NI-488.2M™
Function Reference Manual for OS/2

April 1994 Edition

Part Number 370951A-01

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20,
Canada (Ontario) (519) 622-9310, Canada (Québec) (514) 694-8521,
Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921,
Netherlands 03480-33466, Norway 32-848400, Spain (91) 640 0085,
Sweden 08-730 49 70, Switzerland 056/20 51 51, U.K. 0635 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, the NI-488.2 routines, and the Application Program Interface (API) calls that comprise the NI-488.2M software for OS/2. The NI-488.2M software is meant to be used with OS/2 (IBM Operating System/2) version 2.0 or higher. This manual assumes that you are already familiar with the OS/2 system.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *NI-488 Functions,* lists the available NI-488 functions, then describes the purpose, format, input and output parameters, and possible errors for each function.
- Chapter 2, *NI-488.2 Routines,* lists the available NI-488 routines, then describes the purpose, format, input and output parameters, and possible errors for each routine.
- Chapter 3, *API-Application Program Interface Functions*, describes the OS/2 Application Program Interface (API) functions.
- Appendix A, *Multiline Interface Messages*, contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions.
- 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.

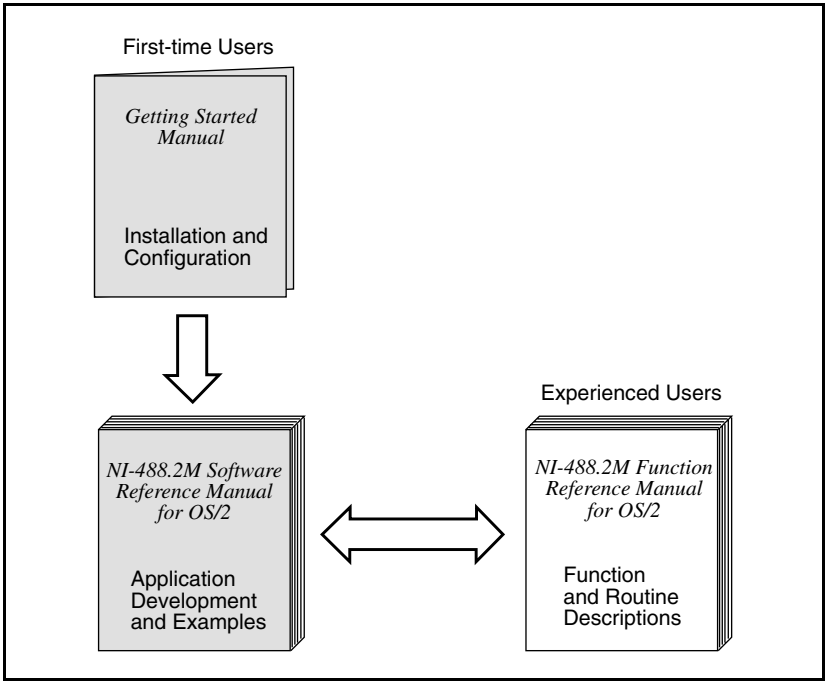
Conventions Used in This Manual

The following conventions are used in this manual:

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Lowercase 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>.
-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys–for example, <Control-Alt-Delete>.
<Control>	Key names are capitalized.
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 488.2-1987, respectively, which define the GPIB.
NI-488.2M software	The term *NI-488.2M software* is used throughout this manual to refer to the NI-488.2M software for OS/2 unless otherwise noted.

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

How to Use This Manual Set



Use the Getting Started Manual to install and configure your GPIB hardware and NI-488.2M software for OS/2.

Use the Software Reference Manual if you want to learn the basics of GPIB and how to develop an application program. The Software Reference Manual also contains debugging information and detailed examples.

Use the Function Reference Manual for specific information about each NI-488 function and NI-488.2 routine such as format, parameters, and possible errors.

Related Documentation

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

- *OS/2 Technical Library, Application Design Guide*
- *OS/2 Technical Library, Programming Guide Volume I*
- *OS/2 Technical Library, Programming Guide Volume II*
- *OS/2 Technical Library, Programming Guide Volume III*
- *OS/2 Technical Library, Control Program Programming Reference*
- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation.*
- ANSI/IEEE Standard 488.2-1987, *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, then describes the purpose, format, input and output parameters, and possible errors for each function.

While using the functions, you might find it helpful to refer to Chapter 2, *Application Examples*, Chapter 3, *Developing Your Application*, and Chapter 6, *GPIB Programming Techniques*, in the *NI-488.2M Software Reference Manual for OS/2*.

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.

Format

The format of the function call is given.

Parameters

The input and output parameters for each function are listed. IN designates an input parameter and OUT designates an output parameter. 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.

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
ibfind	Open a user-configured device
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
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

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
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

(continues)

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

Function	Purpose
ibpad	Change the primary address
ibppc	Parallel poll configure
ibrd	Read data from a device into a user buffer
ibrda	Read data asynchronously from a device into a user buffer
ibrdf	Read data from a device into a file
ibrpp	Conduct a parallel poll
ibrsc	Request or release system control
ibrsv	Request service and change the serial poll status byte
ibsad	Change or disable the secondary address
ibsic	Assert interface clear
ibsre	Set or clear the Remote Enable (REN) line
ibsrq	Request an SRQ *interrupt routine*
ibstop	Abort asynchronous I/O operation
ibtmo	Change or disable the I/O timeout period
ibwait	Wait for GPIB events
ibwrt	Write data to a device from a user buffer
ibwrta	Write data asynchronously to a device from a user buffer
ibwrtf	Write data to a device from a file

IBASK

Board Level
Device Level

IBASK

Purpose

Return information about software configuration parameters.

Format

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

Parameters

IN `ud`	Board or device unit descriptor
IN `option`	Selects the configuration item whose value is being returned
OUT `value`	Current value of the selected configuration item
Function Return	The value of `ibsta`

Description

This function 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 pointed to by `value`.

Possible Errors

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

IBASK

Board Level
Device Level

IBASK
(Continued)

Table 1-3 lists the constants and hexadecimal values of options you can use with `ibask` when `ud` is a board descriptor or a board index. The following options in alphabetical order can be used when `ud` is a board descriptor or a board index.

Constants	Values	Constants	Values
• IbaAUTOPOLL	0x0007	• IbaPAD	0x0001
• IbaBaseAddr	0x0201	• IbaPP2	0x0010
• IbaCICPROT	0x0008	• IbaPPC	0x0005
• IbaDMA	0x0012	• IbaPPollTime	0x0019
• IbaDmaChannel	0x0202	• IbaReadAdjust	0x0013
• IbaEndBitIsNormal	0x001A	• IbaSAD	0x0002
• IbaEOSchar	0x000F	• IbaSC	0x000A
• IbaEOScmp	0x000E	• IbaSendLLO	0x0017
• IbaEOSrd	0x000C	• IbaSRE	0x000B
• IbaEOSwrt	0x000D	• IbaTIMING	0x0011
• IbaEOT	0x0004	• IbaTMO	0x0003
• IbaIRQ	0x0009	• IbaWriteAdjust	0x0014
• IbaIrqLevel	0x0203		

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. See the *Using Serial Polling* section in the *NI-488.2M Software Reference Manual for OS/2.*

(continues)

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

Options (Constants)	Options (Values)	Returned Information
IbaCICPROT	0x0008	zero = The CIC protocol is disabled. non-zero = The CIC protocol is enabled. See the *Device-Level Calls and Bus Management* section in the *NI-488.2M Software Reference Manual for OS/2.*
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`.
IbaSRE	0x000B	zero = The board will not automatically assert the GPIB REN line when it becomes the System Controller. non-zero = The board will automatically assert REN when it becomes the system controller. See `ibrsc` and `ibsre`.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
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. See `ibeos`.
IbaEOScmp	0x000E	zero = A 7-bit compare is used for all EOS comparisons. non-zero = An 8-bit compare is be used for all EOS comparisons. See `ibeos`.
IbaEOSchar	0x000F	The current EOS character of the board. See `ibeos`.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
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). See the *Using Parallel Polling* section in the *NI-488.2M Software Reference Manual for OS/2.*
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 will not have pairs of bytes swapped. 1 = Read operations will have each pair of bytes swapped.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
IbaWriteAdjust	0x0014	0 = Write operations will not have pairs of bytes swapped. 1 = Write operations will have each pair of bytes swapped.
IbaSendLLO	0x0017	zero = The GPIB LLO command will not be sent when a device is put online (`ibfind` or `ibdev`). non-zero = The LLO command will be sent.
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.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
IbaBaseAddr	0x0201	The base I/O address of the board.
IbaDmaChannel	0x0202	The DMA channel that the board is configured to use. If the board is not configured to use DMA, the error ECAP is returned.
IbaIrqLevel	0x0203	The interrupt level that the board is configured to use. If the board is not configured to use interrupts, the error ECAP is returned.

IBASK

Board Level
Device Level

IBASK
(Continued)

Table 1-4 lists the constants and hexadecimal values of options you can use with `ibask` when `ud` is a device descriptor. The following options in alphabetical order can be used when `ud` is a device descriptor.

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. 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.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
IbaEOSrd	0x000C	zero = The EOS character is ignored during read operations. non-zero = Read operation will be 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. 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 device. See `ibeos`.
IbaReadAdjust	0x0013	0 = Read operations will not have pairs of bytes swapped. 1 = Read operations will have each pair of bytes swapped.

(continues)

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

Options (Constants)	Options (Values)	Returned Information
IbaWriteAdjust	0x0014	0 = Write operations will not have pairs of bytes swapped. 1 = Write operations will 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 UNT(Untalk) and UNL (Unlisten) will not be sent after each device level read and write operation. non-zero = The UNT and UNL commands will be sent after each device level read and write.
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.

Format

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

Parameters

IN `ud`	A device unit descriptor
IN `bname`	An access board name, for example, `GPIB0`.
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-488.2M Software Reference Manual for OS/2*.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The access board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBCAC **Board Level** IBCAC

Purpose

Become Active Controller.

Format

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

Parameters

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

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.

IBCAC

Board Level

IBCAC
(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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBCLR Device Level IBCLR

Purpose

Clear a specific device.

Format

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

Parameters

IN `ud`	A device unit descriptor
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 the *NI-488.2M Software Reference Manual for OS/2.*
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBCMD Board Level IBCMD

Purpose

Send GPIB commands.

Format

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

Parameters

IN `ud`	A board unit descriptor
IN `cmdbuf`	Buffer of command bytes to send
IN `cnt`	Number of command bytes to send
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-488.2M driver is not installed.
ENEB	-	The interface board is either not installed or not configured properly.
ENOL	-	There are no Listeners on the GPIB.
EOIP	-	Asynchronous I/O is in progress.

IBCMDA **Board Level** IBCMDA

Purpose

Send GPIB commands asynchronously.

Format

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

Parameters

IN `ud`	A board unit descriptor
IN `buffer`	Buffer of command bytes to send
IN `cnt`	Number of command bytes to send
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 disallowed. The driver returns EOIP in this case.

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

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

- `ibwait` If the returned `ibsta` contains CMPL, the driver and application are resynchronized.
- `ibstop` The I/O is canceled; the driver and application are resynchronized.
- `ibonl` The I/O is canceled and the interface is reset; the driver and application are resynchronized.

Possible Errors

EARG - `ud` is valid but does not refer to an interface board.
ECIC - The interface board is not Controller-In-Charge.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no Listeners on the GPIB.
EOIP - Asynchronous I/O is in progress.

IBCONFIG

Board Level
Device Level

IBCONFIG

Purpose

Change the software configuration parameters.

Format

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

Parameters

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

Description

`ibconfig` alters the current value of 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 returned 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-488.2M driver is not installed.
EOIP - Asynchronous I/O is in progress.

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

Table 1-5 lists the constants and hexadecimal values of the options you can use with `ibconfig` when `ud` is a board descriptor or a board index. The following options in alphabetical order can be used when `ud` is a board descriptor.

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

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`. (Default determined by `ibconf`)
IbcSAD	0x0002	Changes the secondary address of the board. Identical to `ibsad`. (Default determined by `ibconf`)
IbcTMO	0x0003	Changes the I/O timeout limit of the board. Identical to `ibtmo`. (Default determined by `ibconf`)
IbcEOT	0x0004	Changes the data termination mode for write operations. Identical to `ibeot`. (Default determined by `ibconf`)
IbcPPC	0x0005	Configures the board for parallel polls. Identical to board-level `ibppc`. (Default: zero)

(continues)

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

Options (Constants)	Options (Values)	Legal Values
IbcAUTOPOLL	0x0007	zero = Disable automatic serial polling. non-zero = Enable automatic serial polling. (Default determined by `ibconf`) See the *Using Serial Polling* section in the *NI-488.2M Software Reference Manual for OS/2* for a discussion of automatic serial polling.
IbcCICPROT	0x0008	zero = Disable the CIC protocol. non-zero = Enable the CIC protocol. (Default determined by `ibconf`) See the *Device-Level Calls and Bus Management* section in the *NI-488.2M Software Reference Manual for OS/2* for a discussion of the CIC protocol.
IbcIRQ	0x0009	zero = Do not use interrupts. non-zero = Use interrupts (use the hardware interrupt level configured through `ibconf`). (Default determined by `ibconf`)
IbcSC	0x000A	Request or release system control. Identical to `ibrsc`. (Default determined by `ibconf`)

(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. (Default determined by `ibconf`)
IbcEOSwrt	0x000D	zero = Do not assert EOI with the EOS character during writes operations. non-zero = Assert EOI with the EOS character during writes. (Default determined by `ibconf`)
IbcEOScmp	0x000E	zero = Use 7 bits for the EOS character comparison. non-zero = Use 8 bits for the EOS character comparison. (Default determined by `ibconf`)
IbcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character. (Default determined by `ibconf`)

(continues)

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

Options (Constants)	Options (Values)	Legal Values
IbcPP2	0x0010	zero = PP1 mode (remote parallel poll configuration). non-zero = PP2 mode (local parallel poll configuration). (Default: zero) See the *Using Parallel Polling* section in the *NI-488.2M Software Reference Manual for OS/2* for a discussion of 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). (Default determined by `ibconf`) The T1 delay is the GPIB Source Handshake timing.
IbcDMA	0x0012	Identical to `ibdma`. (Default determined by `ibconf`)
IbcReadAdjust	0x0013	0 = No byte swapping. 1 = Swap pairs of bytes during a read. (Default: 0)

(continues)

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

Options (Constants)	Options (Values)	Legal Values
IbcWriteAdjust	0x0014	0 = No byte swapping. 1 = Swap pairs of bytes during a write. (Default: 0)
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: 0)
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)

IBCONFIG

Board Level
Device Level

IBCONFIG
(Continued)

Table 1-6 lists the constants and hexadecimal values of the options you can use with `ibconfig` when `ud` is a device descriptor. The following options in alphabetical order can be used when `ud` is a device descriptor.

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`. (Default determined by `ibconf`)
IbcSAD	0x0002	Changes the secondary address of the device. Identical to `ibsad`. (Default determined by `ibconf`)
IbcTMO	0x0003	Changes the device I/O timeout limit. Identical to `ibtmo`. (Default determined by `ibconf`)
IbcEOT	0x0004	Changes the data termination method for writes. Identical to `ibeot`. (Default determined by `ibconf`)
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. (Default determined by `ibconf`)
IbcEOSrd	0x000C	non-zero = Terminate reads when the EOS character is read. (Default determined by `ibconf`)

(continues)

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

Options (Constants)	Options (Values)	Legal Values
IbcEOSwrt	0x000D	zero = Do not send EOI with the EOS character during write operations. non-zero = Send EOI with the EOS character during writes. (Default determined by `ibconf`)
IbcEOScmp	0x000E	zero = Use 7 bits for the EOS character comparison. non-zero = Use 8 bits for the EOS character comparison. (Default determined by `ibconf`)
IbcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character. (Default determined by `ibconf`)
IbcReadAdjust	0x0013	0 = No byte swapping. 1 = Swap pairs of bytes during a read. (Default: 0)
IbcWriteAdjust	0x0014	0 = No byte swapping. 1 = Swap pairs of bytes during a write. (Default: 0)

(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.

Format

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

Parameters

IN `BdIndx`	Index of the access board for the device
IN `pad`	The primary GPIB address of the device
IN `sad`	The secondary GPIB address of the device
IN `tmo`	The I/O timeout value
IN `eot`	EOI mode of the device
IN `eos`	EOS character and modes
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 configuration, according to the input parameters.

