabled. Refer to the NI-488.2 user manual for more information about automatic serial polling.
IbaCICPROT	0x0008	zero = The CIC protocol is disabled. non-zero = The CIC protocol is enabled. Refer to the NI-488.2 user manual for more information about device-level calls and bus management.
IbaIRQ	0x0009	zero = Interrupts are not enabled. non-zero = Interrupts are enabled.
IbaSC	0x000A	zero = The board is not the GPIB System Controller. non-zero = The board is the System Controller. See `ibrsc`.

(continues)

IBASK

Board Level
Device Level

IBASK (Continued)

Table 1-3. ibask Board Configuration Parameter Options (Continued)

Options (Constants)	Options (Values)	Returned Information
IbaSRE	0x000B	zero = The board does not automatically assert the GPIB REN line when it becomes the System Controller. non-zero = The board automatically asserts REN when it becomes the System Controller. See `ibrsc` and `ibsre`.
IbaEOSrd	0x000C	zero = The EOS character is ignored during read operations. non-zero = Read operation is terminated by the EOS character. See `ibeos`.
IbaEOSwrt	0x000D	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. See `ibeos`.
IbaEOScmp	0x000E	zero = A 7-bit compare is used for all EOS comparisons. non-zero = An 8-bit compare is used for all EOS comparisons. See `ibeos`.
IbaEOSchar	0x000F	The current EOS character of the board. See `ibeos`.
IbaPP2	0x0010	zero = The board is in PP1 mode–remote parallel poll configuration. non-zero = The board is in PP2 mode–local parallel poll configuration. Refer to the NI-488.2 user manual for more information about parallel polls.

(continues)

IBASK

Board Level
Device Level

IBASK
(Continued)

Table 1-3. ibask Board Configuration Parameter Options (Continued)

Options (Constants)	Options (Values)	Returned Information
IbaTIMING	0x0011	The current bus timing of the board. 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.)
IbaDMA	0x0012	zero = The board will not use DMA for GPIB transfers. non-zero = The board will use DMA for GPIB transfers. See `ibdma`.
IbaReadAdjust	0x0013	0 = Read operations do not have pairs of bytes swapped. 1 = Read operations have each pair of bytes swapped.
IbaWriteAdjust	0x0014	0 = Write operations do not have pairs of bytes swapped. 1 = Write operations have each pair of bytes swapped.
IbaEventQueue	0x0015	zero = The event queue is disabled. non-zero = The event queue is enabled. See `ibevent`.
IbaSpollBit	0x0016	zero = The SPOLL bit of `ibsta` is disabled. non-zero = The SPOLL bit of `ibsta` is enabled. See the NI-488.2 user manual for information about Talker/Listener applications.
IbaSendLLO	0x0017	zero = The GPIB LLO command is not sent when a device is put online-`ibfind` or `ibdev`. non-zero = The LLO command is sent.

(continues)

Table 1-3. ibask Board Configuration Parameter Options (Continued)

Options (Constants)	Options (Values)	Returned Information
IbaPPollTime	0x0019	0 = The board uses the standard duration (2 µs) when conducting a parallel poll. 1 to 17 = The board uses a variable length duration when conducting a parallel poll. The duration values correspond to the `ibtmo` timing values.
IbaEndBitIsNormal	0x001A	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.
IbaHSCableLength	0x001F	0 = High-speed data transfer (HS488) is disabled. 1 to 15 = High-speed data transfer (HS488) is enabled. The number returned represents the number of meters of GPIB cable in your system. See the NI-488.2 user manual for information about high-speed data transfers (HS488).
IbaIst	0x0020	The individual status (`ist`) bit of the board.
IbaRsv	0x0021	The current serial poll status byte of the board.

IBASK

Board Level
Device Level

IBASK
(Continued)

Table 1-4 lists the options you can use with `ibask` when `ud` is a device descriptor or a device index. The following is an alphabetical list of the option constants included in Table 1-4.

Constants	Values	Constants	Values
• IbaBNA	0x0200	• IbaReadAdjust	0x0013
• IbaEndBitIsNormal	0x001A	• IbaREADDR	0x0006
• IbaEOSchar	0x000F	• IbaSAD	0x0002
• IbaEOScmp	0x000E	• IbaSPollTime	0x0018
• IbaEOSrd	0x000C	• IbaTMO	0x0003
• IbaEOSwrt	0x000D	• IbaUnAddr	0x001B
• IbaEOT	0x0004	• IbaWriteAdjust	0x0014
• IbaPAD	0x0001		

Table 1-4. ibask Device Configuration Parameter Options

Options (Constants)	Options (Values)	Returned Information
IbaPAD	0x0001	The current primary address of the device. See `ibpad`.
IbaSAD	0x0002	The current secondary address of the device. See `ibsad`.
IbaTMO	0x0003	The current I/O timeout of the device. See `ibtmo`.
IbaEOT	0x0004	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 operation. See `ibeot`.
IbaREADDR	0x0006	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.
IbaEOSrd	0x000C	zero = The EOS character is ignored during read operations. non-zero = Read operation is terminated by the EOS character. See `ibeos`.
IbaEOSwrt	0x000D	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. See `ibeos`.
IbaEOScmp	0x000E	zero = A 7-bit compare is used for all EOS comparisons. non-zero = An 8-bit compare is used for all EOS comparisons. See `ibeos`.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
IbaEOSchar	0x000F	The current EOS character of the device. See `ibeos`.
IbaReadAdjust	0x0013	0 = Read operations do not have pairs of bytes swapped. 1 = Read operations have each pair of bytes swapped.
IbaWriteAdjust	0x0014	0 = Write operations do not have pairs of bytes swapped. 1 = Write operations have each pair of bytes swapped.
IbaSPollTime	0x0018	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.
IbaEndBitIsNormal	0x001A	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.
IbaUnAddr	0x001B	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 operation.
IbaBNA	0x0200	The index of the GPIB access board used by the given device descriptor.

IBBNA Device Level IBBNA

Purpose

Change the access board of a device.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibbna (ud%, bname$)     or    status% = ilbna (ud%, bname$)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibbna (ud%, bname$)     or    status% = ilbna (ud%, bname$)
```

Direct Entry with C

```
DLLibbna (int ud, char _far *bname, int _far *ibsta,
          int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibbna Lib "gpib.dll"
    (ByVal ud%, ByVal bname$, ibsta%, iberr%, ibcntl&) As
    Integer
```

IBBNA

Device Level

IBBNA
(Continued)

Input

`ud`	A device unit descriptor
`bname`	An access board name, for example, `gpib0`

Output

Function Return	The value of `ibsta`

Description

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

Possible Errors

EARG	Either `ud` does not refer to a device or `bname` does not refer to a valid board name.
ECIC	The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The access board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBCAC **Board Level** IBCAC

Purpose

Become Active Controller.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibcac (ud%, v%)     or     status% = ilcac (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibcac (ud%, v%)     or     status% = ilcac (ud%, v%)
```

Direct Entry with C

```
DLLibcac (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibcac Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As Integer
```

IBCAC

Board Level

IBCAC
(Continued)

Input

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

Output

Function Return	The value of `ibsta`

Description

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

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

To take control asynchronously, the GPIB board 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 board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBCLR Device Level IBCLR

Purpose

Clear a specific device.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibclr (ud%)     or     status% = ilclr (ud%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibclr (ud%)     or     status% = ilclr (ud%)
```

Direct Entry with C

```
DLLibclr (int ud, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibclr Lib "gpib.dll"
     (ByVal ud%, ibsta%, iberr%, ibcntl&) As Integer
```

IBCLR

Device Level

IBCLR
(Continued)

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	There are no devices connected to the GPIB.
ECIC	The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 7 of the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBCMD Board Level IBCMD

Purpose

Send GPIB commands.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibcmd (ud%, cmdbuf$)
     or
status% = ilcmd (ud%, cmdbuf$, cnt&)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibcmd (ud%, cmdbuf$)
     or
status% = ilcmd (ud%, cmdbuf$, cnt&)
```

Direct Entry with C

```
DLLibcmd(int ud, void _far *cmdbuf, long cnt, int _far *ibsta,
         int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibcmd Lib "gpib.dll"
     (ByVal ud%, ByVal cmdbuf$, ByVal cnt&, ibsta%, iberr%,
      ibcntl&) As Integer
```

IBCMD

Board Level

IBCMD
(Continued)

Input

`ud`	A board unit descriptor
`cmdbuf`	Buffer of command bytes to send
`cnt`	Number of command bytes to send

Output

Function Return	The value of `ibsta`

Description

`ibcmd` sends `cnt` 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 board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board 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** IBCMDA

Purpose

Send GPIB commands asynchronously.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibcmda (ud%, cmdbuf$)
     or
status% = ilcmda (ud%, cmdbuf$, cnt&)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibcmda (ud%, cmdbuf$)
     or
status% = ilcmda (ud%, cmdbuf$, cnt&)
```

Direct Entry with C

```
DLLibcmda (int ud, void _far *cmdbuf, long cnt,
           int _far *ibsta, int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibcmda Lib "gpib.dll"
     (ByVal ud%, ByVal cmdbuf$, ByVal cnt&, ibsta%, iberr%,
      ibcntl&) As Integer
```

IBCMDA **Board Level** IBCMDA (Continued)

Input

`ud`	A board unit descriptor
`cmdbuf`	Buffer of command bytes to send
`cnt`	Number of command bytes to send

Output

Function Return	The value of `ibsta`

Description

`ibcmda` sends `cnt` 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 has begun, further GPIB 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 three functions:

- `ibwait` If the returned `ibsta` mask has the CMPL bit set, 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.

IBCMDA **Board Level** **IBCMDA**
(Continued)

Possible Errors

EARG	`ud` is valid but does not refer to an interface board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board 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

IBCONFIG

Purpose

Change the software configuration parameters.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibconfig (ud%, option%, value%)
     or
status% = ilconfig (ud%, option%, value%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibconfig (ud%, option%, value%)
     or
status% = ilconfig (ud%, option%, value%)
```

Direct Entry with C

```
DLLibconfig (int ud, int option, int value,
             int _far *ibsta, int _far *iberr,
             long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibconfig Lib "gpib.dll"
     (ByVal ud%, ByVal option%, ByVal value%, ibsta%, iberr%,
      ibcntl&) As Integer
```

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

Input

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

Output

Function Return	The value of `ibsta`

Description

`ibconfig` changes the configuration item to the specified value for the selected board or device. `option` may 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 return in `iberr`.

Possible Errors

EARG	Either `option` or `value` is not valid. See Table 1-5.
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.

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

Table 1-5 lists the options you can use with `ibconfig` when `ud` is a board descriptor or a board index. If the table does not list the default value for a particular option, the default value is determined by either `ibconf` in DOS or the GPIB software configuration utility in Windows.

The following is an alphabetical list of the `option` constants included in Table 1-5.

Constants	Values	Constants	Values
• IbcAUTOPOLL	0x0007	• IbcPP2	0x0010
• IbcCICPROT	0x0008	• IbcPPC	0x0005
• IbcDMA	0x0012	• IbcPPollTime	0x0019
• IbcEndBitIsNormal	0x001A	• IbcReadAdjust	0x0013
• IbcEOSchar	0x000F	• IbcRsv	0x0021
• IbcEOScmp	0x000E	• IbcSAD	0x0002
• IbcEOSrd	0x000C	• IbcSC	0x000A
• IbcEOSwrt	0x000D	• IbcSendLLO	0x0017
• IbcEOT	0x0004	• IbcSpollBit	0x0016
• IbcEventQueue	0x0015	• IbcSRE	0x000B
• IbcHSCableLength	0x001F	• IbcTIMING	0x0011
• IbcIRQ	0x0009	• IbcTMO	0x0003
• IbcIst	0x0020	• IbcWriteAdjust	0x0014
• IbcPAD	0x0001		

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

Table 1-5. ibconfig Board Configuration Parameter Options

Options (Constants)	Options (Values)	Legal Values
IbcPAD	0x0001	Changes the primary address of the board. Identical to `ibpad`.
IbcSAD	0x0002	Changes the secondary address of the board. Identical to `ibsad`.
IbcTMO	0x0003	Changes the I/O timeout limit of the board. Identical to `ibtmo`.
IbcEOT	0x0004	Changes the data termination mode for write operations. Identical to `ibeot`.
IbcPPC	0x0005	Configures the board for parallel polls. Identical to board-level `ibppc`. Default: zero.
IbcAUTOPOLL	0x0007	zero = Disable automatic serial polling. non-zero = Enable automatic serial polling. Refer to the NI-488.2 user manual for more information about automatic serial polling.
IbcCICPROT	0x0008	zero = Disable the CIC protocol. non-zero = Enable the CIC protocol. Refer to the NI-488.2 user manual for more information about the CIC protocol.
IbcIRQ	0x0009	zero = Do not use interrupts. non-zero = Use interrupts-use the hardware interrupt level configured through `ibconf` in DOS or the GPIB software configuration utility in Windows.
IbcSC	0x000A	Request or release system control. Identical to `ibrsc`.

(continues)

Table 1-5. ibconfig Board Configuration Parameter Options (Continued)

Options (Constants)	Options (Values)	Legal Values
IbcSRE	0x000B	Assert the Remote Enable (REN) line. Identical to `ibsre`. Default: zero.
IbcEOSrd	0x000C	zero = Ignore EOS character during read operations. non-zero = Terminate reads when the EOS character is read match occurs.
IbcEOSwrt	0x000D	zero = Do not assert EOI with the EOS character during write operations. non-zero = Assert EOI with the EOS character during writes operations.
IbcEOScmp	0x000E	zero = Use 7 bits for the EOS character comparison. non-zero = Use 8 bits for the EOS character comparison.
IbcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character.
IbcPP2	0x0010	zero = PP1 mode-remote parallel poll configuration. non-zero = PP2 mode-local parallel poll configuration. Default: zero. Refer to the NI-488.2 user manual for more information about parallel polling.
IbcTIMING	0x0011	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). The T1 delay is the GPIB source handshake timing.

(continues)

Table 1-5. ibconfig Board Configuration Parameter Options (Continued)

Options (Constants)	Options (Values)	Legal Values
IbcDMA	0x0012	Identical to `ibdma`.
IbcReadAdjust	0x0013	0 = No byte swapping. 1 = Swap pairs of bytes during a read. Default: zero.
IbcWriteAdjust	0x0014	0 = No byte swapping. 1 = Swap pairs of bytes during a write. Default: zero.
IbcEventQueue	0x0015	zero = The event queue is disabled. non-zero = The event queue is enabled. Default: zero. See `ibevent`.
IbcSpollBit	0x0016	zero = The SPOLL bit of `ibsta` is disabled. non-zero = The SPOLL bit of `ibsta` is enabled. Default: zero. Refer to the NI-488.2 user manual for information about Talker/Listener applications.
IbcSendLLO	0x0017	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`. Default: zero.
IbcPPollTime	0x0019	0 = 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. Default: zero.