`BdIndx` indicates the access board (for example, `GPIB0` if `BdIndx` is 0), `pad` sets the primary address, `sad` sets the secondary address, `tmo` sets the I/O timeout period, `eot` sets the end of transfer mode, and `EOS` sets the end-of-string information. 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`.

IBDEV **Device Level** IBDEV (Continued)

`ibdev` acquires and initializes a device descriptor from the set of user-configurable devices—that is, `DEV1`, `DEV2`, `...`, `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 either not installed or not configured properly.

IBDMA **Board Level** IBDMA

Purpose

Enable or disable DMA.

Format

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

Parameters

IN `ud`	A board descriptor
IN `v`	Enable or disable the use of DMA
Function Return	The value of `ibsta`

Description

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

Possible Errors

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

IBEOS

Board Level
Device Level

IBEOS

Purpose

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

Format

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

Parameters

IN `ud`	A board or device descriptor
IN `v`	EOS mode and character information
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, the EOS configuration is disabled. Otherwise, the low byte is the EOS character and the upper byte contains flags that define the EOS mode; Table 1-7 describes the different EOS configurations. If no error occurs during the call, the value of the previous EOS setting is returned in `iberr`.

IBEOS

Board Level
Device Level

IBEOS
(Continued)

Table 1-7. EOS Configurations

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

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

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

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

For more information on the termination of I/O operations, refer to Chapter 6, *GPIB Programming Techniques*, in the *NI-488.2M Software Reference Manual for OS/2.*

IBEOS

Board Level
Device Level

IBEOS
(Continued)

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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBEOT

Board Level
Device Level

IBEOT
(Continued)

Purpose

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

Format

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

Parameters

IN `ud`	A board or device descriptor
IN `v`	Enables or disables the assertion of EOI at the end of transmission
Function Return	The value of `ibsta`

Description

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

For more information on the termination of I/O operations, refer to Chapter 6, *GPIB Programming Techniques*, in the *NI-488.2M Software Reference Manual for OS/2.*

Possible Errors

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

IBFIND

Board Level
Device Level

IBFIND

Purpose

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

Format

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

Parameters

IN `udname`	A user-configured device or board name
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`.

Note: ***The use of*** `ibfind` ***to obtain device descriptors is provided for compatibility with existing applications. New applications should use*** `ibdev` ***instead.*** `ibdev` ***is more flexible, simpler to use, and frees the application from unnecessary device name requirements.***

IBFIND

Board Level
Device Level

IBFIND
(Continued)

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-488.2M Software Reference Manual for OS/2*.
EDVR - Either `udname` is not recognized as a board or device name or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.

IBGTS Board Level IBGTS

Purpose

Go from Active Controller to Standby.

Format

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

Parameters

IN `ud`	Board descriptor
IN `v`	Determines whether to perform acceptor handshaking
Function Return	The value of `ibsta`

Description

`ibgts` causes the board specified by `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, the `ibeos` function should be called to establish proper EOS modes.

For more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987.

IBGTS

Board Level

IBGTS
(Continued)

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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBIST Board Level IBIST

Purpose

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

Format

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

Parameters

IN `ud`	Board descriptor
IN `v`	Indicates whether to set or clear the `ist` bit
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 *Parallel Polling* section in the *NI-488.2M Software Reference Manual for OS/2*.

Possible Errors

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

IBLINES **Board Level** IBLINES

Purpose

Return the status of the eight GPIB control lines.

Format

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

Parameters

IN `ud`	Board or device descriptor
OUT `clines`	Returns GPIB control line state information
Function Return	The value of `ibsta`

Description

`iblines` returns the GPIB control lines state information 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 bit can be monitored (indicated by a 1 in the appropriate bit position), check the corresponding bit in the upper byte. If the bit is set (1), the corresponding control line is asserted. If the bit is clear (0), the control line is unasserted.

IBLINES **Board Level** IBLINES (Continued)

Example

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

Possible Errors

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

IBLN

Board Level
Device Level

IBLN

Purpose

Check for the presence of a device on the bus.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `pad`	The primary GPIB address of the device
IN `sad`	The secondary GPIB address of the device
OUT `listen`	Indicates if a device is present or not
Function Return	The value of `ibsta`

Description

This function determines whether there is a listening device at the GPIB address designated by the `pad` and `sad` parameters. If `ud` is a board descriptor, the bus associated with that board is tested for Listeners. If `ud` is a device descriptor, `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`; otherwise, 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` is used to designate that no secondary address is to be tested (only a primary address is tested) and the constant `ALL_SAD` is used to designate that all secondary addresses are to be tested.

IBLN

Board Level
Device Level

IBLN
(Continued)

Possible Errors

EARG - `pad` or `sad` is invalid.
ECIC - Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in the *NI-488.2M Software Reference Manual for OS/2*.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBLOC

Board Level
Device Level

IBLOC

Purpose

Go to Local.

Format

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

Parameters

IN `ud`	Board or device descriptor
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`), this function 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.

IBLOC

Board Level
Device Level

IBLOC
(Continued)

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-488.2M Software Reference Manual for OS/2*.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBONL

Board Level
Device Level

IBONL

Purpose

Place the device or interface board online or offline.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `v`	Indicates whether the board or device is to be put online or taken offline
Function Return	The value of `ibsta`

Description

`ibonl` resets the board or device and places all its software configurable 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. Execute an `ibfind` or `ibdev` to access the board or device again.

Possible Errors

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

IBPAD

Board Level
Device Level

IBPAD

Purpose

Change the primary address.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `v`	GPIB primary address
Function Return	The value of `ibsta`

Description

`ibpad` sets the primary GPIB address of the board or device to `v`, an integer ranging between 0 and 30. If no error occurs during the call, `iberr` contains the previous GPIB primary address.

Possible Errors

EARG - `v` is not a valid primary GPIB address; it must be in the range 0–30.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBPCT Device Level IBPCT

Purpose

Pass control to another GPIB device with Controller capability.

Format

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

Parameters

IN `ud`	Device descriptor
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 (CIDS). This function assumes that the device has Controller capability.

Possible Errors

EARG - `ud` is valid 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 the *NI-488.2M Software Reference Manual for OS/2.*
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBPPC

Board Level
Device Level

IBPPC

Purpose

Parallel poll configure.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `v`	Parallel poll enable/disable value
Function Return	The value of `ibsta`

Description

Device Level

The parameter `v` describes the parallel poll configuration value (Parallel Poll Enable (PPE) or Disable (PPD) message). 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 (PPE or PPD) in `v`. Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E). If no error occurs during the call, `iberr` contains the previous value of the device parallel poll configuration.

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). If no error occurs during the call, `iberr` contains the previous value of the local parallel poll configuration.

For more information on parallel polling, refer to the *Parallel Polling* section in the *NI-488.2M Software Reference Manual for OS/2*.

IBPPC

Board Level
Device Level

IBPPC
(Continued)

Possible Errors

EARG - v does not contain a valid PPE or PPD message.
EBUS - Device level: There are no devices 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-488.2M Software Reference Manual for OS/2*.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBRD

Board Level
Device Level

IBRD

Purpose

Read data from a device into a user buffer.

Format

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

Parameters

IN `ud`	Board or device descriptor
OUT `rdbuf`	Address of buffer into which data is read
IN `cnt`	Number of bytes to be read from the GPIB
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, this function addresses the GPIB and reads up to `cnt` bytes of data from a GPIB device and places it into the buffer pointed to 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`.

IBRD

Board Level
Device Level

IBRD
(Continued)

Board Level

If `ud` is a board descriptor, this function reads up to `cnt` bytes of data from a GPIB device and places it into the buffer pointed to 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 and a Device Clear is sent on the GPIB by the CIC. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Possible Errors

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

EADR - Board level: The GPIB is not correctly addressed; use `ibcmd` to address the GPIB.
Device level: There is a conflict between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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-488.2M Software Reference Manual for OS/2*.

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

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

EOIP - Asynchronous I/O is in progress.

IBRDA

Board Level
Device Level

IBRDA

Purpose

Read data asynchronously from a device into a user buffer.

Format

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

Parameters

IN `ud`	Board or device descriptor
OUT `rdbuf`	Address of buffer into which data is read
IN `cnt`	Number of bytes to be read from the GPIB
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, this function addresses the GPIB and begins an asynchronous read up to `cnt` bytes of data from a GPIB device and places it into the buffer pointed to 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`.

IBRDA

Board Level
Device Level

IBRDA
(Continued)

Board Level

If `ud` is a board descriptor, this function reads up to `cnt` bytes of data asynchronously from a GPIB device and places it into the buffer pointed to 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 transfer could not complete within the timeout period or if the board is not the CIC and a Device Clear is sent on the GPIB by the CIC. 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 disallowed. The driver returns EOIP in this case.

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

- `ibwait` If the returned `ibsta` contains CMPL, the driver and application are resynchronized.
- `ibstop` The I/O is canceled; the driver and application are resynchronized.
- `ibonl` The I/O is canceled and the interface is reset; the driver and application are resynchronized.

Possible Errors

EABO - Board level: a Device Clear message was received from the CIC.

EADR - Board level: The GPIB is not correctly addressed; use `ibcmd` to address the GPIB.
Device level: There is a conflict between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

(continues)

IBRDA

Board Level
Device Level

IBRDA (Continued)

Possible Errors (Continued)

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-488.2M Software Reference Manual for OS/2*.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBRDF

Board Level
Device Level

IBRDF

Purpose

Read data from a device into a file.

Format

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

Parameter

IN `ud`	Board or device descriptor
IN `flname`	Name of file into which data is read
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, this function addresses the GPIB and reads all the bytes of data from a GPIB device and places it into the file indicated by `flname`. The operation terminates normally when all the 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`.

IBRDF

Board Level
Device Level

IBRDF
(Continued)

Board Level

If `ud` is a board descriptor, this function reads all the bytes from a GPIB device and places them into the file named by `flname`. A board-level `ibrdf` assumes that the GPIB is already properly addressed. The operation terminates when all the 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 and a Device Clear is sent on the GPIB by the CIC. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Possible Errors

EABO - Either `cnt` bytes or END was not received within the preconfigured 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: There is a conflict between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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-488.2M Software Reference Manual for OS/2*.

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

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

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

EOIP - Asynchronous I/O is in progress.

IBRPP

Board Level
Device Level

IBRPP

Purpose

Conduct a parallel poll.

Format

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

Parameters

IN `ud`	Board or device descriptor
OUT `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 Chapter 6, *GPIB Programming Techniques,* in the *NI-488.2M Software Reference Manual for OS/2.*

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-488.2M Software Reference Manual for OS/2.*
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBRSC Board Level IBRSC

Purpose

Request or release system control.

Format

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

Parameters

IN `ud`	Board descriptor
IN `v`	Determines if system control is to be requested or released
Function Return	The value of `ibsta`

Description

This function is used to request or relinquish 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 control capability are disallowed. 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBRSP **Device Level** IBRSP

Purpose

Conduct a serial poll.

Format

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

Parameters

IN `ud`	Device descriptor
OUT `spr`	Serial poll response byte
Function Return	The value of `ibsta`

Description

The `ibrsp` function is used to serial poll the device specified by `ud`. The serial poll response byte is returned in `spr`. If bit 6 (hex 40) of the response is set, the device is requesting service. When the automatic serial polling feature is enabled, the device might have already been polled. In this case, `ibrsp` returns a previously acquired status byte.

For more information on serial polling, refer to Chapter 6, *GPIB Programming Techniques,* in the *NI-488.2M Software Reference Manual for OS/2*.

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 - 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 the *NI-488.2M Software Reference Manual for OS/2.*
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.
ESTB - Autopolling is enabled and the serial poll queue of the device has overflowed. Call `ibrsp` more often to keep the queue from overflowing or disable automatic serial polling.

IBRSV Board Level IBRSV

Purpose

Request service and change the serial poll status byte.

Format

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

Parameters

IN `ud`	Board descriptor
IN `v`	Serial poll status byte
Function Return	The value of `ibsta`

Description

`ibrsv` requests service from the Controller and provides 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`.

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

Possible Errors

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

IBSAD

Board Level
Device Level

IBSAD

Purpose

Change or disable the secondary address.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `v`	GPIB secondary address
Function Return	The value of `ibsta`

Description

`ibsad` sets 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, the previous value of the GPIB secondary address is returned in `iberr`.

Possible Errors

EARG - `v` is non-zero and outside the legal range 96 to 126.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBSIC **Board Level** IBSIC

Purpose

Assert interface clear.

Format

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

Parameters

IN `ud`	Board descriptor
Function Return	The value of `ibsta`

Description

This function asserts the GPIB Interface Clear (IFC) line for at least 100 microseconds 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 your device internal functions.

Possible Errors

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

Format

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

Parameters

IN `ud`	Board descriptor
IN `v`	Indicates whether to set or clear the REN line
Function Return	The value of `ibsta`

Description

If `v` is non-zero, the GPIB Remote Enable (REN) line is asserted. Otherwise, it is unasserted. The previous value of REN is returned in `iberr`.

REN is used by devices to select 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
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.

Format

```
int ibsrq (void (far *funcname) (void));
```

Parameters

IN `funcname`	C interrupt handling routine
Function Return	The value of `ibsta`

Description

This function establishes a call to the C routine `funcname` whenever the SRQI bit is set in the status word (`ibsta`). If the SRQI bit is set, `funcname` is called by the language interface 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.

Format

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

Parameters

IN `ud`	Board or device descriptor
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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.

IBTMO

Board Level
Device Level

IBTMO

Purpose

Change or disable the I/O timeout period.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `v`	Timeout duration code
Function Return	The value of `ibsta`

Description

The timeout period is set to `v`. The timeout period is used to select the approximate duration of a synchronous I/O operation (for example, `ibrd` and `ibwrt`). If the operation does not complete before the timeout period elapses, 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 timeout period may be longer.

Possible Errors

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

Table 1-8. Timeout Code Values

Constant	Value of v	MinimumTimeout
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

IBTRG Device Level IBTRG

Purpose

Trigger selected device.

Format

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

Parameters

IN `ud`	Device descriptor
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 - 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 the *NI-488.2M Software Reference Manual for OS/2.*
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

IBWAIT

Board Level
Device Level

IBWAIT

Purpose

Wait for GPIB events.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `mask`	Bit mask of GPIB events to wait for
Function Return	The value of `ibsta`

Description

This function is used to monitor the events selected by `mask` and to delay processing until one or more events occur. If 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 more of the specified events have not already occurred. If TIMO is not set in the wait mask, the function will wait indefinitely for one or more of the specified events to occur. The `ibwait` mask bits are described in Table 1-9 and they are identical to the `ibsta` bits. If `ud` is a device descriptor, the only valid wait mask bits are TIMO, END, RQS and CMPL If `ud` is a board descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period by using the `ibtmo` function.

IBWAIT

Board Level
Device Level

IBWAIT
(Continued)

Possible Errors

EARG - The bit set in `mask` is invalid.
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-488.2M Software Reference Manual for OS/2.*
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
ESRQ - Device level: If the RQS bit is set in the wait mask then ESRQ indicates that the *Stuck SRQ* condition exists. For more information on serial polling, refer to the *GPIB Programming Considerations* section in the *NI-488.2M Software Reference Manual for OS/2* .

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)
CMPL	8	100	I/O completed
LOK	7	80	GPIB board is in Lockout State

(continues)

Table 1-9. Wait Mask Layout (Continued)

Mnemonic	Bit Pos.	Hex Value	Description
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.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `wrtbuf`	Address of the buffer containing the bytes to write
IN `cnt`	Number of bytes to be written
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, this function 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, this function writes `cnt` bytes of data from the buffer pointed to 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 and a Device Clear is sent on the GPIB by the CIC. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBWRT

Board Level
Device Level

IBWRT
(Continued)

Possible Errors

EABO - Either `cnt` bytes were not sent within the preconfigured timeout period or a Device Clear message was received after starting the read operation.

EADR - Board level: The GPIB is not correctly addressed; use `ibcmd` to address the GPIB.
Device level: There is a conflict between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.

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-488.2M Software Reference Manual for OS/2*.

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

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

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.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `wrtbuf`	Address of the buffer containing the bytes to write
IN `cnt`	Number of bytes to be written
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, this function addresses the GPIB properly and writes `cnt` bytes from the buffer pointed to by `wrtbuf` to a GPIB device asynchronously. 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, this function 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 `cnt` bytes could not be sent within the timeout period or if the board is not CIC and a Device Clear is sent on the GPIB by the CIC. 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 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 disallowed. The driver will return EOIP in this case.

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

- `ibwait` If the returned `ibsta` contains CMPL, the driver and application are resynchronized.
- `ibstop` The I/O is canceled; the driver and application are resynchronized.
- `ibonl` The I/O is canceled and the interface is reset; the driver and application are resynchronized.

Possible Errors

EABO - Board level: a Device Clear message was received from the CIC.
EADR - Board level: The GPIB is not correctly addressed; use `ibcmd` to address the GPIB.
Device level: There is a conflict between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.
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-488.2M Software Reference Manual for OS/2*.
EDVR - Either `ud` is invalid or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
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.

Format

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

Parameters

IN `ud`	Board or device descriptor
IN `flname`	Name of file containing the data to be written
Function Return	The value of `ibsta`

Description

Device Level

If `ud` is a device descriptor, this function addresses the GPIB and writes all the bytes in the file `flname` to a GPIB device. The operation terminates normally when all the bytes have been sent. The operation terminates with an error if all 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, this function writes all 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 the bytes have been sent. The operation terminates with an error if all the bytes could not be sent within the timeout period or if the board is not CIC and a Device Clear is sent on the GPIB by the CIC. The actual number of bytes transferred is returned in the global variable `ibcntl`.

IBWRTF

Board Level
Device Level

IBWRTF
(Continued)

Possible Errors

EABO - Either the file could not be transferred within the preconfigured timeout period or a Device Clear message was received after starting the read operation.

EADR - Board level: The GPIB is not correctly addressed; use `ibcmd` to address the GPIB.
Device level: There is a conflict between the device GPIB address and the GPIB address of the device access board use `ibpad` and `ibsad`.

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-488.2M Software Reference Manual for OS/2*.

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

EFSO - Could not access `flname`.

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

EOIP - Asynchronous I/O is in progress.

Chapter 2
NI-488.2 Routines

This chapter lists the available NI-488 routines, then describes the purpose, format, input and output parameters, and possible errors for each routine.

While using the routines, you might find it helpful to refer to Chapter 2, *Application Examples*, Chapter 3, *Developing Your Application*, and Chapter 6, *GPIB Programming Techniques*, in the *NI-488.2M Software Reference Manual for OS/2*.

Routine Names

The routines in this chapter are listed alphabetically.

Purpose

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

Format

The format of each NI-488.2 routine is given.

Parameters

The input and output parameters of each function are listed. IN designates an input parameter and OUT designates an output parameter.

Most of the NI-488.2 routines have an input parameter that is either a single address or a list of addresses. The address parameter is a 16-bit integer that has two comp7 primary address (0 to 30), and the high byte is a valid secondary address (`NO_SAD(0)` or 96 to 126). 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.

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
`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 to be a Listener in preparation for `RcvRespMsg`
`ResetSys`	Reset and initialize IEEE 488.2-compliant devices
`Send`	Send data bytes to a device
`SendCmds`	Send GPIB command bytes
`SendDataBytes`	Send data bytes to devices that are already addressed to listen

(continues)

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

Routine	Purpose
SendIFC	Reset the GPIB by sending the Interface Clear (IFC) message
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.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`
OUT `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 - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

DevClear DevClear

Purpose

Clear a single device.

Format

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

Parameters

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

Description

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

Possible Errors

EARG - The `address` parameter is invalid (out of range).
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

DevClearList DevClearList

Purpose

Clear multiple devices.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`

Description

`DevClearList` sends the Selected Device Clear (SDC) GPIB message to all the device addresses described by `addrlist`. If `addrlist` contains only the constant `NOADDR`, the Universal Device Clear (DCL) message is sent to all the devices on the bus.

Possible Errors

EARG - An invalid address (out of range) appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

EnableLocal EnableLocal

Purpose

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

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`

Description

`EnableLocal` sends the Go To Local (GTL) GPIB message to all the devices described by `addrlist`. This action places the devices into local mode. If `addrlist` contains only the constant `NOADDR`, the Remote Enable (REN) GPIB line is unasserted.

Possible Errors

EARG - An invalid address (out of range) appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.
ESAC - The interface board is not configured as System Controller.

EnableRemote EnableRemote

Purpose

Enable remote GPIB programming of devices.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`

Description

`EnableRemote` asserts the Remote Enable (REN) GPIB line. All devices described by `addrlist` are put into a listen-active state.

Possible Errors:

EARG - An invalid address (out of range) appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
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.

Format

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

Parameter

IN `boardID`	The interface board number
IN `padlist`	A list of primary addresses terminated by `NOADDR`
OUT `resultlist`	Addresses of all listening devices found by `FindLstn` are placed in this array
IN `limit`	Total number of entries that can be placed in `resultlist`

Description

All of the primary addresses in `padlist` are tested, as follows. If a device is present at a primary address given in `padlist`, the primary address is stored in `resultlist`. Otherwise, all secondary addresses of the primary address are tested; the addresses of any devices found are stored in `resultlist`. No more than `limit` addresses will be stored in `resultlist`; `ibcntl` contains the 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 primary address in the `padlist` array.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.
ETAB - The number of devices found on the GPIB exceed `limit`.

FindRQS FindRQS

Purpose

Determines which device is requesting service.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	List of device addresses terminated by `NOADDR`
OUT `result`	Serial poll response byte of the device that is requesting service

Description

The devices described by `addrlist` are serial polled in order until the routine finds a device that is requesting service. The serial poll response byte is 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

EABO - One of the devices times out instead of responding to the serial poll; `ibcntl` contains the index of the timed out device address.
EARG - An invalid address (out of range) appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
EBUS - There are no devices connected to the GPIB.
ECIC - The interface board is not the Controller-In-Charge; see `SendIFC`.

(continues)

FindRQS

FindRQS
(Continued)

Possible Errors (Continued)

EDVR - Either `boardID` is invalid (out of range) or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.
ETAB - None of the devices in `addrlist` are requesting service or `addrlist` contains only `NOADDR`. `ibcntl` contains the index of `NOADDR` in `addrlist`.

PassControl PassControl

Purpose

Pass control to another device with Controller capability.

Format

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

Parameters

IN `boardID`	The interface board number
IN `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 the interface board is no longer CIC.

Possible Errors

EARG - The `address` parameter is invalid (out of range).
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

PPoll PPoll

Purpose

Perform a parallel poll on the GPIB.

Format

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

Parameters

IN `boardID`	The interface board number
OUT `result`	The parallel poll result

Description:

`PPoll` conducts a parallel poll and the result is placed in `result`. Each of the 8 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 Chapter 6, *GPIB Programming Techniques*, in the *NI-488.2M Software Reference Manual for OS/2*.

Possible Errors

EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

PPollConfig PPollConfig

Purpose

Configure a device to respond to parallel polls.

Format

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

Parameters

IN `boardID`	The interface board number
IN `address`	Address of the device to be configured
IN `dataline`	Data line (a value in the range of 1 to 8) on which the device responds to parallel polls
IN `lineSense`	Sense (either 0 or 1) of the parallel poll response

Description

The device described by `address` is configured to respond to parallel polls by asserting or not asserting the GPIB data line, `dataline`. If `lineSense` equals the individual status (`ist`) bit of the device, the assigned GPIB data line is asserted during a parallel poll; otherwise, the data line is not asserted during a parallel poll. The Controller can use parallel polling to obtain one-bit, device-dependent status messages from up to eight devices simultaneously.

For more information on parallel polling, refer to Chapter 6, *GPIB Programming Techniques,* in the *NI-488.2M Software Reference Manual for OS/2.*

PPollConfig

PPollConfig
(Continued)

Possible Errors

EARG - The `address` parameter is invalid (out of range); `dataline` is not in the range 1..8; `lineSense` is not 0 or 1.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

PPollUnconfig PPollUnconfig

Purpose

Unconfigure devices so that the devices will not respond to parallel polls.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`

Description

`PPollUnconfig` unconfigures all the devices described by the constant `addrlist` for parallel polls. If `addrlist` contains only the constant `NOADDR`, the Parallel Poll Unconfigure (PPU) GPIB message is sent to all GPIB devices. The devices unconfigured by this function will not participate in future parallel polls.

For more information on parallel polling, refer to Chapter 6, *GPIB Programming Techniques,* in the *NI-488.2M Software Reference Manual for OS/2.*

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 - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

RcvRespMsg RcvRespMsg

Purpose

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

Format

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

Parameters

IN `boardID`	The interface board number
OUT `buffer`	Stores the received data bytes
IN `cnt`	Number of bytes read
IN `termination`	Description of the data termination mode (`STOPend` or an 8-bit EOS character)

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 are read or the termination condition is detected. If the termination condition is `STOPend`, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when an 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable `ibcntl`.

This routine assumes that the interface board is already in its listen-active state and a device is already addressed to be a Talker (see `ReceiveSetup`, `Receive`, and `SendCmds`).

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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

ReadStatusByte ReadStatusByte

Purpose

Serial poll a single device.

Format

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

Parameters

IN `boardID`	The interface board number
IN `address`	A device address
OUT `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 - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

Receive Receive

Purpose

Read data bytes from a device.

Format

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

Parameters

IN `boardID`	The interface board number
IN `address`	Address of a device to receive data
OUT `buffer`	Stores the received data bytes
IN `cnt`	Number of bytes to read
IN `termination`	Description of the data termination mode (`STOPend` or an EOS character)

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 you read up to `cnt` bytes or the termination condition is detected. If the termination condition is `STOPend`, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when an 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable `ibcntl`.

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. The `termination` parameter must be either `STOPend` or an 8-bit EOS character.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
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`.

Format

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

Parameters

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

Description

`ReceiveSetup` makes the device described by `address` talker active; the interface board is made 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).
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

ResetSys ResetSys

Purpose

Reset and initialize IEEE 488.2 compliant devices.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses 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, 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 is accomplished by sending the message `"*RST"` 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.
EBUS - There are no devices 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-488.2M driver is not installed.

(continues)

ResetSys

ResetSys (Continued)

Possible Errors (Continued)

ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no Listeners 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.

Format

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

Parameters

IN `boardID`	The interface board number
IN `address`	Address of a device to which data is to be sent
IN `buffer`	The data bytes to be sent
IN `datacnt`	Number of bytes to be sent
IN `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`, a new line character (`'\n'`) is sent with the EOI line asserted after the last byte of the `buffer`. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Possible Errors

EABO - The I/O timeout period has expired before all of the bytes were sent.
EARG - The `address` or `eotmode` parameter is invalid, or `eotmode` is `DABEnd` and `datacnt` is zero.
EBUS - There are no devices connected to the GPIB.

(continues)

Send

Possible Errors (Continued)

ECIC - The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR - Either `boardID` is invalid (out of range) or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no Listeners on the GPIB to accept the data bytes.
EOIP - Asynchronous I/O is in progress.

SendCmds SendCmds

Purpose

Send GPIB command bytes.

Format

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

Parameters

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

Description

`SendCmds` sends `cnt` command bytes from `buffer` to the GPIB. You should use this routine when you need specialized command sequences not provided by other routines. The actual number of command bytes transferred is returned in the global variable `ibcntl`.

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 has 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no devices connected to the GPIB.
EOIP - Asynchronous I/O is in progress.

SendDataBytes SendDataBytes

Purpose

Send data bytes to devices that are addressed to listen.

Format

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

Parameters

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

Description

`SendDataBytes` sends `datacnt` number of bytes from the buffer to devices that 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`, 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`.

This routine assumes that the interface board is in talk-active state and there are devices already addressed as Listeners on the GPIB (see `SendSetup`, `Send`, `SendList`, and `SendCmds`).

SendDataBytes SendDataBytes (Continued)

Possible Errors

EABO - The I/O timeout period expired before all of the bytes were sent.
EADR - Interface board 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 `eotmode` is `DABEnd` and `datacnt` is zero.
ECIC - The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR - Either `boardID` is invalid (out of range) or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no Listeners 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.

Format

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

Parameters

IN `boardID` The interface board number

Description

`SendIFC` is used as part of GPIB initialization. It forces the interface board to be Controller-In-Charge of the GPIB, and ensures that the connected devices are all unaddressed and the interface functions of the devices are in their idle states.

Possible Errors

EDVR - Either `boardID` is invalid (out of range) or the NI-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
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.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses to send data
IN `buffer`	The data bytes to be sent
IN `datacnt`	Number of bytes transmitted
IN `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`, a new line character (`'\n'`) is sent with the EOI line asserted after the last byte of the `buffer`. The actual number of bytes transferred is returned in the global variable `ibcntl`.

SendList

SendList
(Continued)

Possible Errors

EABO - The I/O timeout period has 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; or the `eotmode` parameter is invalid; or `eotmode` is `DABEnd` and `datacnt` is zero.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

SendLLO SendLLO

Purpose

Send the Local Lockout (LLO) message to all devices.

Format

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

Parameters

IN `boardID`	The interface board number

Description

`SendLLO` sends the GPIB Local Lockout (LLO) message to all devices. While Local Lockout is in effect, only the Controller-In-Charge can alter the state of the devices by sending appropriate GPIB messages. `SendLLO` is reserved for use in unusual local/remote situations. In the typical case of placing the devices in remote mode with local lockout, you should consider `SetRWLS`.

Possible Errors

EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.
ESAC - The interface board is not configured as System Controller.

SendSetup SendSetup

Purpose

Setup devices to receive data in preparation for `SendDataBytes`.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`

Description

`SendSetup` makes the devices described by `addrlist` listen active and 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 setup the addressing before making multiple calls to `SendDataBytes`: it eliminates the need to readdress the devices for every block of data.

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 - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

SetRWLS SetRWLS

Purpose

Place devices in Remote With Lockout state.

Format

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

Parameters

IN `boardID`	The interface board number
IN `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.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
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.

Format

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

Parameters

IN `boardID`	The interface board number
OUT `result`	State of the SRQ line, 1 if the line is asserted; 0 if the line is not asserted

Description

`TestSRQ` returns the current state of the GPIB SRQ line in `result`. If SRQ is asserted, `result` contains a one; 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-488.2M driver is not installed.

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

TestSys TestSys

Purpose

Cause IEEE 488.2 compliant devices to conduct self tests.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`
OUT `resultlist`	A list of test results; each entry corresponds to an address in `addrlist`

Description

`TestSys` sends the "`*TST`" message to the IEEE 488.2 compliant devices described by `addrlist`. The "`*TST`" message instructs them to conduct their self-test procedures. A 16-bit test result code is read from each device and stored in `resultlist`. A test result of 0 indicates that the device passed its self test, and any other value indicates that the device failed its self test. A test result of -1 indicates that the device timed out before receiving a result; `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 timed out device. -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.
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no Listeners on the GPIB.
EOIP - Asynchronous I/O is in progress.

Trigger Trigger

Purpose

Trigger a device.

Format

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

Parameters

IN `boardID`	The interface board number
IN `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`, no addressing is performed and the GET message is issued to any device that is currently listen active.

Possible Errors

EARG - The `address` parameter is invalid (out of range).
EBUS - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

TriggerList TriggerList

Purpose

Trigger multiple devices.

Format

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

Parameters

IN `boardID`	The interface board number
IN `addrlist`	A list of device addresses terminated by `NOADDR`

Description

`TriggerList` sends the Group Execute Trigger (GET) GPIB message to the devices described by `addrlist`. If the only address in `addrlist` is the constant `NOADDR`, the GET message is issued 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 - There are no devices 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.
EOIP - Asynchronous I/O is in progress.

WaitSRQ WaitSRQ

Purpose

Wait until a device asserts the GPIB Service Request (SRQ) line.

Format

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

Parameters

IN `boardID`	The interface board number
OUT `result`	State of the SRQ line; 1 if line is asserted, 0 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 one 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-488.2M driver is not installed.
ENEB - The interface board is either not installed or not configured properly.

Chapter 3
API-Application Program Interface Functions

This chapter describes the OS/2 Application Program Interface (API) functions. The OS/2 API file I/O and control functions, or standard I/O functions of OS/2 API compatible languages, are used to control and access GPIB devices. The OS/2 API functions are directly accessible from high-level languages. No language interface is required.

The NI-488.2M driver services each board independently. File I/O and control functions issued to the NI-488.2M driver for a given GPIB board and its attached devices have no effect on any other GPIB boards that the NI-488.2M driver may be servicing.