(continues)

Table 1-5. ibconfig Board Configuration Parameter Options (Continued)

Options (Constants)	Options (Values)	Legal Values
IbcEndBitIsNormal	0x001A	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. Default: non-zero.
IbcHSCableLength	0x001F	0 = High-speed data transfer (HS488) 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 data transfer (HS488) mode. See the NI-488.2 user manual for information about high-speed data transfers (HS488).
IbcIst	0x0020	Changes the individual status (`ist`) bit of the board. Identical to `ibist`.
IbcRsv	0x0021	Changes the serial poll status byte of the board. Identical to `ibrsv`. Default: zero.

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

Table 1-6 lists the options you can use with `ibconfig` when `ud` is a device descriptor or a device index. If the table does not list the default value for a particular option, the default value is determined by either `ibconf` in DOS or the GPIB software configuration utility in Windows.

The following is an alphabetical list of the option constants included in Table 1-6.

Constants	Values	Constants	Values
• IbcEndBitIsNormal	0x001A	• IbcREADDR	0x0006
• IbcEOSchar	0x000F	• IbcReadAdjust	0x0013
• IbcEOScmp	0x000E	• IbcSAD	0x0002
• IbcEOSrd	0x000C	• IbcSPollTime	0x0018
• IbcEOSwrt	0x000D	• IbcTMO	0x0003
• IbcEOT	0x0004	• IbcWriteAdjust	0x0014
• IbcPAD	0x0001	• IbcUnAddr	0x001B

Table 1-6. ibconfig Device Configuration Parameter Options

Options (Constants)	Options (Values)	Legal Values
IbcPAD	0x0001	Changes the primary address of the device. Identical to `ibpad`.
IbcSAD	0x0002	Changes the secondary address of the device. Identical to `ibsad`.
IbcTMO	0x0003	Changes the device I/O timeout limit. Identical to `ibtmo`.

(continues)

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

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

Options (Constants)	Options (Values)	Legal Values
IbcEOT	0x0004	Changes the data termination method for writes. Identical to `ibeot`.
IbcREADDR	0x0006	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.
IbcEOSrd	0x000C	non-zero = Terminate reads when the EOS character is read.
IbcEOSwrt	0x000D	zero = Do not send EOI with the EOS character during write operations. non-zero = Send EOI with the EOS character during writes.
IbcEOScmp	0x000E	zero = Use seven bits for the EOS character comparison. non-zero = Use 8 bits for the EOS character comparison.
IbcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character.
IbcReadAdjust	0x0013	0 = No byte swapping. 1 = Swap pairs of bytes during a read. Default: zero.
IbcWriteAdjust	0x0014	0 = No byte swapping. 1 = Swap pairs of bytes during a write. Default: zero.

(continues)

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

Options (Constants)	Options (Values)	Legal Values
IbcSPollTime	0x0018	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: 11.
IbcEndBitIsNormal	0x001A	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. Default: non-zero.
IbcUnAddr	0x001B	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. Default: zero.

IBDEV Device Level IBDEV

Purpose

Open and initialize a device descriptor.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibdev (BdIndx%, pad%, sad%, tmo%, eot%, eos%, ud%)
     or
ud% = ildev (BdIndx%, pad%, sad%, tmo%, eot%, eos%)
```

BASICA

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

Windows 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%)
```

Direct Entry with C

```
DLLibdev (int BdIndx, int pad, int sad, int tmo, int eot, int
          eos, int _far *ibsta, int _far *iberr, long _far
          *ibcntl)
```

IBDEV

Device Level

IBDEV
(Continued)

Direct Entry with Visual Basic

```
Declare Function DLLibdev Lib "gpib.dll"
     (ByVal BdIndx%, ByVal pad%, ByVal sad%, ByVal tmo%,
      ByVal eot%, ByVal eos%, ibsta%, iberr%, ibcntl&)
      As Integer
```

Input

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

Output

Function Return	The device descriptor

Description

`ibdev` acquires a device descriptor to use in subsequent device-level NI-488 functions. 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 functions 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.

IBDEV

Device Level

IBDEV
(Continued)

`ibdev` acquires and initializes a device descriptor from the set of user-configurable devices (for example, `dev1`, `dev2`, and so on through `dev32`). As a result, it is necessary for an application to use `ibdev` only after all calls to `ibfind` for user-configurable devices have been completed. This is the only way to ensure that `ibdev` and `ibfind` do not both return the same device descriptor.

Possible Errors

EARG	`pad`, `sad`, `tmo`, or `eos` is invalid. See the corresponding NI-488 function.
EDVR	Either no device descriptors are available or `BdIndx` refers to a GPIB board that is not installed.
ENEB	The interface board is not installed or is not properly configured.

IBDMA Board Level IBDMA

Purpose

Enable or disable DMA.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibdma (ud%, v%)     or     status% = ildma (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibdma (ud%, v%)     or     status% = ildma (ud%, v%)
```

Direct Entry with C

```
DLLibdma (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibdma Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As Integer
```

IBDMA

Board Level

IBDMA
(Continued)

Input

`ud`	A board descriptor
`v`	Enable or disable the use of DMA

Output

Function Return	The value of `ibsta`

Description

`ibdma` enables or disables DMA transfers for the board described by `ud`. If `v` is zero then DMA is not used for GPIB I/O transfers. If `v` is non-zero, then DMA is used for GPIB I/O transfers.

Possible Errors

EARG	`ud` is valid but does not refer to an interface board.
ECAP	The interface board is not configured to use a DMA channel. To configure a DMA channel, use `ibconf` in DOS or the GPIB software configuration utility in Windows.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBEOS

Board Level
Device Level

IBEOS

Purpose

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

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibeos (ud%, v%)     or     status% = ileos (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibeos (ud%, v%)     or     status% = ileos (ud%, v%)
```

Direct Entry with C

```
DLLibeos  (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibeos Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As Integer
```

IBEOS

Board Level
Device Level

IBEOS
(Continued)

Input

`ud`	A board 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 used by the board or device described by `ud`. The parameter `v` describes the new end-of-string (EOS) configuration to use. If `v` is zero, then the EOS configuration is disabled. Otherwise, the low byte is the EOS character and the upper byte contains flags which define the EOS mode. Table 1-7 describes the different EOS configurations and the corresponding values of `v`. If no error occurs during the call, then the value of the previous EOS setting is returned in `iberr`.

Table 1-7. 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

IBEOS

Board Level
Device Level

IBEOS
(Continued)

Configuration bits A and C determine how to terminate read I/O operations. If bit A is set and bit C is clear, then 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, then 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, then EOI is asserted when the written character matches the low seven bits of the EOS character. If bits B and C are both set, then EOI is asserted when the written character matches all eight bits of the EOS character.

Note: ***Defining an EOS byte does not cause the driver to automatically send that byte at the end of write I/O operations. In your application the EOS byte must be placed at the end of the data strings that it defines.***

For more information on the termination of I/O operations refer to the NI-488.2 user 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 board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBEOT

Board Level
Device Level

IBEOT

Purpose

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

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibeot (ud%, v%)     or     status% = ileot (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibeot (ud%, v%)     or     status% = ileot (ud%, v%)
```

Direct Entry with C

```
DLLibeot (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibeot Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As Integer
```

IBEOT

Board Level
Device Level

IBEOT
(Continued)

Input

`ud`	A board 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, such as `ibwrt`, for the board or device described by `ud`. If `v` is non-zero, then EOI is asserted when the last byte of a GPIB write is sent. If `v` is zero, then nothing occurs when the last byte is sent. If no error occurs during the call, then the previous value of EOT is returned in `iberr`.

For more information on the termination of I/O operations refer to the NI-488.2 user manual.

Possible Errors

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

IBEVENT Board Level IBEVENT

Purpose

Return the oldest recorded event.

DOS Format

C

```
int ibevent (int ud, short *event)
```

QuickBASIC/BASIC

```
CALL ibevent (ud%, event%)
```

or

```
status% = ilevent (ud%, event%)
```

BASICA

```
CALL ibevent (ud%, event%)
```

Windows Format

C

```
int ibevent (int ud, short *event)
```

Visual Basic

```
CALL ibevent (ud%, event%)
```

or

```
status% = ilevent (ud%, event%)
```

Direct Entry with C

```
DLLibevent (int ud, short _far *event, int _far *ibsta,
            int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibevent Lib "gpib.dll"
     (ByVal ud%, event%, ibsta%, iberr%, ibcntl&)As Integer
```

IBEVENT **Board Level** IBEVENT (Continued)

Input

`ud`	A board descriptor

Output

`event`	The returned event
Function Return	The value of `ibsta`

Description

`ibevent` determines which GPIB event (Device Clear, Device Trigger, or Interface Clear) occurred. Usually, you call `ibevent` when the EVENT bit is set in `ibsta`. The variable event can be one of the following values:

0 = No events are in the queue

1 = A Device Trigger message was received

2 = A Device Clear message was received

3 = Interface Clear was received

Note: ***The GPIB board must not be configured as System Controller to detect interface clear. Use*** `ibconfig`***, option*** `IbcSC` ***to disable System Controller capability.***

When this function returns, `ibcntl` contains the number of events that remain in the event queue. To enable the event queue, use the `ibconfig` function, option `IbcEventQueue`.

IBEVENT **Board Level** **IBEVENT (Continued)**

Possible Errors

EARG	`ud` is a valid descriptor but does not refer to a board.
ECAP	The interface board is not configured to use the event queue (See `ibconfig`, option `IbcEventQueue`).
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ETAB	The event queue has overflowed and events have been discarded.

IBFIND

Board Level
Device Level

IBFIND

Purpose

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

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibfind (udname$, ud%)     or     ud% = ilfind (udname$)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibfind (udname$, ud%)     or     ud% = ilfind (udname$)
```

Direct Entry with C

```
DLLibfind (char _far *udname, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibfind Lib "gpib.dll"
     (ByVal udname$, ibsta%, iberr%, ibcntl&) As Integer
```

IBFIND

Board Level
Device Level

IBFIND
(Continued)

Input

`udname`	A user-configured device or board name

Output

Function Return	The board or device descriptor

Description

`ibfind` is used to acquire a descriptor for a board or user-configured device; this board or device descriptor can be used in subsequent NI-488 functions.

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

If `ibfind` is unable to get a valid descriptor, a -1 is returned; the ERR bit is set in `ibsta` and `iberr` contains 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: There are no devices connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `udname` is not recognized as a board or device name or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

IBGTS **Board Level** IBGTS

Purpose

Go from Active Controller to Standby.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibgts (ud%, v%)     or     status% = ilgts (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibgts (ud%, v%)     or     status% = ilgts (ud%, v%)
```

Direct Entry with C

```
DLLibgts (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibgts Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&)As Integer
```

IBGTS **Board Level** IBGTS (Continued)

Input

`ud`	Board descriptor
`v`	Determines whether to perform acceptor handshaking

Output

Function Return	The value of `ibsta`

Description

`ibgts` causes the board `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 board can participate in data handshake as an acceptor without actually reading data. If END is detected, the interface board 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 more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987.

Possible Errors

EADR	`v` is non-zero, and either ATN is low or the interface board is a Talker or a Listener.
EARG	`ud` is valid but does not refer to an interface board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBIST Board Level IBIST

Purpose

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

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibist (ud%, v%)     or     status% = ilist (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibist (ud%, v%)     or     status% = ilist (ud%, v%)
```

Direct Entry with C

```
DLLibist (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibist Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As Integer
```

IBIST

Board Level

IBIST
(Continued)

Input

`ud`	Board descriptor
`v`	Indicates whether to set or clear the `ist` bit

Output

Function Return	The value of `ibsta`

Description

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

For more information on parallel polling, refer to the NI-488.2 user manual.

Possible Errors

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

IBLINES **Board Level** IBLINES

Purpose

Return the status of the eight GPIB control lines.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL iblines (ud%, clines%)
     or
status% = illines (ud%, clines%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL iblines (ud%, clines%)
     or
status% = illines (ud%, clines%)
```

Direct Entry with C

```
DLLiblines (int ud, short _far *clines, int _far *ibsta,
            int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLiblines Lib "gpib.dll"
     (ByVal ud%, clines%, ibsta%, iberr%, ibcntl&) As Integer
```

IBLINES **Board Level** IBLINES (Continued)

Input

`ud`	Board 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 board 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), then 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");
   }
}
```

IBLINES **Board Level** **IBLINES (Continued)**

Possible Errors

EARG	`ud` is valid but does not refer to an interface board.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

IBLN

Board Level
Device Level

IBLN

Purpose

Check for the presence of a device on the bus.

DOS Format

C

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

QuickBASIC/BASIC

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

BASICA

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

Windows 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%)
```

Direct Entry with C

```
DLLibln (int ud, int pad, int sad, short _far *listen,
         int _far *ibsta, int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibln Lib "gpib.dll"
     (ByVal ud%, ByVal pad%, ByVal sad%, listen%, ibsta%,
      iberr%, ibcntl&) As Integer
```

IBLN

Board Level
Device Level

IBLN
(Continued)

Input

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

Output

`listen`	Indicates whether or not a device is present
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 a board descriptor, then the bus associated with that board is tested for Listeners. If `ud` is a device descriptor, then `ibln` uses the access board 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.

IBLN

Board Level
Device Level

IBLN
(Continued)

Possible Errors

EARG	Either the `pad` or `sad` argument is invalid.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBLOC

Board Level
Device Level

IBLOC

Purpose

Go to Local.

DOS Format

C

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

QuickBASIC/BASIC

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

BASICA

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLibloc (int ud, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibloc Lib "gpib.dll"
     (ByVal ud%, ibsta%, iberr%, ibcntl&) As Integer
```

IBLOC

Board Level
Device Level

IBLOC
(Continued)

Input

`ud`	Board or device descriptor

Output

Function Return	The value of `ibsta`

Description

Board Level

If the board is not in a lockout state (LOK does not appear in the status word, `ibsta`), `ibloc` places the board in local mode. Otherwise, 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 board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBONL

Board Level
Device Level

IBONL

Purpose

Place the device or interface board online or offline.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibonl (ud%, v%)     or     status% = ilonl (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibonl (ud%, v%)     or     status% = ilonl (ud%, v%)
```

Direct Entry with C

```
DLLibonl (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibonl Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&)As Integer
```

IBONL

Board Level
Device Level

IBONL
(Continued)

Input

`ud`	Board or device descriptor
`v`	Indicates whether the board or device is to be put online or taken offline

Output

Function Return	The value of `ibsta`

Description

`ibonl` resets the board or device and places all its software configuration parameters in their pre-configured state. In addition, if `v` is zero, the device or interface board is taken offline. If `v` is non-zero, the device or interface board is left operational, or online.

If a device or an interface board is taken offline, the board or device descriptor (`ud`) is no longer valid. You must execute an `ibfind` or `ibdev` to access the board or device again.

Possible Errors

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

IBPAD

Board Level
Device Level

IBPAD

Purpose

Change the primary address.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibpad (ud%, v%)     or     status% = ilpad (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibpad (ud%, v%)     or     status% = ilpad (ud%, v%)
```

Direct Entry with C

```
DLLibpad (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibpad Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&)As Integer
```

IBPAD

Board Level
Device Level

IBPAD
(Continued)

Input

`ud`	Board or device descriptor
`v`	GPIB primary address

Output

Function Return	The value of `ibsta`

Description

`ibpad` sets the primary GPIB address of the board or device to `v`, an integer ranging from 0 to 30. If no error occurs during the call, then `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 board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBPCT Device Level IBPCT

Purpose

Pass control to another GPIB device with Controller capability.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibpct (ud%)     or     status% = ilpct (ud%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibpct (ud%)     or     status% = ilpct (ud%)
```

Direct Entry with C

```
DLLibpct (int ud, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibpct Lib "gpib.dll"
     (ByVal ud%, ibsta%, iberr%, ibcntl&) As Integer
```

IBPCT **Device Level** IBPCT (Continued)

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 board automatically unasserts the ATN line and goes to Controller Idle State. 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 board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBPPC

Board Level
Device Level

IBPPC

Purpose

Parallel poll configure.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibppc (ud%, v%)     or     status% = ilppc (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibppc (ud%, v%)     or     status% = ilppc (ud%, v%)
```

Direct Entry with C

```
DLLibppc (int ud, int v, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibppc Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&)As Integer
```

IBPPC

Board Level
Device Level

IBPPC
(Continued)

Input

`ud`	Board or device descriptor
`v`	Parallel poll enable/disable value

Output

Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, `ibppc` enables or disables the device from responding to parallel polls. The device is addressed and sent the appropriate parallel poll 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 a board descriptor, `ibppc` performs a local parallel poll configuration using the parallel poll configuration value `v`. Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero to send PPD. If no error occurs during the call, then `iberr` contains the previous value of the local parallel poll configuration.