Accessing GPIB Boards and Devices as Files

Because file I/O functions are issued to the NI-488.2M driver, GPIB board and device access are controlled by the access and sharing modes supported by OS/2. These modes are of particular importance in a multitasking system.

Devices and boards are accessed like OS/2 files and have the same characteristics as files. The following characteristics are particularly important for the development of GPIB applications:

- While opening a device or board, processes can define file sharing modes (that is, operations that other processes can perform by using the device or board). You can open devices to deny read and/or write access to other processes.

- While opening a device or board, processes can define file access modes (that is, operations that it can perform on a device or board). You can open devices specifically for read and/or write access.

API and Programming Language Compatibility

You can use the following OS/2 API functions to issue file I/O or GPIB control functions to the NI-488.2M software:

- `DosOpen`
- `DosClose`
- `DosRead`
- `DosWrite`
- `DosDevIOCtl`

The file I/O functions of the C language for OS/2 are fully compatible with the OS/2 API file I/O and control functions. The device or board descriptor returned by a `DosOpen` function is used to perform OS/2 API file and control functions.

The following section describes each API function.

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.

Format

32-bit and 16-bit formats of the API function are given.

Parameters

The input and output parameters for each function are listed. IN designates an input parameter and OUT designates an output parameter. Function Return describes the return value of the function.

Description

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

Possible Errors

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

Examples

32-bit and 16-bit examples are given for each function.

For more specific examples of tasks such as serial polling and parallel polling, refer to Chapter 6, *GPIB Programming Techniques,* in the *NI-488.2M Software Reference Manual for OS/2*.

Table 3-1. List of API Board-Level IOCtl Functions

Function	**Purpose**
ATNOFF	Go from Active Controller to Standby
ATNON	Become Active Controller
BConfRd	Return information about the initial configuration parameters of the board
BConfRdNew	Return information about initial configuration parameters of the board
BConfWrt	Modify initial configuration parameters of the board
BConfWrtNew	Modify initial configuration parameters of the board
BLOCAL	Go to local
BTempRd	Return information about the current configuration parameters of the board
BTempRdNew	Return information about the current configuration parameters of the board
BTempWrt	Modify current configuration parameters of the board
BTempWrtNew	Modify current configuration parameters of the board
BWAIT	Wait for GPIB events
CLRREMOTE	Clear the Remote Enable line
CMD	Send GPIB commands
PPOLL	Conduct a parallel poll
PPOLLCONF	Parallel poll configure
PPOLLIST	Set or clear `ist` bit for parallel polls
PRESENT	Check for the presence of the board
REQUEST	Request service

(continues)

Table 3-1. List of API Board-Level IOCtl Functions (Continued)

Function	Purpose
SENDIFC	Assert Interface Clear (IFC) for 100 µs
SETREMOTE	Set the Remote Enable line
SPOLLBYTE	Request service
STATUS	Return current status of the board

Table 3-2 List of API Device-Level IOCtl Functions

Function	Purpose
BWAIT	Wait for GPIB events
DCLEAR	Clear a specific device
DConfRd	Return information about the initial configuration parameters of the device
DConfRdNew	Return information about the initial configuration parameters of the device
DConfWrt	Modify initial configuration parameters of the device
DConfWrtNew	Modify initial configuration parameters of the device
DLOCAL	Go to local
DTempRd	Return information about current configuration parameters of the device
DTempRdNew	Return information about current configuration parameters of the device
DTempWrt	Modify current configuration parameters of the device
DTempWrtNew	Modify current configuration parameters of the device

(continues)

Table 3-2. List of API Device-Level IOCtl Functions (Continued)

Function	Purpose
OFFLINE	Take the device offline
ONLINE	Place the device online
PASSCONTROL	Pass control to another device with Controller capability
PPOLL	Conduct a parallel poll
SPOLL	Conduct a serial poll
STATUS	Return the status of the device
TRIGGER	Trigger selected device

DosOpen

Board Level
Device Level

DosOpen

Purpose

Open a board or device.

Format

32-Bit

```
ULONG DosOpen    ((PSZ) pDeviceName,
                 (PHFILE) AddrDevHandle,
                 (PULONG) AddrActionTaken,
                 (ULONG) FileSize,
                 (ULONG) DeviceAttribute,
                 (ULONG) OpenFlag,(ULONG) OpenMode, 0)
```

16-Bit

```
USHORT DosOpen   ((PSZ) pDeviceName,
                 (PHFILE) AddrDevHandle,
                 (PUSHORT) AddrActionTaken,
                 (USHORT) FileSize,
                 (USHORT) DeviceAttribute,
                 (USHORT) OpenFlag,(USHORT) OpenMode, 0)
```

DosOpen

Board Level
Device Level

DosOpen
(Continued)

Parameters

IN `pDeviceName`	Pointer to the name of the device to be opened
OUT `AddrDevHandle`	Address of device handle
OUT `AddrActionTaken`	Address of the variable receiving the value containing the action taken by the `DosOpen` function
IN `FileSize`	Logical size of the file. Not applicable while opening a device. Must be set to zero.
IN `DeviceAttribute`	Attributes of the device to be opened
IN `OpenFlag`	Determines the action to be taken depending on whether the file exists or not
IN `OpenMode`	The mode of the open function
Function Return	Zero, if no error; OS/2 error value, otherwise.

For more information on these parameters, refer to the `DosOpen` function in the *Control Program Programming Reference, OS/2 Technical Library*.

Description

The `DosOpen` function for a device or a board is used to acquire a file descriptor for a board or user-configured device. This file descriptor can be used in subsequent API calls.

`DosOpen` for a device initializes the software configuration parameters to default settings and brings the access board online, if it is not already online.

`DosOpen` for a board initializes the software configuration parameters to default settings and brings the board online.

The file descriptor returned by `DosOpen` remains valid until the device or board is closed using `DosClose`.

DosOpen

Board Level
Device Level

DosOpen
(Continued)

Possible Errors

For information on the possible error codes that could be returned by the system when `DosOpen` fails, refer to the `DosOpen` function in the *OS/2 Technical Library, Control Program Programming Reference.*

Example

```
#define INCL_DOSFILEMGR
#include <os2.h>

rc = DosOpen  ("DEV1", &bdh,
               &Action, 0,
               FILE_READONLY, /* device attribute is
                                 read only */
               OPEN_ACTION_OPEN_IF_EXISTS,  /* open the
                                               device if
                                               it already
                                               exists */
               OPEN_FLAGS_WRITE_THROUGH |
               OPEN_FLAGS_FAIL_ON_ERROR | /* device open
                                             modes */
               OPEN_FLAGS_NOINHERIT | OPEN_SHARE_DENYNONE |
               OPEN_ACCESS_READWRITE,
               0);
```

DosClose

Board Level
Device Level

DosClose

Purpose

Close a device or board.

Format

32-Bit

```
ULONG DosClose ((HFILE) DevHandle)
```

16-Bit

```
USHORT DosClose ((HFILE) DevHandle)
```

Parameters

IN `DevHandle`	Device Handle returned by a previous `DosOpen`
Function Return	Zero, if no error; OS/2 error value, otherwise.

Description

The `DosClose` function closes the device or board and returns the handle to the system. In addition, the device or interface board is taken offline. If a device or interface board is closed, the handle (`DevHandle`) is no longer valid. Execute a `DosOpen` command to access the device or board again.

DosClose

Board Level
Device Level

DosClose
(Continued)

Possible Errors

For information on the possible error codes that could be returned by the system when `DosClose` fails, refer to the `DosClose` function in the *OS/2 Technical Library, Control Program Programming Reference.*

Example

```
#define INCL_DOSFILEMGR
#include <os2.h>

rc = DosClose (bdh);
```

DosRead

Board Level
Device Level

DosRead

Purpose

Read data from a device into a user buffer.

Format

32-Bit

```
ULONG DosRead ((HFILE) bdh, (PVOID) rdbuf,
               (ULONG) BufferLength,
               (PULONG) AddrBytesRead)
```

16-Bit

```
USHORT DosRead ((HFILE) bdh, (PVOID) rdbuf,
                (USHORT) BufferLength,
                (PUSHORT) AddrBytesRead)
```

Parameters

IN `bdh`	Device or board handle
OUT `rdbuf`	Buffer to receive the bytes read
IN `BufferLength`	Number of bytes to be read
OUT `AddrBytesRead`	Address of variable to receive the number of bytes actually read
Function Return	Zero, if no error; OS/2 error value, otherwise.

Description

Device Level

If `bdh` is a device handle, this function addresses the GPIB and reads up to `BufferLength` bytes of data from a GPIB device and places it into the buffer pointed to by `rdbuf`. The operation terminates normally when `BufferLength` bytes have been received or END is received. The operation terminates with an error if the transfer could not be completed within the timeout period. The actual number of bytes transferred is returned in the variable `AddrBytesRead`.

DosRead

Board Level
Device Level

DosRead
(Continued)

Board Level

If `bdh` is a board handle, this function reads up to `BufferLength` bytes of data from a GPIB device and places it into the buffer pointed to by `rdbuf`. A board-level `DosRead` assumes that the GPIB is already properly addressed. The operation terminates normally when `BufferLength` bytes have been received or END is received. The operation terminates with an error if the transfer could not be completed within the timeout period or if the board is not CIC and a Device Clear is sent on the GPIB by the CIC. The actual number of bytes transferred is returned in the variable `AddrBytesRead`.

Possible Errors

For information on the possible error codes that could be returned by the system when `DosRead` fails, refer to the `DosRead` function in the *OS/2 Technical Library, Control Program Programming Reference.*

Example

32-Bit

```
#define INCL_DOSFILEMGR
#include <os2.h>

UCHAR BufferArea[256];     /* User Buffer   */
ULONG BufferLength = 256; /* Buffer Length */
ULONG BytesRead;          /* Bytes actually read*/


rc = DosRead  (bdh, BufferArea, BufferLength, &BytesRead);
                  /* On successful return, the user
                  supplied buffer contains up to the
                  requested number of bytes, and the
                  variable BytesRead contains the
                  number of bytes that were actually
                  read into the buffer.  */
```

DosRead

Board Level
Device Level

DosRead
(Continued)

16-Bit

```
#define INCL_DOSFILEMGR
#include <os2.h>

UCHAR BufferArea[256];      /* User Buffer   */
USHORT BufferLength = 256;  /* Buffer Length */
USHORT BytesRead;


rc = DosRead  (bdh, BufferArea, BufferLength,
&BytesRead);
                /* On successful return, the user
                supplied buffer contains up to the
                requested number of bytes, and the
                variable BytesRead contains the
                number of bytes that were actually
                read into the buffer.*/
```

DosWrite

Board Level
Device Level

DosWrite

Purpose

Write data to a device from a user buffer.

Format

32-Bit

```
ULONG DosWrite ((HFILE) bdh, (PVOID) wrtbuf,
                (ULONG) BufferLength,
                (PULONG) AddrBytesWritten)
```

16-Bit

```
USHORT DosWrite((HFILE) bdh, (PVOID) wrtbuf,
                (USHORT) BufferLength,
                (PUSHORT) AddrBytesWritten)
```

Parameters

IN `bdh`	Device or board handle
IN `wrtbuf`	Buffer that contains data to write
IN `BufferLength`	Number of bytes to be written
OUT `AddrBytesWritten`	Address of variable to receive the number of bytes actually written
Function Return	Zero, if no error; OS/2 error value, otherwise.

DosWrite

Board Level
Device Level

DosWrite
(Continued)

Description

Device Level

If `bdh` is a device handle, this function addresses the GPIB and writes up to `BufferLength` bytes of data from the buffer pointed to by `wrtbuf` to a GPIB device. The operation terminates normally when `BufferLength` bytes have been sent. The operation terminates with an error if `BufferLength` bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the variable `AddrBytesWritten`.

Board Level

If `bdh` is a board handle, this function writes `BufferLength` bytes of data from the buffer pointed to by `wrtbuf` to a GPIB device; a board-level `DosWrite` assumes that the GPIB is already properly addressed. The operation terminates normally when `BufferLength` bytes have been sent. The operation terminates with an error if `BufferLength` bytes could not be sent within the timeout period or if the board is not CIC and a Device Clear is sent on the GPIB by the CIC. The actual number of bytes transferred is returned in the variable `AddrBytesWritten`.

Possible Errors

For information on the possible error codes that could be returned by the system when `DosWrite` fails, refer to the `DosWrite` function in the *OS/2 Technical Library*, *Control Program Programming Reference.*

DosWrite

Board Level
Device Level

DosWrite
(Continued)

Example

32-Bit

```
#define INCL_DOSFILEMGR
#include <os2.h>

UCHAR BufferArea = "hello world"; /* Data to write */
ULONG BufferLength = strlen(BufferArea); /* Number of
                                            bytes to
                                            write */
ULONG BytesWritten;

rc = DosWrite(bdh, BufferArea, BufferLength,
              &BytesWritten);
```

16-Bit

```
#define INCL_DOSFILEMGR
#include <os2.h>

UCHAR BufferArea = "hello world"; /* Data to write */
USHORT BufferLength = strlen(BufferArea); /* Number of
                                             bytes to
                                             write */
USHORT BytesWritten;


rc = DosWrite(bdh, BufferArea, BufferLength,
              &BytesWritten);
```

DosDevIOCtl

You can use the OS/2 API `DosDevIOCtl` function or the `nictl` function to perform the GPIB control functions to a device or board. `nictl` is a C macro that expands to the `DosDevIOCtl` function. This macro is only for the purpose of backward compatibility. National Instruments recommends that you always use the `DosDevIOCtl` function.

32-Bit DosDevIOCtl

The 32-bit control function, `DosDevIOCtl`, is used to perform GPIB specific board and device control functions. The 32-bit `DosDevIOCtl` function has nine parameters:

- A file handle returned by `DosOpen`
- A 32-bit category number (0 x 81)
- A 32-bit function code
- A `void` pointer to the parameter area
- Size of the parameter area in bytes
- Address of a variable that holds the size of the parameter area
- A `void` pointer to the data area
- Size of the data area in bytes
- Address of a variable that holds the size of the data area

The file handle identifies the board or device on which the control function is to be performed. The category number is used by the system to identify the device type. National Instruments uses the category number hex 81 to identify the GPIB API category. It remains the same for all GPIB API control functions. This category number is defined as a constant named `CATEGORY` in the include file `nicode.h`. The function code identifies the GPIB operation that is to be performed. The other parameters vary depending on the GPIB function. For more information on how to set up these parameters for each call, refer to the specific GPIB function you are using in this chapter.

16-Bit DosDevIOCtl

The 16-bit control function, `DosDevIOCtl`, is used to perform GPIB-specific board and device control functions. The `DosDevIOCtl` function has five parameters:

- A 32-bit pointer to a data area
- A 32-bit pointer to a parameter area
- A 16-bit function number
- A 16-bit category number (0 x 81)
- A file handle returned by `DosOpen`

The file handle identifies the board or device on which the control function is to be performed. The category number is used by the system to identify the device type. National Instruments uses the category number hex 81 to identify the GPIB API category. It remains the same for all GPIB API control functions. This category number is defined as a constant named `CATEGORY` in the include file `nicode.h`. The function code identifies the GPIB operation to be performed. The other parameters vary depending on the GPIB function. Refer to the specific GPIB function you are using in this chapter for more information on how to set up these parameters for each call.

32-Bit C

```
#include INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG DosDevIOCtl(HFILE, ULONG, ULONG,
                  PVOID, ULONG, PULONG,
                  PVOID, ULONG, PULONG)
```

16-Bit C

```
#include INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT DosDevIOCtl(PVOID,PVOID
                   USHORT, USHORT,
                   HFILE)
```

GPIB and OS/2 Device Errors

When the NI-488.2M driver detects an error condition for a GPIB Control function, it either returns a GPIB device error or an OS/2 system error to the calling function.

By default, the NI-488.2M driver is configured to return a GPIB-specific error if a GPIB control function fails. You can modify the driver configuration so that an OS/2 system error is returned in such cases. For more information on how to change this driver configuration, refer to board API functions `BTempRd`, `BTempRdNew`, `BTempWrt`, and `BTempWrtNew`, and device API functions `DTempRd`, `DTempRdNew`, `DTempWrt`, and `DTempWrtNew` described later in this chapter. Refer to the `DosDevIOCtl` function in the *OS/2 Technical Library, Control Program Programming Reference,* for more information on OS/2 system error codes. If the NI-488.2M driver is configured to return GPIB-specific errors and if an OS/2 API Control function fails, the lower byte of the return value contains the GPIB-specific error code. For efficient debugging purposes, National Instruments suggests that the driver always remain configured to return GPIB-specific errors. Refer to Table 3-5 for more information on the different GPIB error codes and their descriptions.