For more information on parallel polling, refer to the NI-488.2 user manual.

IBPPC **Board Level** **IBPPC**
Device Level **(Continued)**

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 board is not configured to perform local parallel poll configuration (see `ibconfig`, option `IbcPP2`).
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRD

Board Level
Device Level

IBRD

Purpose

Read data from a device into a user buffer.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibrd (ud%, rdbuf$)
     or
status% = ilrd (ud%, rdbuf$, cnt&)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibrd (ud%, rdbuf$)
     or
status% = ilrd (ud%, rdbuf$, cnt&)
```

Direct Entry with C

```
DLLibrd (int ud, void _far *rdbuf, long cnt, int _far *ibsta,
         int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrd Lib "gpib.dll"
    (ByVal ud%, ByVal rdbuf$, ByVal cnt&, ibsta%, iberr%,
    ibcntl&) As Integer
```

IBRD

Board Level
Device Level

IBRD
(Continued)

Input

`ud`	Board or device descriptor
`cnt`	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 `cnt` bytes of data, and places the data into the buffer specified by `rdbuf`. The operation terminates normally when `cnt` 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 a board descriptor, `ibrd` reads up to `cnt` bytes of data from a GPIB device and places it into the buffer specified by `rdbuf`. A board-level `ibrd` assumes that the GPIB is already properly addressed. The operation terminates normally when `cnt` bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not the CIC, the CIC sends a Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBRD **Board Level** **IBRD**
Device Level **(Continued)**

Possible Errors

EABO — Either `cnt` bytes or END was not received within the timeout period or a Device Clear message was received after the read operation began.

EADR — Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB.

Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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

ECIC — Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

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

EOIP — Asynchronous I/O is in progress.

IBRDA

Board Level
Device Level

IBRDA

Purpose

Read data asynchronously from a device into a user buffer.

DOS Format

C

```
int ibrda (int ud, void *rdbuf, long cnt)
```

QuickBASIC/BASIC

```
CALL ibrda (ud%, rdbuf$)
     or
status% = ilrda (ud%, rdbuf$, cnt&)
```

BASICA

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

Windows Format

C

```
int ibrda (int ud, void *rdbuf, long cnt)
```

Visual Basic

```
CALL ibrda (ud%, rdbuf$)
     or
status% = ilrda (ud%, rdbuf$, cnt&)
```

Direct Entry with C

```
DLLibrda(int ud, void _far *rdbuf, long cnt, int _far *ibsta,
         short _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrda Lib "gpib.dll"
        (ByVal ud%, ByVal rdbuf$, ByVal cnt&, ibsta%, iberr%,
         ibcntl&) As Integer
```

IBRDA

Board Level
Device Level

IBRDA
(Continued)

Input

`ud`	Board or device descriptor
`cnt`	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 `cnt` bytes of data from a GPIB device, and places the data into the memory location specified by `rdbuf`. The operation terminates normally when `cnt` 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 a board descriptor, `ibrda` reads up to `cnt` 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 `cnt` bytes have been received or END is received. The operation terminates with an error if the board is not the CIC or if the CIC sends the Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBRDA

Board Level
Device Level

IBRDA
(Continued)

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 GPIB 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 three functions:

- `ibwait` If the returned `ibsta` mask has the CMPL bit set, then 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 board. Use `ibpad` and `ibsad`.

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

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

EOIP Asynchronous I/O is in progress.

IBRDF

Board Level
Device Level

IBRDF

Purpose

Read data from a device into a file.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibrdf (ud%, flname$)
     or
status% = ilrdf (ud%, flname$)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibrdf (ud%, flname$)
     or
status% = ilrdf (ud%, flname$)
```

Direct Entry with C

```
DLLibrdf (int ud, char _far *flname,short _far *ibsta,
          int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrdf Lib "gpib.dll"
         (ByVal ud%, ByVal flname$, ibsta%, iberr%, ibcntl&)
          As Integer
```

IBRDF

Board Level
Device Level

IBRDF
(Continued)

Input

`ud`	Board 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 a board descriptor, `ibrdf` reads data from a GPIB device and places the data into the file specified by `flname`. A board-level `ibrdf` assumes that the GPIB is already properly addressed. The operation terminates normally when END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not the CIC, the CIC sends a Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBRDF **Board Level** **IBRDF**
Device Level **(Continued)**

Possible Errors

EABO — Either `cnt` bytes or END was not received within the timeout period, or `ud` is a board descriptor and Device Clear was received after the read operation began.

EADR — Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB.

Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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

ECIC — Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

EFSO — `ibrdf` could not access `flname`.

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

EOIP — Asynchronous I/O is in progress.

IBRDI

Board Level
Device Level

IBRDI

Purpose

Read data from a device into a user integer buffer.

DOS Format

C

Not supported–use `ibrd`

QuickBASIC/BASIC

```
CALL ibrdi (ud%, rdbuf%(), cnt&)
     or
status% = ilrdi (ud%, rdbuf%(), cnt&)
```

BASICA

```
CALL ibrdi (ud%, rdbuf%(0), cnt%)
```

Windows Format

C

Not supported–use `ibrd`

Visual Basic

```
CALL ibrdi (ud%, rdbuf%(), cnt&)
     or
status% = ilrdi (ud%, rdbuf%(), cnt&)
```

Direct Entry with C

Not supported–use `ibrd`

IBRDI

Board Level
Device Level

IBRDI
(Continued)

Direct Entry with Visual Basic

```
Declare Function DLLibrd Lib "gpib.dll"
      (ByVal ud%, rdbuf%, ByVal cnt&, ibsta%, iberr%, ibcntl&)
       As Integer
```

Note: ***For direct entry with Visual Basic, the correct format is*** `DLLibrd`, ***not*** `DLLibrdi`.

Input

`ud`	Board or device descriptor
`cnt`	Number of bytes to be read from the GPIB

Output

`rdbuf`	Address of buffer into which integer data is read. You can replace `rdbuf%` with `rdbuf&` to read long integer data.
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, `ibrdi` addresses the GPIB, begins a read of up to `cnt` bytes of data from a GPIB device, and places the data into the memory location specified by `rdbuf`. The operation terminates normally when `cnt` 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`.

IBRDI

Board Level
Device Level

IBRDI
(Continued)

Board Level

If `ud` is a board descriptor, `ibrdi` reads up to `cnt` bytes of data from a GPIB device and places the data into the buffer specified by `rdbuf`. A board-level `ibrdi` assumes that the GPIB is already properly addressed. The operation terminates normally when `cnt` bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not the CIC, when the CIC sends a Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Note: ***The*** `cnt` ***parameter specifies the number of bytes to transfer. For example, if you want to transfer 16 integers, the number of bytes is 16*2=32.***

Possible Errors

EABO	Neither `cnt` bytes nor End was received within the timeout period or a Device Clear message was received after the read operation began.
EADR	Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB. Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.
EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRDIA

Board Level
Device Level

IBRDIA

Purpose

Read integer data asynchronously from a device into a user buffer.

DOS Format

C

Not supported–use `ibrda`

QuickBASIC/BASIC

```
CALL ibrdia (ud%, rdbuf%(), cnt&)
     or
status% = ilrdia (ud%, rdbuf%(), cnt&)
```

BASICA

```
CALL ibrdia (ud%, rdbuf%(0), cnt%)
```

Windows Format

C

Not supported–use `ibrda`

Visual Basic

```
CALL ibrdia (ud%, rdbuf%(), cnt&)
     or
status% = ilrdia (ud%, rdbuf%(), cnt&)
```

Direct Entry with C

Not supported–use `ibrda`

IBRDIA

Board Level
Device Level

IBRDIA
(Continued)

Direct Entry with Visual Basic

```
Declare Function DLLibrda Lib "gpib.dll"
       (ByVal ud%, rdbuf%, ByVal cnt&, ibsta%, iberr%, ibcntl&)
        As Integer
```

Note: ***For direct entry with Visual Basic, the correct format is*** `DLLibrd`, ***not*** `DLLibrdia`.

Input

`ud`	Board or device descriptor
`cnt`	Number of bytes to be read from the GPIB

Output

`rdbuf`	Address of buffer into which integer data is read. You can replace `rdbuf%` with `rdbuf&` to read long integer data.
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, `ibrdia` addresses the GPIB, begins an asynchronous read of up to `cnt` bytes of data from a GPIB device, and places the data into the memory location specified by `rdbuf`. The operation terminates normally when `cnt` 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`.

IBRDIA

Board Level
Device Level

IBRDIA
(Continued)

Board Level

If `ud` is a board descriptor, `ibrdia` reads up to `cnt` bytes of data from a GPIB device and places the data into the buffer specified by `rdbuf`. A board-level `ibrdia` assumes that the GPIB is already properly addressed. The operation terminates normally when `cnt` bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not the CIC, the CIC sends a Device Clear message 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 GPIB 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 three functions:

- `ibwait` If the returned `ibsta` mask has the CMPL bit set, then 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.

Note: ***The*** `cnt` ***parameter specifies the number of bytes to transfer. For example, if you want to transfer 16 integers, the number of bytes is 16*2=32.***

IBRDIA **Board Level** **IBRDIA**
Device Level **(Continued)**

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 board. Use `ibpad` and `ibsad`.

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

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

EOIP Asynchronous I/O is in progress.

IBRPP

Board Level
Device Level

IBRPP

Purpose

Conduct a parallel poll.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibrpp (ud%, ppr%)     or     status% = ilrpp (ud%, ppr%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibrpp (ud%, ppr%)     or     status% = ilrpp (ud%, ppr%)
```

Direct Entry with C

```
DLLibrpp (int ud, char _far *ppr,int _far *ibsta,
          int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrpp Lib "gpib.dll"
         (ByVal ud%, ppr%, ibsta%, iberr%, ibcntl&) As Integer
```

Input

ud Board or device descriptor

IBRPP

Board Level
Device Level

IBRPP
(Continued)

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 on parallel polling, refer to the NI-488.2 user manual.

Possible Errors

EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRSC Board Level IBRSC

Purpose

Request or release system control.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibrsc (ud%, v%)     or     status% = ilrsc (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibrsc (ud%, v%)     or     status% = ilrsc (ud%, v%)
```

Direct Entry with C

```
DLLibrsc(int ud, int v, int _far *ibsta, int _far *iberr,
         long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrsc Lib "gpib.dll"
       (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As
       Integer
```

IBRSC

Board Level

IBRSC
(Continued)

Input

`ud`	Board descriptor
`v`	Determines if system control is to be requested or released

Output

Function Return	The value of `ibsta`

Description

`ibrsc` requests or releases the capability to send Interface Clear (IFC) and Remote Enable (REN) messages to devices. If `v` is zero, the board releases system control and 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, then `iberr` contains the previous System Controller state of the board.

Possible Errors

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

IBRSP Device Level IBRSP

Purpose

Conduct a serial poll.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibrsp (ud%, spr%)     or     status% = ilrsp (ud%, spr%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibrsp (ud%, spr%)     or     status% = ilrsp (ud%, spr%)
```

Direct Entry with C

```
DLLibrsp  (int ud, char _far *spr, int _far *ibsta,
         int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrsp Lib "gpib.dll"
         (ByVal ud%, spr%, ibsta%, iberr%, ibcntl&) As Integer
```

IBRSP **Device Level** **IBRSP (Continued)**

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 byte is set, the device is requesting service. If 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 on serial polling, refer to the NI-488.2 user manual.

IBRSP **Device Level** **IBRSP (Continued)**

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 board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESTB	Autopolling is enabled and the serial poll queue has overflowed. Disable automatic serial polling or call `ibrsp` more often to keep the queue from overflowing.

IBRSV Board Level IBRSV

Purpose

Request service and change the serial poll status byte.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibrsv (ud%, v%)     or     status% = ilrsv (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibrsv (ud%, v%)     or     status% = ilrsv (ud%, v%)
```

Direct Entry with C

```
DLLibrsv(int ud, int v, int _far *ibsta, int _far *iberr,
         long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibrsv Lib "gpib.dll"
      (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As Integer
```

IBRSV

Board Level

IBRSV
(Continued)

Input

`ud`	Board descriptor
`v`	Serial poll status byte

Output

Function Return	The value of `ibsta`

Description

`ibrsv` requests service from the Controller and provides the Controller with an application-dependent status byte when the Controller serial polls the GPIB board.

The value `v` is the status byte that the GPIB board returns when serial polled by the Controller-In-Charge. If bit 6 (hex 40) is set in `v`, the GPIB board 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 a board.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBSAD

Board Level
Device Level

IBSAD

Purpose

Change or disable the secondary address.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibsad (ud%, v%)     or     status% = ilsad (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibsad (ud%, v%)     or     status% = ilsad (ud%, v%)
```

Direct Entry with C

```
DLLibsad(int ud, int v, int _far *ibsta, int _far *iberr,
         long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibsad Lib "gpib.dll"
     (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&)
      As Integer
```

IBSAD

Board Level
Device Level

IBSAD
(Continued)

Input

`ud`	Board or device descriptor
`v`	GPIB secondary address

Output

Function Return	The value of `ibsta`

Description

`ibsad` changes the secondary GPIB address of the board or device to `v`, an integer in the range 96 to 126 (hex 60 to hex 7E) or zero. If `v` is zero, secondary addressing is disabled. If no error occurs during the call, then the previous 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 board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBSIC Board Level IBSIC

Purpose

Assert interface clear.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibsic (ud%)     or     status% = ilsic (ud%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibsic (ud%)     or     status% = ilsic (ud%)
```

Direct Entry with C

```
DLLibsic (int ud, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibsic Lib "gpib.dll"
         (ByVal ud%, ibsta%, iberr%, ibcntl&) As Integer
```

IBSIC — Board Level — IBSIC (Continued)

Input

`ud`	Board descriptor

Output

Function Return	The value of `ibsta`

Description

`ibsic` asserts the GPIB interface clear (IFC) line for at least 100 µs if the GPIB board is System Controller. This initializes the GPIB and makes the interface board 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 a board.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	Board does not have System Controller capability.

IBSRE Board Level IBSRE

Purpose

Set or clear the Remote Enable line.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibsre (ud%, v%)     or     status% = ilsre (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibsre (ud%, v%)     or     status% = ilsre (ud%, v%)
```

Direct Entry with C

```
DLLibsre(int ud, int v, int _far *ibsta, int _far *iberr,
         long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibsre Lib "gpib.dll"
       (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As
       Integer
```

IBSRE

Board Level

IBSRE
(Continued)

Input

`ud`	Board 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 a board.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	Board does not have System Controller capability.

IBSRQ Board Level IBSRQ

Purpose

Request an SRQ interrupt routine.

DOS Format

C

```
void ibsrq (void (_far *funcname) (void));
```

QuickBASIC/BASIC

Not supported–see Chapter 7, *GPIB Programming Techniques*, in the *NI-488.2 User Manual for DOS* for information about ON SRQ capability.

BASICA

Not supported–see Chapter 7, *GPIB Programming Techniques*, in the *NI-488.2 User Manual for DOS* for information about ON SRQ capability.

Windows Format

Not supported

Input

`funcname`	C interrupt-handling routine

Description

`ibsrq` establishes a call to the C routine `funcname` whenever the SRQI bit is set in the status word (`ibsta`). If SRQI is set, the language interface calls `funcname` before returning to the application program. If `ibsrq` is called with `funcname` equal to NULL, SRQ servicing is turned off.

Note: ***You must disable automatic serial polling with*** `ibconfig` ***(option*** `IbcAUTOPOLL`***) before using this function. Also, device-level calls should not be used when*** `ibsrq` ***is in effect. Device-level calls mask the SRQI bit, preventing*** `funcname` ***from being called.***

IBSTOP

Board Level
Device Level

IBSTOP

Purpose

Abort asynchronous I/O operation.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibstop (ud%)     or     status% = ilstop (ud%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibstop (ud%)     or     status% = ilstop (ud%)
```

Direct Entry with C

```
DLLibstop (int ud, int _far *ibsta, int _far *iberr,
           long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibstop Lib "gpib.dll"
         (ByVal ud%, ibsta%, iberr%, ibcntl&) As Integer
```

IBSTOP

Board Level
Device Level

IBSTOP
(Continued)

Input

`ud`	Board or device descriptor

Output

Function Return	The value of `ibsta`

Description

The `ibstop` function aborts any asynchronous read, write, or command operation that is in progress and resynchronizes the application with the driver. If asynchronous I/O is in progress, the error bit is set in the status word, `ibsta`, and EABO is returned, indicating that the I/O was successfully stopped.

Possible Errors

EABO	Asynchronous I/O was successfully stopped.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

IBTMO

Board Level
Device Level

IBTMO

Purpose

Change or disable the I/O timeout period.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibtmo (ud%, v%)     or     status% = iltmo (ud%, v%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibtmo (ud%, v%)     or     status% = iltmo (ud%, v%)
```

Direct Entry with C

```
DLLibtmo(int ud, int v, int _far *ibsta, int _far *iberr,
         long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibtmo Lib "gpib.dll"
       (ByVal ud%, ByVal v%, ibsta%, iberr%, ibcntl&) As
       Integer
```

IBTMO

Board Level
Device Level

IBTMO
(Continued)

Input

`ud`	Board or device descriptor
`v`	Timeout duration code

Output

Function Return	The value of `ibsta`

Description

The timeout period is set to `v`. The timeout period is used to select the maximum duration allowed for a synchronous operation (for example, `ibrd` and `ibwait`). If the operation does not complete before the timeout period elapses, then the operation is aborted and TIMO is returned in `ibsta`. See Table 1-8 for a list of valid timeout values. These timeout values represent the minimum timeout period. The actual period might be longer.

Possible Errors

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

IBTMO

Board Level
Device Level

IBTMO
(Continued)

Table 1-8. Timeout Code Values

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

IBTRAP **Board Level** IBTRAP

Purpose

Change the trap and display modes of the GPIB Applications Monitor utility, `appmon`.

DOS Format

C

```
void ibtrap (int mask, int mode)
```

QuickBASIC/BASIC

```
CALL ibtrap (mask%, mode%)
     or
status% = iltrap (mask%, mode%)
```

BASICA

```
CALL ibtrap (mask%, mode%)
```

Windows Format

Not supported–refer to Chapter 6, *GPIB Spy*, in the *NI-488.2 User Manual for Windows* for information about the monitoring utility for Windows applications.

Input

`mask`	Trap bit mask
`mode`	Trap mode

IBTRAP **Board Level** IBTRAP (Continued)

Description

`mask` specifies a bit mask with the same bit assignments as `ibsta`. Each mask bit can be set to trap a call to the driver, when the corresponding bit appears in the status word after the GPIB call. If all the bits are set, then every GPIB call is trapped. Mode determines whether the recording and trapping occur. The valid values of mode are as follows:

Mode Value	Effect
1	Turn monitor off. No recording or trapping occurs
2	Turn recording on. All calls are recorded, but no trapping occurs.
3	Turn recording and trapping on. All calls are recorded and the monitor is displayed whenever a trap condition occurs.

Refer to Chapter 6, *appmon–GPIB Applications Monitor*, in the *NI-488.2 User Manual for DOS* for more information about `appmon`.

Possible Errors

EARG	`mode` is an invalid value.
ECAP	The GPIB Applications Monitor is not installed.

IBTRG Device Level IBTRG

Purpose

Trigger selected device.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibtrg (ud%)     or     status% = iltrg (ud%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibtrg (ud%)     or     status% = iltrg (ud%)
```

Direct Entry with C

```
DLLibtrg (int ud, int _far *ibsta, int _far *iberr,
          long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibtrg Lib "gpib.dll"
          (ByVal ud%, ibsta%, iberr%, ibcntl&) As Integer
```

Input

ud	Device descriptor

IBTRG

Device Level

IBTRG
(Continued)

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 board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBWAIT

Board Level
Device Level

IBWAIT

Purpose

Wait for GPIB events.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibwait (ud%, mask%)     or     status% = ilwait (ud%, mask%)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibwait (ud%, mask%)     or     status% = ilwait (ud%, mask%)
```

Direct Entry with C

```
DLLibwait(int ud, int mask, int _far *ibsta, int _far *iberr,
        long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibwait Lib "gpib.dll"
        (ByVal ud%, ByVal  mask%, ibsta%, iberr%, ibcntl&)
         As Integer
```

IBWAIT

Board Level
Device Level

IBWAIT
(Continued)

Input

`ud`	Board or device descriptor
`mask`	Bit mask of GPIB events to wait on

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, then the function waits indefinitely for one or more of the specified events to occur. The `ibwait` mask bits are identical to the `ibsta` bits and they are described in Table 1-9. If `ud` is a device descriptor, the only valid wait mask bits are TIMO, END, RQS and CMPL. If `ud` is a board descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period using the `ibtmo` function.

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 board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ESRQ	Device level: If RQS is set in the wait mask, then ESRQ indicates that the *Stuck SRQ* condition exists. For more information on serial polling, refer to the NI-488.2 user manual.

IBWAIT

Board Level
Device Level

IBWAIT
(Continued)

Table 1-9. Wait Mask Layout

Mnemonic	Bit Pos.	Hex Value	Description
ERR	15	8000	GPIB error
TIMO	14	4000	Time limit exceeded
END	13	2000	GPIB board detected END or EOS
SRQI	12	1000	SRQ asserted (board only)
RQS	11	800	Device requesting service (device only)
SPOLL	10	400	The board has been serial polled by the Controller
EVENT	9	200	A DTAS, DCAS, or IFC event has occurred
CMPL	8	100	I/O completed
LOK	7	80	GPIB board is in Lockout State
REM	6	40	GPIB board is in Remote State
CIC	5	20	GPIB board is CIC
ATN	4	10	Attention is asserted
TACS	3	8	GPIB board is Talker
LACS	2	4	GPIB board is Listener
DTAS	1	2	GPIB board is in Device Trigger State
DCAS	0	1	GPIB board is in Device Clear State

IBWRT

Board Level
Device Level

IBWRT

Purpose

Write data to a device from a user buffer.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibwrt (ud%, wrtbuf$)
     or
status% = ilwrt (ud%, wrtbuf$, cnt&)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibwrt (ud%, wrtbuf$)
     or
status% = ilwrt (ud%, wrtbuf$, cnt&)
```

Direct Entry with C

```
DLLibwrt(int ud, void _far *wrtbuf, long cnt, int _far *ibsta,
         int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibwrt Lib "gpib.dll"
        (ByVal ud%, ByVal wrtbuf$, ByVal cnt&, ibsta%,
         iberr%,ibcntl&) As Integer
```

IBWRT

Board Level
Device Level

IBWRT
(Continued)

Input

`ud`	Board or device descriptor
`wrtbuf`	Address of the buffer containing the bytes to write
`cnt`	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 `cnt` bytes from the memory location specified by `wrtbuf` to a GPIB device. The operation terminates normally when `cnt` bytes have been sent. The operation terminates with an error if `cnt` 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 a board descriptor, `ibwrt` writes `cnt` 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 `cnt` bytes have been sent. The operation terminates with an error if `cnt` bytes could not be sent within the timeout period or, if the board is not CIC, the CIC sends the Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Possible Errors

EABO — Either `cnt` bytes were not sent within the timeout period, or a Device Clear message was received after the write operation began.

EADR — Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB.

Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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

ECIC — Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

ENEB — The interface board 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

IBWRTA

Purpose

Write data asynchronously to a device from a user buffer.

DOS Format

C

```
int ibwrta (int ud, void *wrtbuf, long cnt)
```

QuickBASIC/BASIC

```
CALL ibwrta (ud%, wrtbuf$)
     or
status% = ilwrta (ud%, wrtbuf$, cnt&)
```

BASICA

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

Windows Format

C

```
int ibwrta (int ud, void *wrtbuf, long cnt)
```

Visual Basic

```
CALL ibwrta (ud%, wrtbuf$)
     or
status% = ilwrta (ud%, wrtbuf$, cnt&)
```

Direct Entry with C

```
DLLibwrta (int ud, void _far *wrtbuf, long cnt, int _far *ibsta,
           int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibwrta Lib "gpib.dll"
         (ByVal ud%, ByVal wrtbuf$, ByVal cnt&, ibsta%, iberr%,
          ibcntl&) As Integer
```

IBWRTA

Board Level
Device Level

IBWRTA
(Continued)

Input

`ud`	Board or device descriptor
`wrtbuf`	Address of the buffer containing the bytes to write
`cnt`	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 and writes `cnt` bytes from the buffer `wrtbuf` to a GPIB device. The operation terminates normally when `cnt` bytes have been sent. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Board Level

If `ud` is a board descriptor, `ibwrta` begins an asynchronous write of `cnt` bytes of data from the buffer pointed to by `wrtbuf` to a GPIB device. A board-level `ibwrta` assumes that the GPIB is already properly addressed. The operation terminates normally when `cnt` bytes have been sent. The operation terminates with an error if the board is not CIC or if the CIC sends the Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBWRTA

Board Level
Device Level

IBWRTA
(Continued)

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 GPIB 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 three functions:

- `ibwait` If the returned `ibsta` mask has the CMPL bit set, then 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 board. Use `ibpad` and `ibsad`.

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

ENEB The interface board 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

IBWRTF

Purpose

Write data to a device from a file.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL ibwrtf (ud%, flname$)
     or
status% = ilwrtf (ud%, flname$)
```

BASICA

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

Windows Format

C

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

Visual Basic

```
CALL ibwrtf (ud%, flname$)
     or
status% = ilwrtf (ud%, flname$)
```

Direct Entry with C

```
DLLibwrtf (int ud, char _far *flname, int _far *ibsta,
           int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Function DLLibwrtf Lib "gpib.dll"
        (ByVal ud%, ByVal flname$, ibsta%, iberr%, ibcntl&)
         As Integer
```

IBWRTF

Board Level
Device Level

IBWRTF
(Continued)

Input

`ud`	Board or device descriptor
`flname`	Name of file containing the data to be written

Output

Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, `ibwrtf` addresses the GPIB and writes all of the bytes in 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 a board descriptor, `ibwrtf` writes all of the bytes in the file `flname` to a GPIB device. A board-level `ibwrtf` assumes that the GPIB is already properly addressed. The operation terminates normally when all of the bytes have been sent. The operation terminates with an error if all of the bytes could not be sent within the timeout period or, if the board is not CIC, the CIC sends the Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBWRTF **Board Level** **IBWRTF**
Device Level **(Continued)**

Possible Errors

EABO Either the file could not be transferred within the timeout period or a Device Clear message was received after the write operation began.

EADR Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB.

Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.

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

EFSO `ibwrtf` could not access `flname`.

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

EOIP Asynchronous I/O is in progress.

IBWRTI

Board Level
Device Level

IBWRTI

Purpose

Write data to a device from a user integer buffer.

DOS Format

C

Not supported–use `ibwrt`

QuickBASIC/BASIC

```
CALL ibwrti (ud%, wrtbuf%(), cnt&)
     or
status% = ilwrti (ud%, wrtbuf%(), cnt&)
```

BASICA

```
CALL ibwrti (ud%, wrtbuf%(0), cnt%)
```

Windows Format

C

Not supported–use `ibwrt`

Visual Basic

```
CALL ibwrti (ud%, wrtbuf%(), cnt&)
     or
status% = ilwrti (ud%, wrtbuf%(), cnt&)
```

Direct Entry with C

Not supported–use `ibwrt`

IBWRTI

Board Level
Device Level

IBWRTI
(Continued)

Direct Entry with Visual Basic

```
Declare Function DLLibwrt Lib "gpib.dll"
       (ByVal ud%, wrtbuf%, ByVal cnt&, ibsta%, iberr%,ibcntl&)
        As Integer
```

Note: ***For direct entry with Visual Basic, the correct format is*** `DLLibwrt`, ***not*** `DLLibwrti`.

Input

`ud`	Board or device descriptor
`wrtbuf`	Address of the integer buffer containing the bytes to write. You can replace `wrtbuf%` with `wrtbuf&` to send long integer data.
`cnt`	Number of bytes to be written

Output

Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, `ibwrti` addresses the GPIB and writes `cnt` bytes from the memory location specified by `wrtbuf` to a GPIB device. The operation terminates normally when `cnt` bytes have been sent. The operation terminates with an error if `cnt` bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBWRTI

Board Level
Device Level

IBWRTI
(Continued)

Board Level

If `ud` is a board descriptor, `ibwrti` writes `cnt` bytes of data from the buffer specified by `wrtbuf` to a GPIB device; a board-level `ibwrti` assumes that the GPIB is already properly addressed. The operation terminates normally when `cnt` bytes have been sent. The operation terminates with an error if `cnt` bytes could not be sent within the timeout period or, if the board is not CIC, the CIC sends the Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Note: ***The*** `cnt` ***parameter specifies the number of bytes to transfer. For example, if you want to transfer 16 integers, the number of bytes is 16*2=32.***

Possible Errors

EABO	Either `cnt` bytes were not sent within the timeout period, or a Device Clear message was received after the write operation began.
EADR	Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB. Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.
EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners were detected on the bus.
EOIP	Asynchronous I/O is in progress.

IBWRTIA

Board Level
Device Level

IBWRTIA

Purpose

Write data asynchronously to a device from a user integer buffer.

DOS Format

C

Not supported–use `ibwrta`

QuickBASIC/BASIC