File I/O functions (`DosOpen`, `DosClose`, `DosRead`, and `DosWrite`) cannot return GPIB-specific error codes. If a file I/O function fails, use the function `STATUS` to get the status and error information. The status word that gets returned contains 16 bits, all of which are meaningful. A bit value of one (1) indicates that the corresponding condition is in effect. A bit value of zero (0) indicates that the condition is not in effect. Table 3-3 lists the conditions and the bit position to test for each condition in `ibsta`. Some bits are valid only on device functions (`dev`); some bits are valid only on board functions (`brd`), and some bits are set on either type (`dev`, `brd`).

Table 3-3. Status Word (ibsta) Layout

Mnemonics	Bit Pos.	Hex Value	Function Type	Description
BERR	15	8000	dev, brd	GPIB error
BTIMO	14	4000	dev, brd	Time limit exceeded
BEND	13	2000	dev, brd	END or EOS detected
BSRQI	12	1000	brd	SRQ interrupt received
BRQS	11	800	dev	Device requesting service
BCMPL	8	100	dev, brd	I/O completed
BLOK	7	80	brd	Lockout State
BREM	6	40	brd	Remote State
BCIC	5	20	brd	Controller-In-Charge
BATN	4	10	brd	Attention is asserted
BTACS	3	8	brd	Talker
BLACS	2	4	brd	Listener
BDTAS	1	2	brd	Device Trigger State
BDCAS	0	1	brd	Device Clear State

Table 3-4 defines OS/2 system errors for the NI-488.2M driver for GPIB.

Table 3-4. OS/2 System Errors for GPIB

Code	Mnemonics
1	Invalid Function
6	Invalid Handle
31	General Failure
87	Invalid Parameter
117	Invalid Category

GPIB Error Codes

Table 3-5 lists the GPIB error codes. Remember that the error variable is meaningful only when the ERR bit in the status variable is set. For a detailed description of each error and possible solutions, refer to Appendix C, *Error Codes and Solutions*.

Table 3-5. GPIB Error Codes

Error Mnemonic	iberr Value	Meaning
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
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
ECAP	11	No capability for operation
EBUS	14	GPIB bus error
ESTB	15	Serial poll status byte queue overflow
ESRQ	16	SRQ stuck in ON position

Examining The Error

When an error occurs after a `DosDevIOCtl` API function call, the error code is in the lower eight bits of the return value. If no error occurs, the return value is zero.

To determine if an error has occurred, test whether the return value is non-zero by entering the following statement:

```
rc = DosDevIOCtl ( param1, . . .);
if (rc) {
  error = (rc&0XFF);
}
```

Writing Your API Application

The following section outlines some considerations when developing an application using the API functions.

Items to Include

The file `nicode.h` in your API directory contains GPIB-related constant and function definitions that you need to use in your GPIB application program. Include the declaration file as follows:

```
#include "nicode.h"
```

If you wish to use the OS/2 `nictl` function instead of the `DosDevIOCtl` function, you need to include the file `nictl_xx.h`. This file contains the macro that expands the `nictl` function to the `DosDevIOCtl` function. Include the macro file appropriate for your compiler as follows:

```
#include "nictl_32.h" /* 32-bit C compiler */
#include "nictl_16.h" /* 16-bit C compiler */
```

Compiling and Linking Your Program

After you have written your application program, you need to compile your program and link it as follows.

32-bit C Applications

Compile your 32-bit program using the following command:

```
icc /c apiprog.c
```

Then enter the following command to link your compiled program:

```
link386 /NOI apiprog.obj,apiprog.exe;
```

16-bit C Applications

Compile your 16-bit program using the following command:

```
cl /c apiprog.c
```

Then enter the following command to link your compiled program:

```
link /NOI apiprog.obj,apiprog.exe;
```

OS/2 Control Function Descriptions

The following section is an alphabetical listing of each OS/2 control function.

ATNOFF **Board Level** ATNOFF

Purpose

Go from Active Controller to Standby.

Format

32-Bit

```
ULONG DosDevIOCtl ((HFILE) bdh, CATEGORY, ATNOFF,
                    NULL, (ULONG) ParmLength,
                   (PULONG) AddrParmLength,
                    NULL, (ULONG) DataLength,
                   (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, ATNOFF, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero if no error; GPIB-specific error value otherwise.

Description

The GPIB board goes to Standby Controller and the GPIB Attention (ATN) line is unasserted. No acceptor handshaking or holdoff is performed.

ATNOFF **Board Level** ATNOFF **(Continued)**

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.
ECIC - The interface board is not Controller-In-Charge.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
# define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh,CATEGORY, ATNOFF,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
# define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


rc = DosDevIOCtl   (NULL, NULL, ATNOFF, CATEGORY, bdh);
```

ATNON Board Level ATNON

Purpose

Become Active Controller.

Format

32-Bit

```
ULONG DosDevIOCtl ((HFILE) bdh, CATEGORY, ATNON,
                       NULL, (ULONG) ParmLength,
                      (PULONG) AddrParmLength,
                       NULL, (ULONG) DataLength,
                      (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, ATNON, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The designated GPIB board attempts to become the active controller by asserting ATN. The GPIB 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 `ATNON`. Functions that require ATN to be asserted—for example, `CMD`—do so automatically.

ATNON — Board Level — ATNON (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.
ECIC - The interface board is not Controller-In-Charge.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh,CATEGORY, ATNON,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


rc = DosDevIOCtl (NULL, NULL, ATNON, CATEGORY, bdh)
```

BConfRd **Board Level** BConfRd

Purpose

Return information about the initial configuration parameters of the board.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, BConfRd,
                   (PVOID) &nib, (ULONG) ParmLength,
                   (PULONG) AddrParmLength,
                    NULL, (ULONG) DataLength,
                   (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib, BConfRd, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
OUT `nib`	Board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BConfRd` function returns the initial values of various board configuration parameters. Once these parameters are read, they can be modified using the `BConfWrt` function.

BConfRd — Board Level — BConfRd (Continued)

The board configuration structure `nibrd`, defined in the `nicode.h` include file, is used to read the board configuration parameters. Table 3-6 describes the `nibrd` structure.

Table 3-6. Board Configuration Structure

Field	Description
uflags	Board configuration flags
pad	Primary address of the board
sad	Secondary address of the board
eos	End-of-string character
timo	Minimum timeout for board level operations
rbase	Base address of board
dmaCH	DMA channel used by the board

BConfRd **Board Level** **BConfRd (Continued)**

Table 3-7 describes the field `uflags` (Board Configuration flags) in `nibrd` structure.

Table 3-7. Board Configuration Flags

Flag	Hex Code	Description
UEOT	0x1	If set, EOI is asserted at the end of a write operation. If not set, EOI is not asserted at the end of a write operation.
UREOS	0x4	If set, the EOS character terminates the read operation. If not set, the EOS character is ignored during read operations.
UWEOS	0x8	If set, the EOI line is asserted when the EOS character is sent during a write operation. If not set, the EOI line is not asserted when the EOS character is sent during a write operation.
UCMP8	0x10	If set, an 8-bit compare is used for all EOS comparisons. If not set, a 7-bit compare is used for all EOS comparisons.
UGPIB	0x400	If set, GPIB-specific errors are returned when OS/2 API control functions fail. If not set, system error codes are returned when OS/2 API control functions fail.
UDMA	0x2000	If set, the board uses DMA for GPIB transfers. If not set, the board does not use DMA for GPIB transfers.
USC	0x800	If set, the board is System Controller. If not set, the board is not GPIB System Controller.

BConfRd **Board Level** BConfRd (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh,CATEGORY, BConfRd,
                    (PVOID) &nib, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;
rc = DosDevIOCtl   (NULL, &nib, BConfRd, CATEGORY, bdh);
```

BConfRdNew **Board Level** BConfRdNew

Purpose

Return information about the default configuration parameters of the GPIB board.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,BConfRdNew,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib,BConfRdNew,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
OUT `nib`	New board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BConfRdNew` function returns the default values of various GPIB board configuration parameters. Once these parameters are read, they can be modified using the `BConfWrtNew` function.

BConfRdNew Board Level BConfRdNew (Continued)

You can use the new board configuration structure `new_nibrd`, defined in the `nicode.h` include file, to read the board configuration parameters. Table 3-8 describes the `new_nibrd` structure.

Table 3-8. Board Configuration Structure

Field	Type	Description
Brd_Size	ushort	Size of `new_nibrd` structure
State	ulong	Board configuration flags
BoardBase**	ushort	Base address of the board
Pad	uchar	Primary address of the device
Sad	uchar	Secondary address of the device
DmaChannel**	uchar	DMA channel used by the board - to disable DMA, use the UDMA configuration flag
IrqLevel**	uchar	Interrupt level used by the board - to disable interrupts, use the UIRQ configuration flag
BlockTime*	uchar	Block time for non-interrupt I/O
PPollEnableByte*	uchar	Parallel poll enable/disable value
PPollLengthTime	uchar	Parallel poll duration
Timeout	uchar	Maximum timeout for board-level operations
EOSbyte	uchar	End-of-string character
Timing	uchar	1 = Normal timing (T1 delay = 2 µs) 2 = High speed timing (T1 delay = 500 ns) 3 = Very high speed timing (T1 delay = T1 delay is determined by GPIB Source Handshake timing.
HSCableLength	ushort	Reserved for future revision of NI-488.2M driver
* Available through `BTempRdNew` and `BTempWrtNew` only.		
** Available through `BConfRdNew` and `BConfWrtNew` only.		

BConfRdNew — Board Level — BConfRdNew (Continued)

Note: `Brd_Size` ***should be set to the size of the*** `new_nibrd` ***structure before calling*** `BConfRdNew`**,** `BConfWrtNew`**,** `BTempRdNew` ***or*** `BTempWrtNew`**.** ***If*** `Brd_Size` ***is not specified, these calls return an EARG error.***

With `BConfRdNew`, you can read only the default values of a particular board that are configurable using ibconf. You cannot use `BConfRdNew` to read `BlockTime` or `PPollEnableByte`.

On Micro Channel systems, you cannot change the `BoardBase`, `DmaChannel`, or `IrqLevel`. However, you can enable or disable the use of DMA and interrupts using the board configuration flags `UDMA` and `UIRQ`.

Table 3-9 gives a detailed description of the board configuration flags in the `State` field of the `new_nibrd` structure.

Table 3-9. Board Configuration Flags for New API Calls

Flag	Hex Code	Description
UAUTOPOLL	0x4000	If set, automatic serial polling is enabled. If not set, automatic serial polling is disabled.
UCICPROT	0x8000	If set, CIC protocol is enabled. If not set, CIC protocol is disabled.
UCMP8	0x0010	If set, an 8-bit compare is used for all EOS comparisons. If not set, a 7-bit compare is used for all EOS comparisons.
UDMA	0x2000	If set, the board uses DMA for GPIB transfers. If not set, the board does not use DMA for GPIB transfers.

(continues)

Table 3-9. Board Configuration Flags for New API Calls (Continued)

Flag	Hex Code	Description
UEOT	0x0001	If set, EOI is asserted at the end of a write operation. If not set, EOI is not asserted at the end of a write operation.
UGPIB	0x0400	If set, GPIB-specific errors are returned when OS/2 API control functions fail. If not set, system error codes are returned when OS/2 API control functions fail.
UIRQ	0x1000	If set, the board uses interrupts. If not set, the board does not use interrupts.
UNOENDBIT*	0x0080	If set, the END bit of status in a STATUS call is not set when an EOS match occurs during a read. If not set, the END bit of status in a STATUS call is set when an EOS match occurs during a read.
UPP2*	0x80000	If set, PP2 mode (local parallel poll configuration). If not set, PP1 mode (remote parallel poll configuration).
URDSWAP*	0x10000	If set, pairs of bytes are swapped during a read operation. If not set, no byte swapping is performed during a read operation.

(continues)

Table 3-9. Board Configuration Flags for New API Calls (Continued)

Flag	Hex Code	Description
UREOS	0x0004	If set, the EOS character terminates the read operation. If not set, the EOS character is ignored during read operations.
USC	0x0800	If set, the board is GPIB System Controller. If not set, the board is not GPIB System Controller.
USEBRD	0x100000	If set, the driver accesses this board. If not set, the driver does not access this board.
USENDLLO*	0x40000	If set, LLO is sent when placing a device online. If not set, LLO is not sent when placing a device online.
USRE	0x0100	If set, the Remote Enable (REN) line is asserted when the board is System Controller. If not set, the Remote Enable (REN) line is not asserted when the board is System Controller.
UWEOS	0x0008	If set, the EOI line is asserted when the EOS character is sent during a write operation. If not set, the EOI line is not asserted when the EOS character is sent during a write operation.
UWRTSWAP*	0x20000	If set, pairs of bytes are swapped during a write operation. If not set, no byte swapping is performed during a write operation.
* Available through `BTempRdNew` and `BTempWrtNew` only.		

BConfRdNew **Board Level** BConfRdNew (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;
nib.Brd_Size = sizeof(nib)

rc = DosDevIOCtl   (bdh,CATEGORY, BConfRdNew,
                   (PVOID) &nib, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;
nib.Brd_Size = sizeof (nib)
rc = DosDevIOCtl  (NULL, &nib, BConfRdNew, CATEGORY,
                   bdh);
```

BConfWrt **Board Level** BConfWrt

Purpose

Modify initial configuration parameters of the board.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,BConfWrt,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib, BConfWrt, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `nib`	Board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BConfWrt` function modifies the initial values of various board configuration parameters. Changes go into effect when the board is placed online. These configuration changes are in effect until the system is restarted.

BConfWrt **Board Level** BConfWrt **(Continued)**

The board configuration structure `nibrd`, defined in the `nicode.h` include file, is used to read the board configuration parameters. Refer to Tables 3-6 and 3-7 for a description of the `nibrd` structure and board configuration flags, respectively.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to an interface board or one of the parameters set in the `nibrd` structure is invalid.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;

rc = DosDevIOCtl  (bdh,CATEGORY, BConfRd,
                   &nib, ParmLength, &ParmLength,
                   NULL, DataLength, &DataLength);
if (rc != 0)  {
    nib.pad = 1 ;  /* change the primary address to one */
    nib.uflags &= ~UDMA;  /* Do not use DMA for I/O
                             transfers */


    rc = DosDevIOCtl  (bdh,CATEGORY, BConfWrt,
                       &nib, ParmLength, &ParmLength,
                       NULL, DataLength, &DataLength);
}
```

BConfWrt — Board Level — BConfWrt (Continued)

16-Bit

```
# define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;

rc = DosDevIOCtl (NULL, &nib, BConfRd, CATEGORY, bdh);
if (rc != 0)  {
    nib.pad = 1 ;  /* change the primary address to one */
    nib.uflags &= ~UDMA; /* Do not use DMA
                            for I/O transfers */
    rc = DosDevIOCtl  (NULL, &nib, BConfWrt, CATEGORY,
    bdh);
}
```

BConfWrtNew **Board Level** BConfWrtNew

Purpose

Modify default values of the board configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,BConfWrtNew,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib,BConfWrtNew,
                     CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `nib`	New board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BConfWrtNew` function modifies the initial values of various board configuration parameters. Changes take effect when the board is placed online and remain in effect until the system is restarted.

The board configuration structure `new_nibrd`, defined in the `nicode.h` include file, is used to modify the board configuration parameters.

BConfWrtNew **Board Level** BConfWrtNew (Continued)

Refer to Tables 3-8 and 3-9 for a description of the `new_nibrd` structure and board configuration flags.

With `BConfWrtNew`, you can modify only the default values of a particular board that are configurable using `ibconf`. You cannot use `BConfWrtNew` to modify `PPollEnableByte`.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to an interface board, or one of the parameters set in the `new_nibrd` structure is invalid.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;

nib.Brd_Size = sizeof (nib);
rc = DosDevIOCtl  (bdh,CATEGORY, BConfRdNew,
                   &nib, ParmLength, &ParmLength,
                   NULL, DataLength, &DataLength);
if (rc != 0)  {
    nib.Pad = 1 ;/* change the primary address to one  */
    nib.State &= ~UAUTOPOLL;  /* Do not use autopoll   */


    rc = DosDevIOCtl  (bdh,CATEGORY, BConfWrtNew,
                       &nib, ParmLength, &ParmLength,
                       NULL, DataLength,&DataLength);
}
```

BConfWrtNew Board Level BConfWrtNew (Continued)

16-Bit

```
# define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;

nib.Brd_Size = sizeof (nib);
rc = DosDevIOCtl(NULL,&nib,BConfRdNew,CATEGORY,bdh);
if (rc != 0)  {
    nib.Pad = 1 ;  /* change the primary address to one */
    nib.State &= ~UAUTOPOLL; /* Do not use autopoll */

    rc = DosDevIOCtl (NULL, &nib, BConfWrtNew,
                      CATEGORY, bdh);
}
```

BLOCAL **Board Level** BLOCAL

Purpose

Go to local.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, BLOCAL,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, BLOCAL, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

If the board is not in a lockout state (`LOK` does not appear in the status word), this function places the board in local mode. Otherwise, the call has no effect. The `BLOCAL` function can simulate a front panel RTL (Return to Local) switch if the computer is used as an instrument.