```
CALL ibwrtia (ud%, wrtbuf%(), cnt&)
     or
status% = ilwrtia (ud%, wrtbuf%(), cnt&)
```

BASICA

```
CALL ibwrtia (ud%, wrtbuf%(0), cnt%)
```

Windows Format

C

Not supported–use `ibwrta`

Visual Basic

```
CALL ibwrtia (ud%, wrtbuf%(), cnt&)
     or
status% = ilwrtia (ud%, wrtbuf%(), cnt&)
```

Direct Entry with C

Not supported–use `ibwrta`

IBWRTIA

Board Level
Device Level

IBWRTIA
(Continued)

Direct Entry with Visual Basic

```
Declare Function DLLibwrta Lib "gpib.dll"
        (ByVal ud%, wrtbuf%, ByVal cnt&, ibsta%, iberr%,
          ibcntl&) As Integer
```

Note: ***For direct entry with Visual Basic, the correct format is*** `DLLibwrt`, ***not*** `DLLibwrtia`.

Input

`ud`	Board or device descriptor
`wrtbuf`	Address of the integer buffer containing the bytes to write. You can replace `wrtbuf%` with `wrtbuf&` to send long integer data.
`cnt`	Number of bytes to be written

Output

Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, `ibwrtia` addresses the GPIB and writes `cnt` bytes from the buffer pointed to by `wrtbuf` to a GPIB device. The operation terminates normally when `cnt` bytes have been sent. The operation terminates with an error if `cnt` bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBWRTIA

Board Level
Device Level

IBWRTIA
(Continued)

Board Level

If `ud` is a board descriptor, `ibwrtia` begins an asynchronous write of `cnt` bytes of data from the buffer pointed to by `wrtbuf` to a GPIB device. A board-level `ibwrtia` assumes that the GPIB is already properly addressed. The operation terminates normally when `cnt` bytes have been sent. The operation terminates with an error if `cnt` bytes could not be sent within the timeout period or, if the board is not CIC, the CIC sends the Device Clear message 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 GPIB 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 three functions:

- `ibwait` If the returned `ibsta` mask has the CMPL bit set, then 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.

Note: ***The*** `cnt` ***parameter specifies the number of bytes to transfer. For example, if you want to transfer 16 integers, the number of bytes is 16*2=32.***

IBWRTIA

Board Level
Device Level

IBWRTIA
(Continued)

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 board. Use `ibpad` and `ibsad`.
EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the NI-488.2 user manual.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners were detected on the bus.
EOIP	Asynchronous I/O is in progress.

Chapter 2
NI-488.2 Routines

This chapter lists the available NI-488.2 routines and then describes the purpose, format, input and output parameters, and possible errors for each routine.

For general programming information, refer to the NI-488.2 user manual. The user manual explains how to develop and debug your program. It also describes the example programs included with your NI-488.2 software.

Routine Names

The routines in this chapter are listed alphabetically.

Purpose

Each routine description includes a brief statement of the purpose of the routine.

DOS Format

The DOS format is given for each of the languages supported by the NI-488.2 software:

- Microsoft C version 5.1 or higher
- Microsoft Professional BASIC version 7.0 or higher and Microsoft Visual Basic for MS-DOS version 1.0 or higher
- Microsoft QuickBASIC version 4.0 or higher
- BASICA and GWBASIC

Windows Format

The Windows format is given for the following:

- Microsoft C (version 5.1 or higher), LabWindows/CVI for Windows, and Borland C++ (version 2.0 or higher)
- Microsoft Visual Basic version 1.0 or higher
- Direct entry into the Windows Dynamic Link Library `gpib.dll`
  - Direct entry for Microsoft C version 5.1 or higher
  - Direct entry for Microsoft Visual BASIC version 1.0 or higher

Input and Output

The input and output parameters for each routine are listed. Most of the NI-488.2 routines 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 this 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

The description section gives details about the purpose and effect of each routine.

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 included with your NI-488.2 software. The example programs are described in Chapter 2 of the NI-488.2 user manual.

Possible Errors

Each routine description includes a list of errors that could occur when the routine is invoked.

List of NI-488.2 Routines

The following table contains an alphabetical list of each NI-488.2 routine.

Table 2-1. List of NI-488.2 Routines

Routine	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
`GenerateREQF`	Cancel service request generated by `GenerateREQT`
`GenerateREQT`	Request service from the GPIB Controller-In-Charge
`GotoMultAddr`	Place the driver in multiple address mode
`PassControl`	Pass control to another device with Controller capability
`PPoll`	Perform a parallel poll on the GPIB
`PPollConfig`	Configure a device for 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 board ID to be a Listener in preparation for `RcvRespMsg`
`ResetSys`	Reset and initialize IEEE 488.2-compliant devices

(continues)

Table 2-1. List of NI-488.2 Routines (Continued)

Routine	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 the 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 AllSpoll

Purpose

Serial poll all devices.

DOS Format

C

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

QuickBASIC/BASIC

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

BASICA

```
CALL AllSpoll (boardID%, addrlist%(0), resultlist%(0))
```

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLAllSpoll(int boardID, Addr4882_t _far addrlist[],
            short _far  resultlist[], int _far *ibsta,
            int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLAllSpoll Lib "gpib.dll"
       (ByVal boardID%, addrlist%, resultlist%, ibsta%, iberr%,
        ibcntl&)
```

AllSpoll

AllSpoll
(Continued)

Input

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

DevClear DevClear

Purpose

Clear a single device.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLDevClear (int boardID, Addr4882_t address, int _far *ibsta,
             int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLDevClear Lib "gpib.dll"
       (ByVal boardID%, ByVal address%, ibsta%, iberr%,
       ibcntl&)
```

Input

`boardID`	The interface board number
`address`	Address of the device you want to clear

DevClear

DevClear
(Continued)

Description

`DevClear` sends the Selected Device Clear (SDC) GPIB message to the device described by `address`. If `address` is the constant `NOADDR`, then the Universal Device Clear (DCL) message is sent to all devices.

Possible Errors

EARG	An `address` parameter is invalid (out of range).
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

DevClearList DevClearList

Purpose

Clear multiple devices.

DOS Format

C

```
void DevClearList (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL DevClearList (boardID%, addrlist%(0))
```

Windows Format

C

```
void DevClearList (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLDevClearList (int boardID, Addr4882_t _far addrlist[],
                 int _far *ibsta, int _far *iberr,
                 long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLDevClearList Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%, ibcntl&)
```

DevClearList

DevClearList
(Continued)

Input

`boardID`	The interface board 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`, then the Universal Device Clear (DCL) message is sent to all the devices on the bus.

Possible Errors

EARG	An invalid address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

EnableLocal EnableLocal

Purpose

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

DOS Format

C

```
void EnableLocal (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL EnableLocal (boardID%, addrlist%(0))
```

Windows Format

C

```
void EnableLocal (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLEnableLocal (int boardID, Addr4882_t _far addrlist[],
                int _far *ibsta, int _far *iberr,
                long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLEnableLocal Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%, ibcntl&)
```

EnableLocal

EnableLocal
(Continued)

Input

`boardID`	The interface board 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 in local mode. If `addrlist` contains only the constant `NOADDR`, then the Remote Enable (REN) GPIB line is unasserted.

Possible Errors

EARG	An invalid address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface board is not configured as System Controller.

EnableRemote EnableRemote

Purpose

Enable remote GPIB programming for devices.

DOS Format

C

```
void EnableRemote (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL EnableRemote (boardID%, addrlist%(0))
```

Windows Format

C

```
void EnableRemote (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLEnableRemote (int boardID, Addr4882_t _far addrlist[],
                 int _far *ibsta, int _far *iberr,
                 long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLEnableRemote Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%, ibcntl&)
```

EnableRemote

EnableRemote
(Continued)

Input

`boardID`	The interface board 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 in a listen-active state.

Possible Errors

EARG	An invalid address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface board is not configured as System Controller.

FindLstn FindLstn

Purpose

Find listening devices on the GPIB.

DOS Format

C

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

QuickBASIC/BASIC

```
CALL FindLstn (boardID%, padlist%(), resultlist%(), limit%)
```

BASICA

```
CALL FindLstn (boardID%, padlist%(0), resultlist%(0), limit%)
```

Windows Format

C

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

Visual Basic

```
CALL FindLstn (boardID%, padlist%(), resultlist%(), limit%)
```

Direct Entry with C

```
DLLFindLstn (int boardID, Addr4882_t _far padlist[],
             Addr4882_t _far resultlist[], int limit,
             int _far *ibsta, int _far *iberr,
             long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLFindLstn Lib "gpib.dll"
       (ByVal boardID%, padlist%, resultlist%, ByVal limit%,
        ibsta%, iberr%, ibcntl&)
```

FindLstn FindLstn (Continued)

Input

`boardID`	The interface board 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`, then 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`.

FindLstn

FindLstn
(Continued)

Possible Errors

EARG	An invalid primary address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board 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 FindRQS

Purpose

Determine which device is requesting service.

DOS Format

C

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

QuickBASIC/BASIC

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

BASICA

```
CALL FindRQS (boardID%, addrlist%(0), result%)
```

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLFindRQS (int boardID, Addr4882_t _far addrlist[],
            short _far *result, int _far *ibsta,
            int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLFindRQS Lib "gpib.dll"
       (ByVal boardID%, addrlist%, result%, ibsta%, iberr%,
       ibcntl&)
```

FindRQS

FindRQS
(Continued)

Input

`boardID`	The interface board 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, then the index corresponding to `NOADDR` in `addrlist` is returned in `ibcntl` and ETAB is returned in `iberr`.

Possible Errors

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

GenerateREQF GenerateREQF

Purpose

Cancel service request generated by GenerateREQT.

DOS Format

C

```
void GenerateREQF (int boardID, unsigned short GPIBaddr)
```

QuickBASIC/BASIC

Not supported

BASICA

Not supported

Windows Format

C

```
void GenerateREQF (int boardID, unsigned short GPIBaddr)
```

Visual Basic

Not supported

Direct Entry with C

```
DLLGenerateREQF (int boardID, unsigned short GPIBaddr,
                 int _far *ibsta, int _far *iberr,
                 long _far *ibcntl)
```

Direct Entry with Visual Basic

Not supported

GenerateREQF

GenerateREQF
(Continued)

Input

`boardID`	The interface board number
`GPIBaddr`	The 5-bit GPIB addresses of the simulated device that is no longer requesting service.

Description

The driver keeps track of the simulated devices currently requesting service so that it knows when to unassert the SRQ bus line. `GenerateREQF` cancels the service request of the simulated device at `GPIBaddr`. The driver unasserts the SRQ line if no other simulated devices are requesting service. The driver usually calls `GenerateREQF` automatically after the simulated device is serial polled. You can call this function if you want to cancel the request for service before the device is serial polled.

For an example of the `GenerateREQF` routine call, refer to the description of the `GotoMultAddr` routine.

Possible Errors

EARG	`GPIBaddr` is an invalid 5-bit GPIB address. See `GotoMultAddr` for a description of valid 5-bit GPIB addresses.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	`boardID` is not installed or is not properly configured
EOIP	Asynchronous I/O is in progress.

GenerateREQT GenerateREQT

Purpose

Request service from the GPIB Controller-In-Charge.

DOS Format

C

```
void GenerateREQT (int boardID, unsigned short GPIBaddr)
```

QuickBASIC/BASIC

Not supported

BASICA

Not supported

Windows Format

C

```
void GenerateREQT (int boardID, unsigned short GPIBaddr)
```

Visual Basic

Not supported

Direct Entry with C

```
DLLGenerateREQT (int boardID, unsigned short GPIBaddr,
                 int _far *ibsta, int _far *iberr,
                 long _far *ibcntl)
```

Direct Entry with Visual Basic

Not supported

GenerateREQT

GenerateREQT
(Continued)

Input

`boardID`	The interface board number
`GPIBaddr`	The 5-bit GPIB address of the simulated device that is requesting service

Description

The driver keeps track of the simulated devices currently requesting service so that it knows when to assert and unassert the SRQ bus line. Use `GenerateREQT` when the simulated device at `GPIBaddr` needs service from the CIC. GenerateREQT causes the driver to assert the SRQ line. When the Controller determines that SRQ is asserted, it conducts a serial poll of the device. The `spollfunc` then returns the appropriate serial poll response byte. The driver sets the RSV (Request Service) bit to 1 before sending the response byte to the Controller.

For an example of the `GenerateREQT` routine call, refer to the description of the `GotoMultAddr` routine.

Possible Errors

EARG	`GPIBaddr` is an invalid 5-bit GPIB address. See `GotoMultAddr` for a description of valid 5-bit GPIB addresses.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	`boardID` is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

GotoMultAddr GotoMultAddr

Purpose

Place the driver in multiple primary or secondary address mode.

DOS Format

C

```
void GotoMultAddr (int boardID, unsigned short type,
                   unsigned short (_far *addrfunc)(),
                   unsigned short (_far *spollfunc)())
```

QuickBASIC/BASIC

Not supported

BASICA

Not supported

Windows Format

C

```
void GotoMultAddr (int boardID, unsigned short type,
                   unsigned short (_far *addrfunc)(),
                   unsigned short (_far *spollfunc)())
```

Visual Basic

Not supported

Direct Entry with C

```
DLLGotoMultAddr (int boardID,unsigned short type,
                 unsigned short (_far *addrfunc)(),
                 unsigned short (_far *spollfunc)()),
                 int _far *ibsta, int _far *iberr,
                 long _far *ibcntl)
```

Direct Entry with Visual Basic

Not supported

GotoMultAddr

GotoMultAddr
(Continued)

Input

`boardID`	The interface board number
`type`	Either the constant `MultAddrPrimary` or `MultAddrSecondary`
`addrfunc`	The address of your address selection function
`spollfunc`	The address of your serial poll response function

Description

`GotoMultAddr` places the driver in multiple address mode. You must call `GotoMultAddr` once for every board at the beginning of any program that simulates multiple GPIB addresses. The `type` parameter specifies whether you want to use primary or secondary address mode. You cannot use both.

`addrfunc` and `spollfunc` are pointers to functions you have written as part of the application. `addrfunc` points to your address selection function. It is called whenever a GPIB address is on the bus. Your address selection function must determine whether the address is the address of one of the simulated GPIB devices. If it is a simulated address, `addrfunc` should return 1. If it is not, `addrfunc` should return 0. `spollfunc` points to your serial poll response function. It is called whenever one of the simulated devices is serial polled. `spollfunc` should return the serial poll response byte.

To disable multiple address mode, call `ibonl` with a 0 or 1. You must always call `ibonl` with a 0 before your application program terminates. Otherwise, the driver maintains the `addrfunc` and `spollfunc` pointers and might try to access the functions when they are no longer in memory, causing your computer to lock up.

Before you use the `GotoMultAddr` routine, make sure that hardware interrupts on the interface board have been enabled with either the `ibconf` utility in DOS, the GPIB software configuration utility in Windows, or the `ibconfig` function. Because the `addrfunc` and `spollfunc` functions are called at interrupt time, you must take special care when writing your function code. Follow these rules when writing functions that are called at interrupt time:

- Return from the interrupt call as soon as possible. Performing large calculations prevents the system from performing other interrupt activities, such as maintaining the system clock.

GotoMultAddr GotoMultAddr (Continued)

- Make sure that any function you call is re-entrant. (Most system functions are not re-entrant.) This includes DOS and BIOS functions, standard C library function, Windows functions, and GPIB functions.
- Do not use a large amount of stack space. The stack provided to your function has about 512 bytes available for its use. You should disable run-time stack overflow checking using the `/Gs` option of the Microsoft C compiler.
- Ensure that your code and data segments are `fixed` in memory if your application is written for Microsoft Windows 3. You can do this in the module definition file of your application.

The Address Selection Function

The driver calls the address selection function whenever a primary or secondary GPIB address is present on the bus. The driver passes the GPIB address to the function. The function determines whether to accept or reject the given address. If the function accepts the address, the interface board uses the given GPIB address. The application program can then read from or write to the bus as if it were the device at the given GPIB address.

Here is the function prototype of the address selection function:

```
unsigned short _far _loadds addrfunc (short board, unsigned short
                                      type,unsigned short addr)
```

The `_far` directive tells the compiler to generate a far return when this function exits. The `_loadds` directive tells the compiler to load the Data Segment register with the default data segment of the application. The function can now access the global variables of the application.

`board` is the index of the interface board on which the GPIB address is present. `type` is the type of address that is present on the bus; either a talk address, a listen address, or a talk address while the board is in Serial Poll Mode State. These types are defined by the constants: `MultAddrListen`, `MultAddrTalk`, and `MultAddrSerialPoll` which are defined in the language interface include file. `addr` is the 5-bit GPIB address that is currently present on the bus.

This function should return a non-zero value to accept the given address. It should return zero to reject the address.

GotoMultAddr GotoMultAddr (Continued)

Serial Poll Response Function

The driver calls the serial poll response function when the interface board is serial polled by the GPIB Controller-In-Charge. The driver passes the GPIB address of the device being polled to this function. This function should return an 8-bit serial poll response byte which is sent to the Controller. If the simulated device is requesting service through the use of the `GenerateREQT` function at the time of the poll, the driver sets the Request Service (RSV) bit to 1 before sending the response byte to the Controller.

Before the application calls the serial poll response function, it calls the address selection function with `type` set to `MultAddrSerialPoll`. Because the address selection function is called first, it can return zero to reject the address. This prevents the driver from calling the serial poll response function.

Here is the function prototype of the serial poll response function:

```
unsigned short _far _loadds spollfunc (short board,
                                       unsigned short addr)
```

`board` is the index of the interface board on which the GPIB address is present. `addr` is the 5-bit GPIB address that is currently present on the bus.

This function should return the 8-bit serial poll response byte that is sent to the Controller.

Possible Errors

EARG	The `type` parameter is invalid.
ECAP	Hardware interrupt s are disabled.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

GotoMultAddr GotoMultAddr (Continued)

Example

The following example program in C simulates four GPIB devices at primary addresses 1, 3, 24, and 30.

```
#include <stdio.h>
#include <string.h>
#include <process.h>
#include <bios.h>
#include "c:\at-gpib\c\decl.h"

#define TRUE  1
#define FALSE 0

#define LAD  0x20  /* listen address mask */
#define TAD  0x40  /* talk address mask   */

#define BUFSIZE 512

/*
 *    Globals.
 */
short addressed = FALSE;
unsigned short address;
char buffer[ BUFSIZE + 2 ];

/*
 *  The following function implements the address selection call-
 *  back function.  It is used to validate the addresses on the
 *  bus.  If GPIB addresses 1, 3, 24, or 30 are seen on the bus,
 *  this function returns TRUE.
 */

unsigned short _far _loadds addrfunc (short board,
                                      unsigned short type,
                                      unsigned short addr)
{
   if ((addr == 1) || (addr == 3) ||
       (addr == 24) || (addr == 30)) {
      /*
       * If the device is to be serial polled, then accept
       * the address so that the spollfunc can be called to
       * return the serial poll response byte.
       */
```

GotoMultAddr GotoMultAddr (Continued)

```
      if (type == MultAddrSerialPoll) {
         return (TRUE);
      }
      /*
       * If this is a listen address, then set the global
       * "addressed" to TRUE and store the listen address in
       * the global "address".
       */

      else if (type == MultAddrListen) {
         addressed = TRUE;
         address = (LAD | addr);
         return (TRUE);
      }
      /*
       * If this is a talk address, then set the global
       * "addressed" to TRUE and store the talk address in
       * the global "address".
       */
      else if (type == MultAddrTalk) {
         addressed = TRUE;
         address = (TAD | addr);
         return (TRUE);
      }
   }
   /* Return FALSE since you do not claim this address.
    */
   return (FALSE);

}  /* end of addrfunc */

/*
 * The following function implements the Serial Poll Response
 * call-back function. It always returns the GPIB address of the
 * simulated device as the serial poll response byte.
 */
unsigned short _far _loadds spollfunc (short board,
                                       unsigned short addr)
{
   return (addr|0x40);
}
```

GotoMultAddr GotoMultAddr (Continued)

```
/*
 * The following program is an example of how to simulate multiple
 * GPIB addresses. The program waits in a loop until one of its
 * simulated addresses is present on the bus.  It then reads
 * data or writes data for the simulated device.  If you press
 * any key, the program terminates.
 */
short _cdecl main ( void )

{
   short testing;
   short SimulatedAddress;

   addressed = FALSE;
   testing = TRUE;

   /*
    * Enable multiple primary GPIB addresses for interface
    * board #0. Pass the address of the "address selection"
    * function (addrfunc) and the "serial poll response"
    * function (spollfunc).
    */

   GotoMultAddr(0, MultAddrPrimary, addrfunc, spollfunc);
   if (ibsta & ERR) {
      printf("Error calling GotoMultAddr.\n");
      ibonl(0, 0);
      exit(1);
   }
   /*
    * This is the main loop. Stay here until any key is pressed
    * on the keyboard.
    */

   while (testing) {
      printf("\nWaiting to be addressed....\n");
      /*
       * Check for any key to be pressed.
       */
      while (addressed == FALSE) {
         if (_bios_keybrd(_KEYBRD_READY)) {
            testing = FALSE;
            break;
         }
      }
```

GotoMultAddr GotoMultAddr (Continued)

```
addressed = FALSE;
SimulatedAddress = address;

/*
 * As long as you did not press a key to exit, then the
 * program must be addressed to talk or listen.
 */

if (testing == TRUE) {
   /*
    * If the address is a listen address, then read in
    * data byte for the simulated device.  After reading
    * in the bytes, call GenerateREQT to request service
    * for the simulated device.
    */
   if ((SimulatedAddress & (LAD | TAD)) == LAD) {
      printf("Address %d is listening.\n",
             (SimulatedAddress & ~LAD));

      /*
       * Read a buffer for the given device.
       */
      RcvRespMsg(0, buffer,
            (unsigned long)BUFSIZE, STOPend);
      if (ibsta & ERR) {
         printf("Error from RcvRespMsg.\n");
         ibonl(0, 0);
         exit(1);
      }
      /*
       * Put a NULL byte at the end of the buffer and call
       * printf to output the buffer to the screen.
       */
      buffer[ibcntl] = '\0';
      printf("Received ‘%s’ for PAD %d\n", buffer,
              (SimulatedAddress & ~LAD));
```

GotoMultAddr GotoMultAddr (Continued)

```
            /*
             * Now assert SRQ to request service for
             * the simulated device.
             */
            GenerateREQT(0,(SimulatedAddress & ~LAD));
         }
         /*
          * If the address is a talk address, then output a
          * buffer containing the GPIB address of the simulated
          * device.  Then call GenerateREQF to cancel the service
          * request for the simulated device.
          */
         else if ((SimulatedAddress & (LAD | TAD))
                        == TAD) {
            printf("Address %d talking.\n",
                   (SimulatedAddress & ~TAD));

            sprintf(buffer, "Data from GPIB address %d.",
                    (SimulatedAddress & ~TAD));

            SendDataBytes(0, buffer,
               (unsigned long)strlen(buffer), DABend);
            if (ibsta & ERR) {
               printf("Error from SendDataBytes.\n");
               ibonl(0, 0);
               exit(1);
            }

            GenerateREQF(0,(SimulatedAddress & ~TAD));
         }
         else {
            printf("NOT talk or listen addressed.\n");
            ibonl(0, 0);
            exit(1);
         }
      }
   }
   /*
    * You must call ibonl with a value of 0 before exiting the
    * program.
    */
   ibonl(0, 0);

   return 0;

}  /* end of main */
```

PassControl PassControl

Purpose

Pass control to another device with Controller capability.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLPassControl (int boardID, Addr4882_t address,
              int _far *ibsta,  int _far *iberr,
              long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLPassControl Lib "gpib.dll"
       (ByVal boardID%, ByVal address%, ibsta%, iberr%,
       ibcntl&)
```

PassControl

PassControl
(Continued)

Input

`boardID`	The interface board 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`. That device becomes Controller-In-Charge and `boardID` is no longer CIC.

Possible Errors

EARG	The `address` parameter is invalid (out of range) or `NOADDR`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

PPoll PPoll

Purpose

Perform a parallel poll on the GPIB.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLPPoll (int boardID, short _far *result, int _far *ibsta,
          int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLPPoll Lib "gpib.dll"
       (ByVal boardID%, result%, ibsta%, iberr%, ibcntl&)
```

Input

`boardID`	The interface board number

Output

`result`	The parallel poll result

PPoll

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 on parallel polling, refer to the NI-488.2 user manual.

Possible Errors

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

PPollConfig PPollConfig

Purpose

Configure a device to respond to parallel polls.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

```
CALL PPollConfig (boardID%, address%, dataline%, lineSense%)
```

Windows Format

C

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

Visual Basic

```
CALL PPollConfig (boardID%, address%, dataline%, lineSense%)
```

Direct Entry with C

```
DLLPPollConfig (int boardID, Addr4882_t address, int dataline,
                int lineSense, int _far *ibsta, int _far *iberr,
                long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLPPollConfig Lib "gpib.dll"
    (ByVal boardID%, ByVal address%, ByVal dataline%,
     ByVal lineSense%, ibsta%, iberr%, ibcntl&)
```

PPollConfig

PPollConfig
(Continued)

Input

`boardID`	The interface board 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, then 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 one-bit, device-dependent status messages from up to eight devices simultaneously.

For more information on parallel polling, refer to the NI-488.2 user manual.

Possible Errors

EARG	The `address` parameter is invalid (out of range) or `NOADDR`; `dataline` is not in the range 1 to 8, or `lineSense` is not 0 or 1.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

PPollUnconfig PPollUnconfig

Purpose

Unconfigure devices for parallel polls.

DOS Format

C

```
void PPollUnconfig (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL PPollUnconfig (boardID%, addrlist%(0))
```

Windows Format

C

```
void PPollUnconfig (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLPPollUnconfig (int boardID, Addr4882_t _far addrlist[],
                  int _far *ibsta, int _far *iberr,
                  long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLPPollUnconfig Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%,
        ibcntl&)
```

PPollUnconfig

PPollUnconfig
(Continued)

Input

`boardID`	The interface board 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`, then 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 on parallel polling, refer to the NI-488.2 user manual.

Possible Errors

EARG	An invalid address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

RcvRespMsg RcvRespMsg

Purpose

Read data bytes from a device that is already addressed to talk.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLRcvRespMsg (int boardID, void _far *buffer, long cnt,
               int termination, int _far *ibsta,
               int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLRcvRespMsg Lib "gpib.dll"
       (ByVal boardID%, ByVal buffer$, ByVal cnt&,
        ByVal termination%, ibsta%, iberr%, ibcntl&)
```

RcvRespMsg

Input

`boardID`	The interface board number
`cnt`	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 `cnt` bytes from the GPIB and places these bytes into `buffer`. Data bytes are read until either `cnt` 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 board is already in listen-active state and a device is already addressed to be a Talker (see `ReceiveSetup` or `Receive`).

RcvRespMsg

RcvRespMsg (Continued)

Possible Errors

EABO	The I/O timeout period elapsed before all the bytes were received.
EADR	The interface board 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

ReadStatusByte ReadStatusByte

Purpose

Serial poll a single device.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLReadStatusByte (int boardID, Addr4882_t address,
                   short _far     *result, int _far *ibsta,
                   int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLReadStatusByte Lib "gpib.dll"
       (ByVal boardID%, ByVal address%, result%, ibsta%,
        iberr%, ibcntl&)
```

ReadStatusByte

ReadStatusByte (Continued)

Input

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

Receive Receive

Purpose

Read data bytes from a device.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

```
CALL Receive (boardID%, address%, buffer$, termination%)
```

Windows Format

C

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

Visual Basic

```
CALL Receive (boardID%, address%, buffer$, termination%)
```

Direct Entry with C

```
DLLReceive (int boardID, Addr4882_t address, void _far *buffer,
            long cnt, int termination, int _far *ibsta,
            int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLReceive Lib "gpib.dll"
     (ByVal boardID%, ByVal address%, ByVal buffer$,
     ByVal cnt&, ByVal termination%, ibsta%, iberr%, ibcntl&)
```

Receive

Receive
(Continued)

Input

`boardID`	The interface board number
`address`	Address of a device from which to receive data
`cnt`	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 board to listen. Then up to `cnt` bytes are read and placed into the buffer. Data bytes are read until either `cnt` 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`.