BLOCAL — Board Level — BLOCAL (Continued)

Possible GPIB-Specific Errors

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

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, BLOCAL,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, BLOCAL, CATEGORY, bdh);
```

BTempRd **Board Level** BTempRd

Purpose

Return information about the current configuration parameters of the board.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,BTempRd,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib, BTempRd,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
OUT `nib`	Board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BTempRd` function returns the current values of various board configuration parameters. Once these parameters are read, they can be modified using the `BTempWrt` function.

The board configuration structure `nibrd`, defined in the `nicode.h` include file, is used to read the board configuration parameters. Refer to Tables 3-6 and 3-7 for a description of the `nibrd` structure and board configuration flags, respectively.

BTempRd **Board Level** BTempRd (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, BTempRd,
                    &nib, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;

rc = DosDevIOCtl   (NULL, &nib, BTempRd, CATEGORY, bdh);
```

BTempRdNew **Board Level** BTempRdNew

Purpose

Return current values of the board configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,BTempRdNew,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib,BTempRdNew,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
OUT `nib`	New board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BTempRdNew` function returns the current values of various board configuration parameters that are set after the board is placed online. Once these parameters are read, they can be modified using the `BTempWrtNew` function.

BTempRdNew — Board Level — BTempRdNew (Continued)

The board configuration structure `new_nibrd`, defined in the `nicode.h` include file, is used to modify the board configuration parameters. Refer to Tables 3-8 and 3-9 for a description of the `new_nibrd` structure and board configuration flags.

With `BTempRdNew`, you can read `PPollEnableByte`.

Possible GPIB-Specific Errors

EARG - `bdh` is valid but does not refer to an interface board.
ENEB - The interface board is not installed or configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;

nib.Brd_Size = sizeof (nib);
rc = DosDevIOCtl   (bdh,CATEGORY, BTempRdNew,
                    &nib, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;

nib.Brd_Size = sizeof (nib);
rc = DosDevIOCtl  (NULL, &nib, BTempRdNew, CATEGORY, bdh);
```

BTempWrt Board Level BTempWrt

Purpose

Modify current configuration parameters of the board.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,BTempWrt,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nib, BTempWrt, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `nib`	Board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BTempWrt` function dynamically modifies the current value of various board configuration parameters. Changes remain in effect until the board is placed offline.

The board configuration structure `nibrd`, defined in the `nicode.h` include file, is used to read the board configuration parameters. Refer to Tables 3-6 and 3-7 for a description of the `nibrd` structure and board configuration flags, respectively.

BTempWrt **Board Level** BTempWrt (Continued)

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to an interface board or one of the parameters set in the `nibrd` structure is invalid.

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

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh,CATEGORY, BTempRd,
                    &nib, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
if (rc != 0)  {
    nib.pad = 1 ;  /* change primary address to one */
    nib.uflags &= ~UDMA; /* Do not use DMA
                            for I/O transfers */

    rc = DosDevIOCtl  (bdh,CATEGORY, BTempWrt,
                       &nib, ParmLength, &ParmLength,
                       NULL, DataLength, &DataLength);
}
```

BTempWrt **Board Level** BTempWrt (Continued)

16-Bit

```
# define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nibrd nib;

rc = DosDevIOCtl   (NULL, &nib, BTempRd, CATEGORY,
                    bdh)

if (rc != 0)  {
    nib.pad = 1 ;  /* change the primary address to one */
    nib.uflags &= ~UDMA; /* Do not use DMA for
                            I/O transfers */
    rc = DosDevIOCtl (NULL, &nib, BTempWrt, CATEGORY,
                      bdh);
}
```

BTempWrtNew **Board Level** BTempWrtNew

Purpose

Modify current values of the board configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh,CATEGORY,BTempWrtNew,
                    (PVOID) &nib, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID)&nib,BTempWrtNew,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `nib`	New board configuration structure
IN `ParmLength`	Size of `nib`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `BTempWrtNew` function dynamically modifies the current values of various board configuration parameters that are set after the board is placed online. Changes remain in effect until the board is placed offline.

The board configuration structure `new_nibrd`, defined in the `nicode.h` include file, is used to read the board configuration parameters. Refer to Tables 3-8 and 3-9 for a description of the `new_nibrd` structure and board configuration flags.

BTempWrtNew Board Level BTempWrtNew (Continued)

You cannot use `BTempRdNew` to modify `BoardBase`, `DmaChannel`, or `IrqLevel`. If you need to change those options, use `BConfWrtNew`. You can use `BTempWrtNew` to modify `PPollEnableByte`.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to an interface board or one of the parameters set in the `new_nibrd` structure is invalid.
ENEB - The interface board is not installed or configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;
ULONG ParmLength = sizeof (nib);
ULONG DataLength = 0;


nib.Brd_Size = sizeof (nib);
rc = DosDevIOCtl   (bdh,CATEGORY, BTempRdNew,
                    &nib, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
if (rc != 0)  {
    nib.Pad = 1 ;  /* change primary address to one */
    nib.State &= ~UAUTOPOLL; /* Do not use autopoll */

    rc = DosDevIOCtl  (bdh,CATEGORY, BTempWrtNew,
                       &nib, ParmLength, &ParmLength,
                       NULL, DataLength,&DataLength);
}
```

BTempWrtNew **Board Level** BTempWrtNew **(Continued)**

16-Bit

```
# define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nibrd nib;

nib.Brd_Size = sizeof (nib);
rc = DosDevIOCtl(NULL,&nib,BTempRdNew,CATEGORY,bdh)

if (rc != 0)  {
    nib.Pad = 1 ;  /* change the primary address to one */
    nib.State &= ~UAUTOPOLL; /* Do not use autopoll*/

    rc = DosDevIOCtl (NULL, &nib, BTempWrtNew,
                      CATEGORY, bdh);
}
```

BWAIT

Board Level
Device Level

BWAIT

Purpose

Wait for GPIB events.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, BWAIT,
                    (PVOID)&mask, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &mask, BWAIT,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle
IN `mask`	Bit mask of GPIB events to wait for
IN `ParmLength`	Size of `mask`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

BWAIT

Board Level
Device Level

BWAIT
(Continued)

Description

This function monitors the events selected in `mask` and delays processing until one or more occur. If TIMO is set in `mask`, `BWAIT` returns when the timeout period has elapsed if any of the other specified events have not already occurred. If TIMO is not set in `mask`, the function will wait indefinitely until one of the specified events occurs. The `BWAIT mask` bits are described in Table 3-10. If `bdh` is a device handle, the only valid wait mask bits are `BTIMO`, `BEND`, `BRQS`, and `BCMPL`. If `bdh` is a board handle, all wait mask bits are valid except for `BRQS`.

Possible GPIB-Specific Errors

EARG - The bit set in `mask` is invalid.
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-488.2M Software Reference Manual for OS/2.*
ENEB - The interface board is either not installed or not configured properly.
ESRQ - Device level: If RQS is set in the wait mask then ESRQ indicates that the *Stuck SRQ* condition exists. See the *NI-488.2M Software Reference Manual* for more information.

BWAIT **Board Level** **Device Level** **BWAIT (Continued)**

Table 3-10. Wait Mask Layout

Mnemonic	Bit Pos.	Hex Values	Description
BERR	15	8000	GPIB error
BTIMO	14	4000	Time limit exceeded
BEND	13	2000	GPIB board detected END or EOS
BSRQI	12	1000	SRQ on
BRQS	11	800	Device requesting service
BCMPL	8	100	Asynchronous I/O completed
BLOK	7	80	GPIB board is in lockout state
BREM	6	40	GPIB board is in remote state
BCIC	5	20	GPIB board is CIC
BATN	4	10	Attention is asserted
BTACS	3	8	GPIB board is Talker
BLACS	2	4	GPIB board is Listener
BDTAS	1	2	GPIB board is in device trigger state
BDCAS	0	1	GPIB board is in Device Clear state

BWAIT

Board Level
Device Level

BWAIT
(Continued)

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT mask ;
ULONG ParmLength = sizeof (mask);
ULONG DataLength = 0;

mask = BTIMO | BDCAS ;
rc = DosDevIOCtl   (bdh,CATEGORY, BWAIT,
                    &mask, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"
USHORT mask ;

mask = BTIMO | BDCAS ;
rc = DosDevIOCtl(NULL, &mask, BWAIT, CATEGORY, bdh);
```

CLRREMOTE Board Level CLRREMOTE

Purpose

Clear the Remote Enable line.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, CLRREMOTE,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, CLRREMOTE, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLength`	Length of parameter area; must be set to zero
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

`CLRREMOTE` unasserts the Remote Enable (REN) signal. REN is used by devices to select between local and remote modes of operation. A device does not actually enter remote mode until it receives its listen address.

CLRREMOTE Board Level CLRREMOTE (Continued)

Possible GPIB-Specific Errors

EARG - bdh is a valid handle but does not refer to a board.
ENEB - The interface board is either not installed or not configured properly.
ESAC - The board does not have System Controller capability.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


ULONG ParmLength = 0;
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh,CATEGORY, CLRREMOTE,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


rc = DosDevIOCtl   (NULL, NULL, CLRREMOTE, CATEGORY,
                    bdh);
```

CMD Board Level CMD

Purpose

Send GPIB commands.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, CMD,
                    (PVOID) &cnt, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                    (PVOID) cmdbuf, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl ((PVOID) cmdbuf, (PVOID) &cnt, CMD,
                     CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `cnt`	Number of command bytes to send
IN `ParmLength`	Size of `cnt`
IN `AddrParmLength`	Address of `ParmLength`
IN `cmdbuf`	Buffer of command bytes to send
IN `DataLength`	Size of `cmdbuf`
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

`cnt` bytes from `cmdbuf` are sent over the GPIB as command bytes (interface messages). You can find a table of the defined Interface Messages in Appendix A, *Multiline Interface Messages.*

CMD — Board Level — CMD (Continued)

Command bytes are used to configure the state of the GPIB; they are not used to send instructions to GPIB devices. Use `DosWrite` to send device-specific instructions.

Possible GPIB-Specific Errors

EABO - The timeout period expired before all of the command bytes were sent.
EARG - `bdh` is valid but does not refer to an interface board.
ECIC - The interface board is not Controller-In-Charge.
ENEB - The interface board is either not installed or not configured properly.
ENOL - There are no Listeners on the GPIB.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT cnt;
char CmdBuf[3];
ULONG ParmLength = sizeof(cnt);
ULONG DataLength = sizeof(CmdBuf);

CmdBuf = "?@!";
cnt = 3;

rc = DosDevIOCtl   (bdh,CATEGORY, CMD,
                    &cnt, ParmLength, &ParmLength,
                    CmdBuf, DataLength, &DataLength);
```

CMD

Board Level

CMD
(Continued)

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT cnt;
char CmdBuf[3];


CmdBuf = "?@!";
cnt = 3;
rc = DosDevIOCtl   (CmdBuf, &cnt, CMD, CATEGORY, bdh);
```

DCLEAR Device Level DCLEAR

Purpose

Clear a specific device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, DCLEAR,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, DCLEAR, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of Data Area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`.
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The Selected Device Clear (SDC) message is sent to the device described by `bdh`.

DCLEAR Device Level DCLEAR (Continued)

Possible GPIB-Specific Errors

EARG - bdh is a valid handle 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 the *NI-488.2M Software Reference Manual for OS/2.*
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh,CATEGORY, DCLEAR,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


rc = DosDevIOCtl   (NULL, NULL,CATEGORY, DCLEAR, bdh);
```

DConfRd Device Level DConfRd

Purpose

Return information about the initial configuration parameters of the device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, DConfRd,
                    (PVOID)&nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DConfRd,
CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
OUT `nid`	Device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DConfRd` function returns the initial values of various device configuration parameters. Once these parameters are read, you can modify them using the `DConfWrt` function.

DConfRd — Device Level — DConfRd (Continued)

The device configuration structure `nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Tables 3-11 and 3-12 describe the `nidev` structure and the device configuration flags, respectively.

Table 3-11. Device Configuration Structure

Field	Description
uflags	Device configuration flags
brdno	Index of device's access board
pad	Primary address of the device
sad	Secondary address of the device
eos	Device end-of-character string
tmo	Maximum timeout for device-level operations

Table 3-12. Device Configuration Flags

Flag	Hex Code	Description
UEOT	1	If set, EOI is asserted at the end of a write operation. If not set, EOI is not asserted at the end of a write operation.
UREOS	4	If set, the EOS character terminates the read operation. If not set, the EOS character is ignored during read operations.
UWEOS	8	If set, the EOI line is asserted when the EOS character is sent during a write operation. If not set, the EOI line is not asserted when the EOS character is sent during a write operation.
UCMP8	10	If set, an 8-bit compare is used for all EOS comparisons. If not set, a 7-bit compare is used for all EOS comparisons.
UGPIB	400	If set, GPIB-specific errors are returned when OS/2 API control functions fail. If not set, system error codes are returned when OS/2 API control functions fail.
UNAD	20	If set, the Untalk (UNT) and Unlisten (UNL) commands are sent after each device-level read and write operation. If not set, the UNT and UNL commands are not sent after each device-level read and write operation.

DConfRd **Device Level** DConfRd (Continued)

Possible GPIB-Specific Errors

EARG - `bdh` is valid but does not refer to a device.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, DConfRd,
                    &nid, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;
rc = DosDevIOCtl   (NULL, &nid, DConfRd, CATEGORY, bdh);
```

DConfRdNew Device Level DConfRdNew

Purpose

Return default values of the device configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, DConfRdNew,
                    (PVOID)&nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DConfRdNew,
                    CATEGORY,(HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
OUT `nid`	New device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

DConfRdNew **Device Level** DConfRdNew (Continued)

Description

The `DConfRdNew` function returns the default values of various device configuration parameters. Once these parameters are read, you can modify them using the `DConfWrtNew` function.

The device configuration structure `new_nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Tables 3-13 and 3-14 give a description of the `new_nidev` structure and device configuration flags in the `State` field of the `new_nidev` structure.

Table 3-13. Device Configuration Structure

Field	Type	Description
Dev_Size	ushort	Size of `new_nidev` structure
State	ulong	Device configuration flags
AccessBoard	uchar	Index of device's access board
Pad	uchar	Primary address of the device
Sad	uchar	Secondary address of the device
EOSbyte	uchar	Device end-of-character string
Timeout	uchar	Maximum timeout for device-level operations
SPollTimeout	uchar	Maximum timeout for serial poll of a device

Note: `Dev_Size` ***should be set to the size of the*** `new_nidev` ***structure before calling*** `DConfRdNew`**,** `DConfWrtNew`**,** `DTempRdNew` ***and*** `DTempWrtNew`**.** ***If*** `Dev_Size` ***is not specified, these calls return an EARG error.***

DConfRdNew **Device Level** **DConfRdNew (Continued)**

Table 3-14. Device Configuration Flags for New API Calls

Flag	Hex Code	Description
UCMP8	0x0010	If set, an 8-bit compare is used for all EOS comparisons. If not set, a 7-bit compare is used for all EOS comparisons.
UEOT	0x0001	If set, EOI is asserted at the end of a write operation. If not set, EOI is not asserted at the end of a write operation.
UGPIB	0x0400	If set, GPIB-specific errors are returned when OS/2 API control functions fail. If not set, system error codes are returned when OS/2 API control functions fail.
UNAD	0x0020	If set, the Untalk (UNT) and Unlisten (UNL) commands are sent after each device-level read and write operation. If not set, the UNT and UNL commands are not sent after each device-level read and write operation.
UNOENDBIT*	0x0080	If set, the END bit of status in a STATUS call is not set when as EOS match occurs during a read. If not set, the END bit of status in a STATUS call is set when an EOS match occurs during a read.
URDSWAP*	0x10000	If set, pairs of bytes are swapped during a read operation. If not set, no byte swapping is performed during a read operation.

(continues)

Table 3-14. Device Configuration Flags for New API Calls (Continued)

Flag	Hex Code	Description
UREADDR	0x0040	If set, addressing is always performed before every device-level read and write. If not set, no unnecessary addressing is performed between device-level reads and writes.
UREOS	0x0004	If set, the EOS character terminates the read operation. If not set, the EOS character is ignored during read operations.
UWEOS	0x0008	If set, the EOI line is asserted when the EOS character is sent during a write operation. If not set, the EOI line is not asserted when the EOS character is sent during a write operation.
UWRTSWAP*	0x20000	If set, pairs of bytes are swapped during a write operation. If not set, no byte swapping is performed during a write operation.
* Available through `DTempRdNew` and `DTempWrtNew` only.		

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to a device or one of the parameters set in the `new_nidev` structure is invalid.

DConfRdNew — Device Level — DConfRdNew (Continued)

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;

nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl   (bdh,CATEGORY, DConfRdNew,
                    &nid, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;
nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl   (NULL,&nid,DConfRdNew,CATEGORY,bdh);
```

DConfWrt **Device Level** DConfWrt

Purpose

Modify initial configuration parameters of the device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,DConfWrt,
                    (PVOID) &nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DConfWrt, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `nid`	Device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DConfWrt` function modifies the initial values of various device configuration parameters. Changes go into effect when the device goes online and remain in effect until the system is restarted.