Receive

Receive
(Continued)

Possible Errors

EABO	The I/O timeout period elapsed before all the bytes were received.
EARG	The `address` or `termination` parameter is invalid (out of range), or `address` is `NOADDR`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress

ReceiveSetup ReceiveSetup

Purpose

Address a device to be a Talker and the interface board to be a Listener in preparation for `RcvRespMsg`.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLReceiveSetup (int boardID, Addr4882_t address,
                 int _far *ibsta, int _far *iberr,
                 long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLReceiveSetup Lib "gpib.dll"
       (ByVal boardID%, ByVal address%, ibsta%, iberr%,
        ibcntl&)
```

ReceiveSetup

ReceiveSetup (Continued)

Input

`boardID`	The interface board number
`address`	Address of a device to be talk addressed

Description

`ReceiveSetup` makes the device described by `address` talker-active and makes the interface board listen-active. This call is usually followed by a call to `RcvRespMsg` to transfer data from the device to the interface board. This routine is particularly useful to make multiple calls to `RcvRspMsg`; it eliminates the need to readdress the device to receive every block of data.

Possible Errors

EARG	The `address` parameter is invalid (out of range) or `NOADDR`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

ResetSys ResetSys

Purpose

Reset and initialize IEEE 488.2-compliant devices.

DOS Format

C

```
void ResetSys (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL ResetSys (boardID%, addrlist%(0))
```

Windows Format

C

```
void ResetSys (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLResetSys (int boardID, Addr4882_t _far addrlist[],
             int _far *ibsta, int _far *iberr,
             long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLResetSys Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%,
        ibcntl&)
```

ResetSys

Input

`boardID`	The interface board 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	An invalid address (out of range) appears in `addrlist` (`ibcntl` is the index of the invalid address in the `addrlist` array), or the `addrlist` is empty.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners are on the GPIB.
EOIP	Asynchronous I/O is in progress.
ESAC	Board is not System Controller.

Send Send

Purpose

Send data bytes to a device.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

```
CALL Send (boardID%, address%, buffer$, eotmode%)
```

Windows Format

C

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

Visual Basic

```
CALL Send (boardID%, address%, buffer$, eotmode%)
```

Direct Entry with C

```
DLLSend (int boardID, Addr4882_t address, void _far *buffer,
         long datacnt, int eotmode, int _far *ibsta,
         int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSend Lib "gpib.dll"
       (ByVal boardID%, ByVal address%, ByVal buffer$,
        ByVal datacnt&, ByVal eotmode%, ibsta%, iberr%,
       ibcntl&)
```

Send

Send
(Continued)

Input

`boardID`	The interface board number
`address`	Address of a device to which data is sent
`buffer`	The data bytes to be sent
`datacnt`	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 board to talk. Then `datacnt` bytes from `buffer` are sent to the device. The last byte is sent with the EOI line asserted if `eotmode` is `DABend`. The last byte is sent *without* the EOI line asserted if `eotmode` is `NULLend`. If `eotmode` is `NLend` then a new line character (`'\n'`) is sent with the EOI line asserted after the last byte of `buffer`. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Send

Send
(Continued)

Possible Errors

EABO	The I/O timeout period has expired before all of the bytes were sent.
EARG	The `address` parameter is invalid (out of range or the constant `NOADDR`), or the `buffer` is empty and the `eotmode` is `DABend`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board 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 SendCmds

Purpose

Send GPIB command bytes.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLSendCmds (int boardID, void _far *buffer, long cnt,
             int _far *ibsta, int _far *iberr,
             long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSendCmds Lib "gpib.dll"
        (ByVal boardID%, ByVal buffer$, ByVal cnt&, ibsta%,
          iberr%, ibcntl&)
```

SendCmds

SendCmds
(Continued)

Input

`boardID`	The interface board number
`buffer`	Command bytes to be sent
`cnt`	Number of bytes to be sent

Description

`SendCmds` sends `cnt` 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No devices are connected to the GPIB.
EOIP	Asynchronous I/O is in progress.

SendDataBytes SendDataBytes

Purpose

Send data bytes to devices that are already addressed to listen.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

```
CALL SendDataBytes (boardID%, buffer$, eotmode%)
```

Windows Format

C

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

Visual Basic

```
CALL SendDataBytes (boardID%, buffer$, eotmode%)
```

Direct Entry with C

```
DLLSendDataBytes (int boardID, void _far *buffer, long datacnt,
                  int eotmode, int _far *ibsta, int _far *iberr,
                  long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSendDataBytes Lib "gpib.dll"
       (ByVal boardID%, ByVal buffer$, ByVal datacnt&,
        ByVal eotmode%, ibsta%, iberr%, ibcntl&)
```

SendDataBytes SendDataBytes (Continued)

Input

`boardID`	The interface board number
`buffer`	The data bytes to be sent
`datacnt`	Number of bytes to be sent
`eotmode`	The data termination mode: `DABend`, `NULLend`, or `NLend`

Description

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

SendDataBytes

SendDataBytes (Continued)

Possible Errors

EABO	The I/O timeout period expired before all of the bytes were sent.
EADR	Interface boardID is not talk-active; use `SendSetup` to address the GPIB properly.
EARG	The `eotmode` parameter is invalid (it can be only `DABend`, `NULLend`, or `NLend`), or the `buffer` is empty and the `eotmode` is `DABend`.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board 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 SendIFC

Purpose

Reset the GPIB by sending interface clear.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLSendIFC (int boardID, int _far *ibsta, int _far *iberr,
           long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSendIFC Lib "gpib.dll"
       (ByVal boardID%, ibsta%, iberr%, ibcntl&)
```

Input

`boardID`	The interface board number

SendIFC

Description

`SendIFC` is used as part of GPIB initialization. It forces the interface board 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 (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface board is not configured as the System Controller; see `ibrsc`.

SendList SendList

Purpose

Send data bytes to multiple GPIB devices.

DOS Format

C

```
void SendList(int boardID, Addr4882_t addrlist[], void *buffer,
              long datacnt, int eotmode)
```

QuickBASIC/BASIC

```
CALL SendList (boardID%, addrlist%(), buffer$, eotmode%)
```

BASICA

```
CALL SendList (boardID%, addrlist%(0), buffer$, eotmode%)
```

Windows Format

C

```
void SendList (int boardID, Addr4882_t addrlist[],
               void *buffer, long datacnt, int eotmode)
```

Visual Basic

```
CALL SendList (boardID%, addrlist%(), buffer$, eotmode%)
```

Direct Entry with C

```
DLLSendList (int boardID, Addr4882_t _far addrlist[],
             void _far *buffer, long datacnt, int eotmode,
             int _far *ibsta, int _far *iberr,
             long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSendList Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ByVal buffer$,
        ByVal datacnt&, ByVal eotmode%, ibsta%, iberr%, ibcntl&)
```

SendList

Input

`boardID`	The interface board number
`addrlist`	A list of device addresses to send data to
`buffer`	The data bytes to be sent
`datacnt`	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 board to talk. Then, `datacnt` 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`, 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`.

SendList

Possible Errors

EABO	The I/O timeout period expired before all of the bytes were sent.
EARG	An invalid address (out of range) appears in `addrlist` (`ibcntl` is the index of the invalid address in the `addrlist` array), the `eotmode` parameter is invalid (`eotmode` can be only `DABend`, `NULLend`, or `NLend`), or the `buffer` is empty and the `eotmode` is `DABend`.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

SendLLO SendLLO

Purpose

Send the Local Lockout (LLO) message to all devices.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLSendLLO (int boardID, int _far *ibsta, int _far *iberr,
            long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSendLLO Lib "gpib.dll"
       (ByVal boardID%, ibsta%, iberr%, ibcntl&)
```

Input

`boardID`	The interface board number

SendLLO

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 most cases, use `SetRWLS` to place devices in Remote With Lockout State.

Possible Errors

EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface board is not configured as System Controller.

SendSetup SendSetup

Purpose

Set up devices to receive data in preparation for `SendDataBytes`.

DOS Format

C

```
void SendSetup (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL SendSetup (boardID%, addrlist%(0))
```

Windows Format

C

```
void SendSetup (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLSendSetup (int boardID, Addr4882_t _far addrlist[],
              int _far *ibsta, int _far *iberr,
              long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSendSetup Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%,
        ibcntl&)
```

SendSetup

SendSetup
(Continued)

Input

`boardID`	The interface board 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 board talk-active. This call is usually followed by `SendDataBytes` to actually transfer data from the interface board 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	The `addrlist` is empty, or an invalid address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

SetRWLS SetRWLS

Purpose

Place devices in Remote With Lockout State.

DOS Format

C

```
void SetRWLS (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL SetRWLS (boardID%, addrlist%(0))
```

Windows Format

C

```
void SetRWLS (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLSetRWLS (int boardID, Addr4882_t _far addrlist[],
           int _far *ibsta, int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLSetRWLS Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%, ibcntl&)
```

SetRWLS

Input

`boardID`	The interface board number
`addrlist`	A list of device addresses 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. To release the Local Lockout, use the EnableLocal NI-488.2 routine.

Possible Errors

EARG	An invalid address (out of range) appears in `addrlist` (`ibcntl` is the index of the invalid address in the `addrlist` array), or the `addrlist` is empty.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESAC	The interface board is not configured as System Controller.

TestSRQ TestSRQ

Purpose

Determine the current state of the GPIB Service Request (SRQ) line.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLTestSRQ (int boardID, short _far *result, int _far *ibsta,
           int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLTestSRQ Lib "gpib.dll"
       (ByVal boardID%, result%, ibsta%, iberr%, ibcntl&)
```

TestSRQ

TestSRQ
(Continued)

Input

`boardID` The interface board 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, then `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 (out of range) or the NI-488.2 driver is not installed.

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

TestSys TestSys

Purpose

Cause IEEE 488.2-compliant devices to conduct self-tests.

DOS Format

C

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

QuickBASIC/BASIC

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

BASICA

```
CALL TestSys (boardID%, addrlist%(0), resultlist%(0))
```

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLTestSys (int boardID, Addr4882_t _far addrlist[],
           short _far resultlist[], int _far *ibsta,
           int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLTestSys Lib "gpib.dll"
       (ByVal boardID%, addrlist%, resultlist%, ibsta%, iberr%,
        ibcntl&)
```

TestSys

TestSys
(Continued)

Input

`boardID`	The interface board 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?\n`" message to the IEEE 488.2-compliant devices described by `addrlist`. The "`*TST?\n`" 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. Any other value indicates that the device failed its self-test. Refer to the manual that came with your device to determine the meaning of the failure code. A test result of -1 indicates that the I/O timeout period elapsed before the device sent its result code. `ibcntl` contains the number of devices that failed.

TestSys

TestSys
(Continued)

Possible Errors

EABO	The interface board timed out before receiving a result from a device; `ibcntl` contains the index of the first device that timed out. -1 is stored as the test result for the timed-out device.
EARG	An invalid address (out of range) appears in `addrlist` (`ibcntl` is the index of the invalid address in the `addrlist` array), or the `addrlist` is empty.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners are on the GPIB.
EOIP	Asynchronous I/O is in progress.

Trigger Trigger

Purpose

Trigger a device.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLTrigger (int boardID, Addr4882_t address, int _far *ibsta,
           int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLTrigger Lib "gpib.dll"
       (ByVal boardID%, ByVal address%, ibsta%, iberr%,
        ibcntl&)
```

Trigger

Trigger
(Continued)

Input

`boardID`	The interface board 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 Group Execute Trigger message is sent to all devices that are currently listen-active on the GPIB.

Possible Errors

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

TriggerList TriggerList

Purpose

Trigger multiple devices.

DOS Format

C

```
void TriggerList (int boardID, Addr4882_t addrlist[])
```

QuickBASIC/BASIC

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

BASICA

```
CALL TriggerList (boardID%, addrlist%(0))
```

Windows Format

C

```
void TriggerList (int boardID, Addr4882_t addrlist[])
```

Visual Basic

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

Direct Entry with C

```
DLLTriggerList (int boardID, Addr4882_t _far addrlist[],
                int _far *ibsta, int _far *iberr,
                long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLTriggerList Lib "gpib.dll"
       (ByVal boardID%, addrlist%, ibsta%, iberr%,
        ibcntl&)
```

TriggerList

TriggerList
(Continued)

Input

`boardID`	The interface board number
`addrlist`	A list of device addresses terminated by `NOADDR`

Description

`TriggerList` sends the Group Execute Trigger (GET) GPIB message to the devices included in `addrlist`. If `addrlist` contains only `NOADDR`, the Group Execute Trigger message is sent to all devices that are currently listen-active on the GPIB.

Possible Errors

EARG	An invalid address (out of range) 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 board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `boardID` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

WaitSRQ WaitSRQ

Purpose

Wait until a device asserts the GPIB Service Request (SRQ) line.

DOS Format

C

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

BASICA/QuickBASIC/BASIC

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

Windows Format

C

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

Visual Basic

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

Direct Entry with C

```
DLLWaitSRQ (int boardID, short _far *result, int _far *ibsta,
           int _far *iberr, long _far *ibcntl)
```

Direct Entry with Visual Basic

```
Declare Sub DLLWaitSRQ Lib "gpib.dll"
       (ByVal boardID%, result%, ibsta%, iberr%, ibcntl&)
```

WaitSRQ

Input

`boardID` The interface board number

Output

`result` State of the SRQ line: non-zero if line is asserted, zero if line 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 value 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 (out of range) or the NI-488.2 driver is not installed.

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

Appendix A
Multiline Interface Messages

This appendix contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions. These multiline interface messages are sent and received with ATN TRUE.

For more information on these messages, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*.

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US	CFE	3F	077	63	?	UNL

Message Definitions

CFE†	Configuration Enable
CFG†	Configure
DCL	Device Clear
GET	Group Execute Trigger
GTL	Go To Local
LLO	Local Lockout
MLA	My Listen Address
MSA	My Secondary Address
MTA	My Talk Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable

†This multiline interface message is a proposed extension to the IEEE 488.1 specification to support the HS488 high-speed protocol.

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE,CFG1
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE, CFG2
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE,CFG3
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE,CFG4
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE,CFG5
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE,CFG6
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE,CFG7
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE,CFG8
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE,CFG9
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE,CFG10
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE,CFG11
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE,CFG12
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE,CFG13
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE,CFG14
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE,CFG15
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

PPE	Parallel Poll Enable	SPE	Serial Poll Enable
PPU	Parallel Poll Unconfigure	TCT	Take Control
SDC	Selected Device Clear	UNL	Unlisten
SPD	Serial Poll Disable	UNT	Untalk

Appendix B
Status Word Conditions

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

For information about how to use `ibsta` in your application program, refer to Chapter 3, *Developing Your Application*, in the NI-488.2 user manual.

If a function call returns an ENEB or EDVR error, all status word bits except the ERR bit are cleared, indicating that it is not possible to obtain the status of the GPIB board.

Each bit in `ibsta` can be set for NI-488 device calls (dev), NI-488 board calls and NI-488.2 calls (brd), or both (dev, brd).

The following table lists the status word bits.

Table B-1. Status Word Bits

Mnemonic	Bit Pos.	Hex Value	Type	Description
ERR	15	8000	dev, brd	GPIB 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
SPOLL	10	400	brd	Board has been serial polled by Controller
EVENT	9	200	brd	DCAS, DTAS, or IFC event has occurred
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 been exceeded. TIMO is set in the status word following an `ibwait` call if the TIMO bit of the `ibwait` mask parameter is set and the time limit expires. TIMO is also set following any synchronous I/O functions (for example, `ibcmd`, `ibrd`, `ibwrt`, `Receive`, `Send`, and `SendCmds`) if a timeout occurs during one of these calls. TIMO is cleared in all other circumstances.

END (dev, brd)

END indicates that either 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 board 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 board is CIC, the GPIB SRQ line is asserted, and the automatic serial poll capability is disabled. SRQI is cleared either when the GPIB board 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. It indicates that one or more serial poll response bytes are waiting in the device's serial poll response queue. Automatic serial poll responses are not stored in the response queue unless they have bit 6 set.

An automatic serial poll occurs either as a result of a call to `ibwait`, or automatically, if automatic serial polling is enabled. If the serial poll response queue is not empty, `ibrsp` returns the oldest byte stored in the queue. To empty the response queue, call `ibrsp` repeatedly until RQS is no longer set in the device's status word.

SPOLL (brd)

Use SPOLL in Talker/Listener applications (applications in which the GPIB interface is not the Controller) to determine when the Controller has serial polled the GPIB board. The SPOLL bit is disabled by default. Use the `ibconfig` function (option `IbcSPollBit`) to enable it. When the SPOLL bit is enabled, it is set after the board has been serial polled. SPOLL is cleared on any call immediately after an `ibwait` call, if the SPOLL bit was set in the wait mask, or immediately following a call to `ibrsv`.

EVENT (brd)

Use EVENT in Talker/Listener applications to monitor the order of GPIB device clear, group execute trigger, and send interface clear commands. The usual DCAS and DTAS bits of `ibsta` might be insufficient.

The EVENT bit is disabled by default. If you want to use this bit, you must use the `ibconfig` function (option `IbcEventQueue`) to enable it. When you enable this bit, the DCAS and DTAS bits are disabled. When an event occurs, the EVENT bit is set and any I/O in progress is aborted. The application can then call the `ibevent` function to determine which event occurred.

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 an I/O operation is in progress.

LOK (brd)

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

REM (brd)

REM indicates whether or not the board is in the remote state. REM is set whenever the Remote Enable (REN) GPIB line is asserted and the GPIB board detects that its listen address has been sent either by the GPIB board or by another Controller. REM is cleared in the following situations:

- When REN becomes unasserted
- When the GPIB board as a Listener detects that the Go to Local (GTL) command has been sent either by the GPIB board 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 board is the Controller-In-Charge. CIC is set when the `SendIFC` routine or `ibsic` function is executed while the GPIB board is System Controller or when another Controller passes control to the GPIB board. CIC is cleared whenever the GPIB board detects Interface Clear (IFC) from the System Controller, or when the GPIB board 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 board is addressed as a Talker. TACS is set whenever the GPIB board detects that its talk address (and secondary address, if enabled) has been sent either by the GPIB board itself or by another Controller. TACS is cleared whenever the GPIB board 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 board is addressed as a Listener. LACS is set whenever the GPIB board detects that its listen address (and secondary address, if enabled) has been sent either by the GPIB board itself or by another Controller. LACS is also set whenever the GPIB board shadow handshakes as a result of the `ibgts` function. LACS is cleared whenever the GPIB board 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 board has detected a device trigger command. DTAS is set whenever the GPIB board, 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 board has detected a device clear command. DCAS is set whenever the GPIB board detects that the Device Clear (DCL) command has been sent by another Controller, or whenever the GPIB board as a Listener detects that the Selected Device Clear (SDC) command has been sent by another Controller. DCAS is cleared on any call immediately following an `ibwait` call, if the DCAS bit was set in the `ibwait` mask parameter. It also clears on any call immediately following a read or write.

Appendix C
Error Codes and Solutions

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

The following table lists the GPIB error codes.

Table C-1. GPIB Error Codes

Error Mnemonic	iberr Value	Meaning
EDVR	0	Operating system error
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	No Listeners on the GPIB
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
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)
NI-488.2 for DOS

EDVR is returned when the board or device name passed to `ibfind` is not configured in the software. In this case, the variable `ibcntl` contains the system error code 2, *Device Not Found* or 110, *Open failed.* EDVR is also returned when an invalid unit descriptor is passed to any function call. In this case, the variable `ibcntl` contains the system error code 6, *Invalid handle*. EDVR is also returned when the driver (`gpib.com`) is not installed.

Solutions

- Use `ibdev` to open a device without specifying its symbolic name.
- Use only device or board names that are configured in the utility program `ibconf` as parameters to the `ibfind` function.
- Use the unit descriptor returned from the `ibfind` function as the first parameter in subsequent NI-488 functions. Examine the variable after the `ibfind` and before the failing function to make sure it was not corrupted.
- Make sure the NI-488.2 driver is installed by checking the `config.sys` file in the root directory. Make sure it contains the following line:

  `device=`*`drive`*`:\`*`path`*`\gpib.com`

  where *`drive`* is the drive (usually `c`) and *`path`* is the directory (for example, `at-gpib`).

EDVR (0)
NI-488.2 for Windows

EDVR is returned in the following cases:

- The board or device name passed to `ibfind` is not configured in the software. In this case, the variable `ibcntl` contains the DOS error code 2, *Device Not Found.*
- An invalid unit descriptor is passed to any function call. In this case, the variable `ibcntl` contains the DOS error code 6, *Invalid handle*.
- The driver (`gpib.dll`) is not installed.
- The driver configuration file `gpib.ini` is not located in the windows directory. In this case, the variable `ibcntl` contains the value -1.
- The driver file `gpib.ini` is in the windows directory but not compatible with the driver file `gpib.dll` that you are using. In this case, the variable `ibcntl` contains a negative value other than -1.

Solutions

- Use `ibdev` to open a device without specifying its symbolic name.
- Use only device or board names that are configured in the GPIB software configuration utility as parameters to the `ibfind` function.

- Use the unit descriptor returned from `ibfind` as the first parameter in subsequent NI-488 functions. Examine the variable before the failing function to make sure the function has not been corrupted.
- Make sure the NI-488.2 driver is installed by checking that `gpib.dll` and `gpib.ini` are in the windows directory (usually `c:\windows`).

ECIC (1)

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

- Any device-level NI-488 functions that affect the GPIB
- Any board-level NI-488 functions that issue GPIB command bytes such as `ibcmd`, `ibcmda`, `ibln`, `ibrpp`
- `ibcac`, `ibgts`
- Any of the NI-488.2 routines that issue GPIB command bytes such as `SendCmds`, `PPoll`, `Send`, `Receive`

Solutions

- Use `ibsic` or `SendIFC` to make the GPIB board become Controller-In-Charge on the GPIB.
- Use `ibrsc 1` to make sure your GPIB board 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 board.

ENOL (2)

ENOL usually occurs when a write operation is attempted with no Listeners addressed. For a device write, this error 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 also occur in situations in which the GPIB board is not the CIC and the Controller asserts ATN before the write call in progress has ended.

Solutions

- Make sure that the GPIB address of your device matches the GPIB address of the device to which you want to write data.
- If you are not using device-level calls, make sure that your device is properly addressed to listen before writing to it by using `ibcmd` or `SendSetup`.
- 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.
- If you are using device-level calls, call `ibpad` (and `ibsad`, if necessary) to match the configured address to the device switch settings.
- Reduce the write byte count to that which is expected by the Controller.

EADR (3)

EADR occurs when the GPIB board 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

- Make sure that the GPIB board is addressed correctly before calling NI-488 board-level `ibrd` or `ibwrt`, and NI-488.2 routines `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
- `ibpad` or `ibsad` called with primary or secondary addresses
- `ibppc` called with invalid parallel poll configurations

- A board-level NI-488 call made with a valid device descriptor, or a device-level NI-488 call made with a valid board descriptor
- An NI-488.2 routine called with an invalid address parameter
- `PPollConfig` called with an invalid data line or sense bit

Solutions

- Make sure that the parameters passed to the NI-488 function or NI-488.2 routine are valid.
- Do not use a device descriptor in a board function or vice-versa.

ESAC (5)

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

Solutions

Give the GPIB board System Controller capability by calling `ibrsc 1`. You can also configure that capability into the software using `ibconf` in DOS or the GPIB software configuration utility in Windows.

EABO (6)

EABO indicates that an I/O operation has been canceled, usually due to a timeout condition. Other causes for this error 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

- 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 board exists at the I/O address specified in the configuration program. This happens when the board is not physically plugged into the system, the I/O address specified during configuration does not match the actual board setting, or there is a system conflict with the base I/O address, or the `Use This Interface` field is set incorrectly in `ibconf` in DOS or the GPIB software configuration utility in Windows.

Solutions

- Make sure there is a GPIB board in your computer that is properly configured both in hardware and software at a free base I/O address.
- Make sure that the `Use This Interface` field is set to `Yes` in `ibconf` or the GPIB software configuration utility.

EDMA (8)
NI-488.2 for Windows

EDMA occurs when an error occurs using DMA for data transfers. If your computer has more than 16 MB of RAM and you do not have the National Instruments virtual GPIB device (`nivgpibd.386`) installed, the NI-488.2 software returns EDMA if you are using DMA and the data buffer is located in memory above 16 MB.

If you are using Windows 3.0, you are using DMA for data transfers, and you do not have the National Instruments virtual DMA device (`nivdmad.386`) installed, the NI-488.2 software returns EDMA if you try to use DMA to transfer data.

Solutions

- Install the appropriate virtual device in the `system.ini` file in the Windows directory in the [`386Enh`] section. The following line installs the virtual GPIB device:

  `device =` *`drive`*`:\`*`path`*`\nivgpibd.386`

  where *`drive`* and *`path`* describe the location of `nivgpibd.386` on your hard drive.

- By default, only one GPIB board at a time can perform DMA. If you need to perform DMA transfers on multiple GPIB boards at the same time, add a new section to your `system.ini` file, [`vgpibd`]. In this section add the option `NumBoardsUsingDMA`, and set it equal to the number of boards that will be performing DMA. For example, if you want two boards to perform DMA concurrently, add the following lines to the bottom of your `system.ini` file:

```
[vgpibd]
NumBoardsUsingDMA=2
```

- To install the virtual DMA device, first change the default virtual DMA device line to a remark line by adding a semicolon. Then add a line to install the National Instruments virtual DMA device as follows:

```
;device = *vdmad
device = drive:\path\nivdmad.386
```

  where *drive* and *path* describe the location of `nivdmad.386` on your hard drive.

Note: ***You must restart Windows after you modify the*** `system.ini` ***file.***

- Alternatively, you can correct the EDMA problem by disabling DMA in the software. You can use `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`, `ibwait`, and `ibonl` or perform other non-GPIB operations. Once the asynchronous I/O has begun, GPIB calls other than `ibstop`, `ibwait`, or `ibonl` are strictly limited. If a call might interfere with the I/O operation in progress, the driver returns EOIP.

Solutions

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

- `ibwait` If the returned `ibsta` contains CMPL then 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 board 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 board 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 DOS error code for this condition is contained in `ibcntl`.

Solutions

- 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 NI-488 device-level 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

- 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

- Call `ibrsp` more frequently to empty the queue.
- Disable autopolling with the `ibconfig` function, the `ibconf` utility in DOS, or the GPIB software configuration utility in Windows.

ESRQ (16)

ESRQ occurs only during the `ibwait` function. 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 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`, `FindRQS`, and `ibevent` 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.
- In the case of `ibevent`, ETAB means the event queue overflowed and event information was lost.

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. In the case of ETAB returned from `ibevent`, call `ibevent` more often to empty the queue.

Appendix D
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Corporate Headquarters: (512) 795-8248
Technical Support Fax: (512) 794-5678

Branch Offices	**Phone Number**	**Fax Number**
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	90 527 2321	90 502 2930
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___) _____ Phone (___) _____

Computer brand _____

Model _____ Processor _____

Operating system _____

Speed _____MHz RAM _____MB

Display adapter _____

Mouse _____yes _____no

Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____

Revision _____

Configuration _____

National Instruments software product _____

Revision _____

Configuration _____

(continues)

The problem is _____

List any error messages _____

The following steps will reproduce the problem _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **NI-488.2™ Function Reference Manual for DOS/Windows**

Edition Date: **August 1996**

Part Number: **370903A-01**

Please comment on the completeness, clarity, and organization of the manual.

(continues)

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

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

A

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

access board — The GPIB board that controls and communicates with the devices on the bus that are attached to it.

ANSI — American National Standards Institute.

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 is 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	See *Controller-In-Charge*.
Controller-In-Charge (CIC)	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 is the GPIB command used to reset the device or internal functions of all devices. See *SDC*.
Device Clear	See DCL.
device-level function	A function that combines several rudimentary board operations into one function so that the user does not have to be concerned with bus management or other GPIB protocol matters.
DIO1 through DIO8	The GPIB lines that are used to transmit command or data bytes from one device to another.
DLL	Dynamic link library.
DMA (direct memory access)	High-speed data transfer between the GPIB board and memory that is not handled directly by the CPU. Not available on some systems. 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 is the GPIB command used to trigger a device or internal function of an addressed Listener.
Go To Local	See *GTL*.
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 488.2-1987.
GPIB address	The address of a device on the GPIB, composed of a primary address (MLA and MTA) and an optional secondary address (MSA). The GPIB board has both a GPIB address and an I/O address.
GPIB board	Refers to the National Instruments family of GPIB interface boards.
Group Executed Trigger	See *GET*.
GTL	Go To Local is 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.
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 function, this global variable contains the actual number of bytes transmitted.
`ibconf`	The NI-488.2 driver configuration program for DOS.
`iberr`	A global variable that contains the specific error code associated with a function call that failed.
`ibic`	The Interface Bus Interactive Control program for DOS is used to communicate with GPIB devices, troubleshoot problems, and develop your application.
`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 the context of this manual, the transmission of commands or messages between the computer via the GPIB board and other devices on the GPIB.

I/O address	The address of the GPIB board from the point of view of the CPU, as opposed to the GPIB address of the GPIB board. Also called port address or board address.
`ist`	An Individual Status bit of the status byte used in the Parallel Poll Configure function.

K

KB	Kilobytes.

L

LAD (Listen Address)	See *MLA*.
language interface	Code that enables an application program that uses NI-488 functions or NI-488.2 routines to access the driver.
listen address	See *MLA*.
Listener	A GPIB device that receives data messages from a Talker.
low-level function	See *board-level function*.

M

m	Meters.
MAV	The Message Available bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.
MB	Megabytes of memory.
memory-resident	Resident in RAM.

MLA (My Listen Address)	A GPIB command used to address a device to be a Listener. It can be any one of the 31 primary addresses.
MSA (My Secondary Address)	My Secondary Address is the GPIB command used to address a device to be a Listener or a Talker when extended (two byte) addressing is used. The complete address is a MLA or MTA address followed by an MSA address. There are 31 secondary addresses for a total of 961 distinct listen or talk addresses for devices.
MTA (My Talk Address)	A GPIB command used to address a device to be a Talker. It can be any one of the 31 primary addresses.
multitasking	The concurrent processing of more than one program or task.

N

NDAC (Not Data Accepted)	One of the three GPIB handshake lines. 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)	Parallel Poll Configure is the GPIB command used to configure an addressed Listener to participate in polls.
PPD (Parallel Poll Disable)	Parallel Poll Disable is the GPIB command used to disable a configured device from participating in polls. There are 16 PPD commands.
PPE (Parallel Poll Enable)	Parallel Poll Enable is the GPIB command used to enable a configured device to participate in polls and to assign a DIO response line. There are 16 PPE commands.

PPU (Parallel Poll Unconfigure)	Parallel Poll Unconfigure is the GPIB command used to disable any device from participating in polls.
programmed I/O	Low-speed data transfer between the GPIB board and memory in which the CPU moves each data byte according to program instructions. See *DMA*.

R

RAM	Random-access memory.
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 is the GPIB command used to reset internal or device functions of an addressed Listener. See *DCL* and *IFC*.
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)	Serial Poll Disable is the GPIB command used to cancel an SPE command.

SPE (Serial Poll Enable)	Serial Poll Enable is 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 is 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 board or other GPIB device that is the object of the function.
UNL	Unlisten is the GPIB command used to unaddress any active Listeners.
UNT	Untalk is the GPIB command used to unaddress an active Talker.

Index

A

C

D

E

F

G

I

L

M

N

O

P

R

S

T

W