DConfWrt — Device Level — DConfWrt (Continued)

The device configuration structure `nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Tables 3-11 and 3-12 describe the `nidev` structure and the device configuration flags, respectively.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to a device or one of the parameters set in the `nidev` structure is invalid.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;


rc = DosDevIOCtl  (bdh,CATEGORY, DConfRd,
                   &nid, ParmLength, &ParmLength,
                   NULL, DataLength, &DataLength);
if (rc != 0)  {
    nid.pad = 1 ; /* change the primary address to one */
    nib.uflags &= ~UNAD;  /* Do not unaddress the bus after
                             I/O transfers */

    rc = DosDevIOCtl   (bdh,CATEGORY, DConfWrt,
                        &nid, ParmLength, &ParmLength,
                        NULL, DataLength, &DataLength);
}
```

DConfWrt — Device Level — DConfWrt (Continued)

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;

rc = DosDevIOCtl   (NULL, &nid, DConfRd, CATEGORY, bdh);
if (rc != 0)  {
    nid.pad = 1 ;  /* change the primary address to one */
    nid.uflags &= ~UNAD; /* Do not unaddress the bus after
                            I/O transfers */

    rc = DosDevIOCtl  (NULL, &nid, CATEGORY, DConfWrt,
    bdh);

}
```

DConfWrtNew Device Level DConfWrtNew

Purpose

Modify default values of device configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,DConfWrtNew,
                    (PVOID) &nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DConfWrtNew,
                    CATEGORY,(HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `nid`	New device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DConfWrtNew` function modifies the default values of a particular device. Changes take effect when the device is placed online and remain in effect until the system is restarted.

DConfWrtNew Device Level DConfWrtNew (Continued)

The device configuration structure `new_nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Refer to Tables 3-13 and 3-14 for a description of the `new_nidev` structure and device configuration parameters.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to a device or one of the parameters set in the `new_nidev` structure is invalid.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;


nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl  (bdh,CATEGORY, DConfRdNew,
                   &nid, ParmLength, &ParmLength,
                   NULL, DataLength, &DataLength);
if (rc != 0)  {
    nid.Pad = 1 ; /* change the primary address to one */
    nid.State &= ~UNAD;  /* Do not unaddress the bus
                            after I/O transfers */

    rc = DosDevIOCtl(bdh,CATEGORY, DConfWrtNew,
                     &nid, ParmLength,&ParmLength,
                     NULL,DataLength,&DataLength);
}
```

DConfWrtNew Device Level DConfWrtNew (Continued)

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;

nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl   (NULL,&nid,DConfRdNew,CATEGORY,bdh);
if (rc != 0)  {
    nid.Pad = 1 ; /* change the primary address to one */
    nid.State &= ~UNAD; /* Do not unaddress the bus
                           after I/O transfers */

    rc = DosDevIOCtl (NULL, &nid,CATEGORY,DConfWrtNew,
    bdh);

}
```

DLOCAL Device Level DLOCAL

Purpose

Go to Local.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, DLOCAL,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, DLOCAL, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

Unless the Remote Enable line has been unasserted with the `CLRREMOTE` function, all device level functions automatically place the specified device in remote program mode. `DLOCAL` 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.

DLOCAL Device Level DLOCAL (Continued)

Possible GPIB-Specific Errors

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 of the *NI-488.2M Software Reference Manual for OS/2.*
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, DLOCAL,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, DLOCAL, CATEGORY, bdh);
```

DTempRd Device Level DTempRd

Purpose

Return information about current configuration parameters of the device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, DTempRd,
                    (PVOID) &nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DTempRd,
CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
OUT `nid`	Device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DTempRd` function returns the current values of various device configuration parameters. Once these parameters are read, they can be modified by using the `DTempWrt` function.

DTempRd **Device Level** DTempRd (Continued)

The device configuration structure `nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Refer to Tables 3-11 and 3-12 for a description of the `nidev` structure and device configuration flags, respectively.

Possible GPIB-Specific Errors

EARG - `bdh` is valid but does not refer to a device.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, DTempRd,
                    &nid, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;

rc = DosDevIOCtl   (NULL, &nid, DTempRd, CATEGORY, bdh);
```

DTempRdNew Device Level DTempRdNew

Purpose

Return current values of device configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, DTempRdNew,
                   (PVOID) &nid, (ULONG) ParmLength,
                   (PULONG) AddrParmLength,
                    NULL, (ULONG) DataLength,
                   (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL,(PVOID)&nid,DTempRdNew,
                    CATEGORY,(HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
OUT `nid`	New device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DTempRdNew` function returns the current values of various device configuration parameters that are set after the device is placed online. Once these parameters are read, they can be modified by using the `DTempWrtNew` function.

DTempRdNew Device Level DTempRdNew (Continued)

The device configuration structure `new_nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Refer to Tables 3-13 and 3-14 for a description of the `new_nidev` structure and device configuration flags.

Possible GPIB-Specific Errors

EARG - `bdh` is valid but does not refer to a device.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;

nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl   (bdh,CATEGORY, DTempRdNew,
                    &nid, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;

nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl   (NULL, &nid,DTempRdNew,CATEGORY,bdh);
```

DTempWrt Device Level DTempWrt

Purpose

Modify current configuration parameters of the device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,DTempWrt,
                    (PVOID) &nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DTempWrt, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `nid`	Device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DTempWrt` function dynamically modifies the current values of various device configuration parameters. Changes remain in effect until the device is placed offline.

DTempWrt — Device Level — DTempWrt (Continued)

The device configuration structure `nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Refer to Table 3-11 and 3-12 for a description of the `nidev` structure and device configuration parameters, respectively.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to a device or one of the parameters set in the `nibrd` structure is invalid.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, DTempRd,
                    &nid, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
if (rc != 0)  {
    nid.pad = 1 ;    /* change the primary address to one
    */
    nid.uflags &= ~UNAD; /* Do not unaddress the bus after
                            I/O transfers */

    rc = DosDevIOCtl     (bdh,CATEGORY, DTempWrt,
                          &nid, ParmLength, &ParmLength,
                          NULL, DataLength, &DataLength);
}
```

DTempWrt — Device Level — DTempWrt (Continued)

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct nidev nid;

rc = DosDevIOCtl   (NULL, &nid, DTempRd, CATEGORY, bdh);
if (rc != 0)  {
   nid.pad = 1 ;  /* change the primary address to one */
    nid.uflags &= ~UNAD; /* Do not unaddress the bus
                            after I/O transfers */

    rc = DosDevIOCtl  (NULL, &nid, CATEGORY,
                        DTempWrt, bdh);
}
```

DTempWrtNew Device Level DTempWrtNew

Purpose

Modify current values of device configuration parameters.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY,DTempWrtNew,
                    (PVOID) &nid, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &nid, DTempWrtNew,
                    CATEGORY,(HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `nid`	New device configuration structure
IN `ParmLength`	Size of `nid`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `DTempWrtNew` function dynamically modifies the current values of various device configuration parameters that are set after the device is placed online. Changes remain in effect until the device is placed offline.

DTempWrtNew Device Level DTempWrtNew (Continued)

The device configuration structure `new_nidev`, defined in the `nicode.h` include file, is used to read the device configuration parameters. Refer to Tables 3-13 and 3-14 for a description of the `new_nidev` structure and device configuration flags.

Possible GPIB-Specific Errors

EARG - Either `bdh` is valid but does not refer to a device or one of the parameters set in the `new_nidev` structure is invalid.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;
ULONG ParmLength = sizeof (nid);
ULONG DataLength = 0;

nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl (bdh,CATEGORY, DTempRdNew,
                  &nid, ParmLength, &ParmLength,
                  NULL, DataLength, &DataLength);
if (rc != 0)  {
    nid.Pad = 1 ;   /* change the primary address to one
    */
    nid.State &= ~UNAD; /* Do not unaddress the bus
                         after I/O transfers */

    rc = DosDevIOCtl (bdh,CATEGORY, DTempWrtNew,
                      &nid, ParmLength,&ParmLength,
                      NULL,DataLength,&DataLength);
}
```

DTempWrtNew Device Level DTempWrtNew (Continued)

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

struct new_nidev nid;

nid.Dev_Size = sizeof (nid);
rc = DosDevIOCtl   (NULL, &nid,DTempRdNew,CATEGORY,bdh);
if (rc != 0)  {
    nid.Pad = 1 ;/* change the primary address to one*/
    nid.State &= ~UNAD; /*Do not unaddress the bus
                          after I/O transfers */

    rc = DosDevIOCtl(NULL,&nid,CATEGORY,DTempWrtNew, bdh);
}
```

OFFLINE

Device Level
Board Level

OFFLINE

Purpose

Place the device or interface board offline.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh ,CATEGORY, OFFLINE,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, OFFLINE, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle
IN `ParmLengthSize`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The board or device is reset and its software configurable parameters are placed in their pre-configured state. The device or interface board is taken offline.

OFFLINE

Device Level
Board Level

OFFLINE
(Continued)

Possible GPIB-Specific Errors

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

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, OFFLINE,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, OFFLINE, CATEGORY, bdh);
```

ONLINE

Device Level
Board Level

ONLINE

Purpose

Place the device or interface board online.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, ONLINE,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, ONLINE, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The device or board is reset and all software-configurable parameters are placed in their pre-configured state. The device or interface board is left operational or online.

ONLINE

Device Level
Board Level

ONLINE
(Continued)

Possible GPIB-Specific Errors

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

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, ONLINE,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl  (NULL, NULL, ONLINE, CATEGORY,bdh);
```

PASSCONTROL Device Level PASSCONTROL

Purpose

Pass Control to another GPIB device with Controller capability.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, PASSCONTROL,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, PASSCONTROL, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`.
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

`PASSCONTROL` passes Controller-In-Charge status to the device indicated by `bdh`. The access board automatically unasserts the ATN line and goes to Controller Idle State (CIDS). This function assumes that the device has Controller capability.

PASSCONTROL Device Level PASSCONTROL (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to a device.
EBUS - There are no devices connected to the GPIB.
ECIC - The interface board is not CIC. See the *Device-Level Calls and Bus Management* section of the *NI-488.2M Software Reference Manual for OS/2*.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, PASSCONTROL,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, PASSCONTROL, CATEGORY,
                    bdh);
```

PPOLL

Device Level
Board Level

PPOLL

Purpose

Conduct a parallel poll.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, PPOLL,
                    (PVOID) &ppr, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &ppr, PPOLL,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle.
OUT `ppr`	Parallel poll response byte
IN `ParmLengthSize`	Size of `ppr`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

All the devices on the GPIB are polled in parallel by using the access board. The result of this poll is returned in `ppr`.

For more information on parallel polling, refer to the *GPIB Programming Techniques* section of the *NI-488.2M Software Reference Manual for OS/2*.

PPOLL

Device Level
Board Level

PPOLL
(Continued)

Possible GPIB-Specific Errors

EBUS - There are no devices connected to the GPIB.
ECIC - The interface board is not CIC. See the *Device-Level Calls and Bus Management* section of the *NI-488.2M Software Reference Manual for OS/2*.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT ppr;
ULONG ParmLength = sizeof(ppr);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, PPOLL,
                    &ppr, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT ppr;

rc = DosDevIOCtl   (NULL, &ppr, PPOLL, CATEGORY, bdh);
```

PPOLLCONF

Device Level
Board Level

PPOLLCONF

Purpose

Parallel poll configure.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, PPOLLCONF,
                    (PVOID) &ppc, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &ppc, PPOLLCONF,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle
IN `ppc`	Parallel poll enable/disable value
IN `ParmLengthSize`	Size of `ppc`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Size of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

PPOLLCONF

Device Level
Board Level

PPOLLCONF
(Continued)

Description

Device Level

If `bdh` is a device handle, `PPOLLCONF` enables or disables the device to respond 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).

PPOLLCONF

Device Level
Board Level

PPOLLCONF
(Continued)

Board Level

If `bdh` is a board handle, `PPOLLCONF` performs a local parallel poll configuration by using the parallel poll configuration value `ppc`. Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E).

For more information on parallel polling, refer to the *GPIB Programming Techniques* section of the *NI-488.2M Software Reference Manual for OS/2.*

Possible GPIB-Specific Errors

EARG - `ppc` does not contain a valid PPE or PPD message.
EBUS - Device level: There are no devices connected to the GPIB.
ECAP - Board level: The board is not configured to perform local parallel poll configuration.
ECIC - Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section of the *NI-488.2M Software Reference Manual for OS/2.*
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT ppc;
ULONG ParmLength = sizeof(ppc);
ULONG DataLength = 0;

ppc = 0x70;

rc = DosDevIOCtl   (bdh,CATEGORY, PPOLLCONF,
                    &ppc, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

PPOLLCONF

Device Level
Board Level

PPOLLCONF
(Continued)

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT ppc;

ppc = 0x70;
rc = DosDevIOCtl(NULL, &ppc, PPOLLCONF, CATEGORY, bdh);
```

PPOLLIST **Board Level** PPOLLIST

Purpose

Set or clear the `ist` bit for parallel polls.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, PPOLLIST,
                    (PVOID) &istval, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &istval, PPOLLIST,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `istval`	Designates whether `ist` should be set or cleared
IN `ParmLength`	Size of `istval`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The interface board `ist` (individual status) bit is set according to `istval`. If `istval` is zero, the `ist` bit is cleared; if `istval` is not zero, the `ist` bit is set.

For more information about parallel polling, refer to the *GPIB Programming Techniques* section of the *NI-488.2M Software Reference Manual for OS/2*.

PPOLLIST Board Level PPOLLIST (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT istval;
ULONG ParmLength = sizeof(istval);
ULONG DataLength = 0;

istval = 1;

rc = DosDevIOCtl   (bdh,CATEGORY, PPOLLIST,
                    &istval, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT istval;

istval = 1;
rc = DosDevIOCtl   (NULL, &istval, PPOLLIST, CATEGORY,
bdh);
```

PRESENT Board Level PRESENT

Purpose

Check for the presence of the board.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, PRESENT,
                    (PVOID) &bpresent, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &bpresent, PRESENT,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
OUT `bpresent`	Indicates if the board is present or not
IN `ParmLength`	Size of `bpresent`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `PRESENT` function reports whether a board is present in the system. If the board is present in the system, a 1 is returned in `bpresent`; if the board is not present in the system, a 0 is returned.

PRESENT Board Level PRESENT (Continued)

Possible GPIB-Specific Errors

EARG - bdh is valid but does not refer to an interface board.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT bpresent;
ULONG ParmLength = sizeof(bpresent);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, PRESENT,
                    &bpresent, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, &bpresent, PRESENT, CATEGORY,
                    bdh);
```

REQUEST Board Level REQUEST

Purpose

Request service.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, REQUEST,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, REQUEST, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

This function is used to request service from the Controller by asserting the GPIB Service Request (SRQ) line.

REQUEST **Board Level** REQUEST **(Continued)**

Possible GPIB-Specific Errors

EARG - bdh is a valid handle but does not refer to a board.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, REQUEST,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


rc = DosDevIOCtl   (NULL, NULL, REQUEST, CATEGORY, bdh);
```

SENDIFC **Board Level** SENDIFC

Purpose

Assert Interface Clear (IFC) for 100 µs.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, SENDIFC,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl  (NULL, NULL, SENDIFC, CATEGORY,
                     (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLength`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

This function asserts the GPIB Interface Clear (IFC) signal for at least 100 µs if the GPIB board is System Controller. This action initializes the GPIB and makes the interface board CIC and Active Controller with ATN asserted.

SENDIFC **Board Level** SENDIFC (Continued)

The IFC signal resets only the GPIB interface functions of bus devices and does not reset the internal device functions.

Consult your device documentation to determine how to reset your internal device functions.

Possible GPIB-Specific Errors

EARG - bdh is a valid handle but does not refer to a board.
ENEB - The interface board is either not installed or not configured properly.
ESAC - Board does not have System Controller capability.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh, CATEGORY, SENDIFC,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, SENDIFC, CATEGORY, bdh);
```

SETREMOTE **Board Level** SETREMOTE

Purpose

Set the Remote Enable line.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, SETREMOTE,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl  (NULL, NULL, SETREMOTE, CATEGORY,
                     (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `ParmLengthSize`	Size of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

`SETREMOTE` asserts the Remote Enable (REN) signal. REN is used by devices to select between local and remote modes of operation. A device does not actually enter remote mode until it receives its listen address.

SETREMOTE Board Level SETREMOTE (Continued)

Possible GPIB-Specific Errors

EARG - bdh is a valid descriptor but does not refer to a board.
ENEB - The interface board is either not installed or not configured properly.
ESAC - Board does not have System Controller capability.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh, CATEGORY, SETREMOTE,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, SETREMOTE, CATEGORY, bdh);
```

SPOLL **Device Level** SPOLL

Purpose

Conduct a serial poll.

Format

32-Bit

```
ULONG DosDevIOCtl   ((HFILE) bdh, CATEGORY, SPOLL,
                    (PVOID) &spr, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl  (&spr, NULL, SPOLL, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
OUT `spr`	Serial poll response byte
IN `ParmLength`	Sizeof `spr`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `SPOLL` function is used to serial poll the specified device; the serial poll response byte is stored in `spr`. If bit 6 (hex 40) of the response is set, the device is requesting service. When the automatic serial polling feature is enabled, the device may have been polled previously. In this case, `SPOLL` returns the previously acquired serial poll response.

For more information on serial polling, refer to Chapter 6, *GPIB Programming Techniques,* in the *NI-488.2M Software Reference Manual for OS/2*.

SPOLL — Device Level — SPOLL (Continued)

Possible GPIB-Specific Errors

EABO - The serial poll response could not be read within the timeout period.
EARG - `bdh` 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 the *NI-488.2M Software Reference Manual for OS/2*.
ESTB - Autopolling is enabled and the serial poll queue of the device has overflowed. Call `SPOLL` more often to keep the queue from overflowing.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT spr;
ULONG ParmLength = sizeof(spr);
ULONG DataLength = 0;


rc = DosDevIOCtl   (bdh, CATEGORY, SPOLL,
                    &spr, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"


rc = DosDevIOCtl   (NULL, &spr, SPOLL, CATEGORY, bdh);
```

SPOLLBYTE Board Level SPOLLBYTE

Purpose

Request service.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, SPOLLBYTE,
                    (PVOID) &spb, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl  (NULL, (PVOID) &spb, SPOLLBYTE,
                     CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board handle
IN `spb`	Serial poll byte value
IN `ParmLength`	Size of `spb`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

This function is used to request service from the Controller and is also used to give the Controller with an application-dependent status byte when the Controller serial polls the GPIB board. The value `spb` is the status byte that the GPIB board provides when serial polled by the Controller-In-Charge. If bit 6 (hex 40) is set in `spb`, the GPIB board requests service from the Controller by asserting the GPIB SRQ line.

SPOLLBYTE Board Level SPOLLBYTE (Continued)

Possible GPIB-Specific Errors

EARG - bdh is a valid descriptor but does not refer to a board.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT spb;
ULONG ParmLength = sizeof(spb);
ULONG DataLength = 0;

spb = 0x43;
rc = DosDevIOCtl   (bdh, CATEGORY, SPOLLBYTE,
                    &spb, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

spb = 0x43;

rc = DosDevIOCtl   (NULL, &spb, SPOLLBYTE, CATEGORY, bdh);
```

STATUS

Device Level
Board Level

STATUS

Purpose

Return current status of board or device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, STATUS,
                    (PVOID) &stat, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, (PVOID) &stat, STATUS,
                    CATEGORY, (HFILE) bdh)
```

Parameters

IN `bdh`	Board or device handle
OUT `stat`	Status information
IN `ParmLength`	Size of `stat`
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of Data Area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

The `STATUS` function returns the current board or device status information. Refer to Table 3-3 for a complete listing of the board-level and device-level status bits.

STATUS

Device Level
Board Level

STATUS
(Continued)

Possible GPIB-Specific Errors

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

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

USHORT stat;
ULONG ParmLength = sizeof(stat);
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh,CATEGORY, STATUS,
                    &stat, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, &stat, STATUS, CATEGORY, bdh);
```

TRIGGER Device Level TRIGGER

Purpose

Trigger selected device.

Format

32-Bit

```
ULONG DosDevIOCtl  ((HFILE) bdh, CATEGORY, TRIGGER,
                     NULL, (ULONG) ParmLength,
                    (PULONG) AddrParmLength,
                     NULL, (ULONG) DataLength,
                    (PULONG) AddrDataLength)
```

16-Bit

```
USHORT DosDevIOCtl (NULL, NULL, TRIGGER, CATEGORY,
                    (HFILE) bdh)
```

Parameters

IN `bdh`	Device handle
IN `ParmLength`	Length of parameter area; must be set to zero.
IN `AddrParmLength`	Address of `ParmLength`
IN `DataLength`	Length of data area; must be set to zero.
IN `AddrDataLength`	Address of `DataLength`
Function Return	Zero, if no error; GPIB-specific error value, otherwise.

Description

`TRIGGER` sends the Group Execute Trigger (GET) message to the device described by `bdh`.

TRIGGER Device Level TRIGGER (Continued)

Possible GPIB-Specific Errors

EARG - bdh is a valid handle 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 of the *NI-488.2M Software Reference Manual for OS/2*.
ENEB - The interface board is either not installed or not configured properly.

Examples

32-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

ULONG ParmLength = 0;
ULONG DataLength = 0;

rc = DosDevIOCtl   (bdh, CATEGORY, TRIGGER,
                    NULL, ParmLength, &ParmLength,
                    NULL, DataLength, &DataLength);
```

16-Bit

```
#define INCL_DOSDEVICES
#include <os2.h>
#include "nicode.h"

rc = DosDevIOCtl   (NULL, NULL, TRIGGER, CATEGORY, bdh);
```

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 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		3F	077	63	?	UNL

Message Definitions

DCL	Device Clear	MSA	My Secondary Address
GET	Group Execute Trigger	MTA	My Talk Address
GTL	Go To Local	PPC	Parallel Poll Configure
LLO	Local Lockout	PPD	Parallel Poll Disable
MLA	My Listen Address		

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
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE
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-488.2M Software Reference Manual for OS/2*.

If a function call returns an ENEB or EDVR error, all status word bits except the ERR bit are cleared. These error codes indicate that it is not possible to obtain the status of the GPIB board.

Each bit in `ibsta` can be set for device calls (dev), board calls (brd), or both (dev, brd).

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, `ibrd`, `ibwrt`, `ibcmd`, `Send`, `Receive`, 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 may 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. 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 some 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 whenever the GPIB board ceases to be the CIC, or the GPIB SRQ line is unasserted.

RQS (dev)

RQS appears in the status word only after a device-level call. RQS indicates that one or more automatic 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 as a result of a call to `ibwait`, or it occurs 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.

CMPL (dev, brd)

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

LOK (brd)

LOK indicates whether the 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 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 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 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 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 `ibgts` is 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 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 the Device Clear (DCL) command has been sent by another Controller, or whenever the GPIB board as a Listener detects 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, or 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.

EDVR (0)

EDVR is returned when the device or board name passed in an `ibfind` call is not configured in the software. In this case, the variable `ibcntl` contains the OS/2 error code 2, *Device not found* or error code 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 OS/2 error code 6, *Invalid handle*.

EDVR is also returned when the driver (`gpib.sys`) is not installed.

Solutions

1. Use only device or board names that are configured in the software utility `ibconf` as parameters in the `ibfind` function.

2. 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.

3. Make sure the NI-488.2M driver is installed.

4. Check the `config.sys` file in the root directory and make sure it contains the following line:

   `DEVICE=`*`dir`*`\gpib.sys`

   where *dir* is the directory that contains the file `gpib.sys`.

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

ECIC (1)

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

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

Solutions

1. Use `ibsic` or `SendIFC` to make the GPIB board become CIC on the GPIB.
2. Use `ibrsc 1` or run `ibconf` and make sure your GPIB board is configured as System Controller.
3. In multiple CIC situations, always be certain that the CIC bit appears in the status word `ibsta` before attempting these calls. If the CIC bit 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 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

1. Make sure that the GPIB address of your device matches the GPIB address of the device you want to write data to.

2. If you are using board-level functions, make sure that your device is properly addressed to listen before writing to it by using `ibcmd` or `sendsetup`.

3. Use the appropriate hex code in `ibcmd` to address your device.

4. Check your cable connections and make sure at least two-thirds of your devices are powered on.

5. Call `ibpad` (or `ibsad`, if necessary) to match the configured address to the device switch settings.

6. 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

1. Make sure that the GPIB board is addressed correctly before calling `ibrd` or `ibwrt`.

2 Avoid calling `ibgts` except immediately after an `ibcmd` call (`ibcmd` causes ATN to be asserted).

EARG (4)

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

- `ibtmo` called with a value not in the range 0 through 17.
- `ibeos` called with meaningless bits set in the high byte of the second parameter.
- `ibpad` or `ibsad` called with invalid addresses.
- `ibppc` called with invalid parallel poll configurations.
- A board-level function made with a valid device descriptor, or a device-level function made with a board descriptor.
- An NI-488.2 routine called with an invalid address.
- `PPollConfig` called with an invalid data line or sense bit.
- `Termination` parameter in `RcvRespMsg` and `Receive` is neither `STOPend` or an 8-bit EOS character.
- `eotmode` parameter in `Send`, `SendDataBytes`, and `SendList` is not `DABend`, `NULLend`, or `NLend`.
- In `Send`, `SendDataBytes`, or `SendList` routine, `eotmode` is `DABend` and `datacnt` is zero.

Solutions

1. Make sure that the parameters passed to the NI-488 function or NI-488.2 routine are valid.
2. 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` or by using `ibconf` to configure that capability into the software.

EABO (6)

EABO indicates that an I/O operation has been canceled—usually because of a timeout condition. Other causes are calling `ibstop` or receiving the Device Clear message from the CIC while performing an I/O operation.

Frequently, the I/O is not progressing (the Listener is not continuing to handshake or the Talker has stopped talking), or the byte count in the call which timed out was more than the other device was expecting.

Solutions

1. Use the correct byte count in input functions or have the Talker use the END message to signify the end of the transfer.

2. Lengthen the timeout period for the I/O operation using `ibtmo`.

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

ENEB (7)

ENEB occurs when a GPIB board does not exist at the I/O address specified in the configuration program. This situation 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 conflict in the system with the base I/O address.

Solutions

1. Make sure a GPIB board is in your computer that is configured both in hardware and software at a free base I/O address.

2. Make sure that the `Use This GPIB Interface` field in `ibconf` is set to `Yes`.

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 use only `ibstop`, `ibwait`, and `ibonl`. If any other call is attempted, EOIP is returned.

With the asynchronous I/O calls (`ibcmda`, `ibrda`, `ibwrta`) your application can perform additional non-GPIB operations while the I/O in progress. Once the asynchronous I/O has begun, further GPIB calls other than `ibstop`, `ibwait`, or `ibonl` are strictly limited. If a call interferes with the I/O operation in progress, it causes the driver to return 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.

Solutions

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

EBUS (14)

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

Solutions

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

ESTB (15)

ESTB occurs only during 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

1. Call `ibrsp` more frequently to empty the queue.

2. Disable autopolling with the `ibconfig` function or the `ibconf` utility.

ESRQ (16)

ESRQ occurs only during the `WaitSRQ` routine or `ibwait` function. ESRQ indicates that 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` and `FindRQS` routines. ETAB indicates that there was some problem with a table used by these functions.

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

Solutions

Increase the size of result arrays for `FindLstn` and `FindRQS`.

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: (800) 328-2203
(512) 794-5678

Branch Offices	**Phone Number**	**Fax Number**
Australia	(03) 879 9422	(03) 879 9179
Austria	(0662) 435986	(0662) 437010-19
Belgium	02/757.00.20	02/757.03.11
Denmark	45 76 26 00	45 76 71 11
Finland	(90) 527 2321	(90) 502 2930
France	(1) 48 14 24 00	(1) 48 14 24 14
Germany	089/741 31 30	089/714 60 35
Italy	02/48301892	02/48301915
Japan	(03) 3788-1921	(03) 3788-1923
Netherlands	03480-33466	03480-30673
Norway	32-848400	32-848600
Spain	(91) 640 0085	(91) 640 0533
Sweden	08-730 49 70	08-730 43 70
Switzerland	056/20 51 51	056/20 51 55
U.K.	0635 523545	0635 523154

Technical Support Form

Technical support is available at any time by fax. Include the information from the configuration form in the Getting Started Manual. Use 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 _____

Version _____

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-488.2M™ Function Reference Manual for OS/2**

Edition Date: **April 1994**

Part Number: **370951A-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, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(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 — A GPIB interface function that receives data or commands. Listeners use this 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

API — Application Program Interface

ASCII — American Standard Code for Information Interchange.

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

ATN (Attention) — A GPIB line that distinguishes between commands and data messages. When ATN is asserted, bytes on the GPIB DIO lines are commands.

Automatic Serial Polling (Autopolling)	A feature of the NI-488.2M software in which serial polls are executed automatically by the driver whenever a device asserts the GPIB SRQ line.

B

Board Function	A function that operates on or otherwise pertains to one of the GPIB interface boards in the computer.

C

CIC	See *Controller-In-Charge*.
`config.sys`	An OS/2 file that contains the names of the loadable device driver or driver programs that OS/2 loads when it is started up.
Controller-In-Charge (CIC)	The device that manages the GPIB by sending interface messages to other devices.

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 *IFC* and *SDC*.
Declaration File	A file containing definitions that must be placed at the beginning of an application program. `decl.h` is the declaration file for C programs using the NI-488 functions and NI-488.2 routines. See *Language Interface*.
Device Clear	See DCL.

Device Function	A function that operates on or otherwise pertains to a GPIB device rather than to the GPIB interface board in the computer. See *Board Function*.
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. Same as an OS/2-installed device driver.
DVM	digital voltmeter

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

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 perhaps a secondary address (MSA). The GPIB board has both a GPIB address and an I/O address.
GPIB Board	Refers to the National Instruments family of GPIB interface boards.
`gpib.sys`	The NI-488.2M driver file name.
Group Execute 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.
hex	hexadecimal

High-Level Function	A device 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. See *Low-Level Function*.
Hz	hertz

I

ibcnt	After each NI-488.2M I/O function, this global variable contains the actual number of bytes transmitted.
ibconf	The NI-488.2M driver configuration program.
iberr	A global variable that contains the specific error code associated with a function call that failed.
ibic	The Interface Bus Interactive Control program 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.
IFC or Interface Clear	A GPIB line used by the System Controller to initialize the bus. See *DCL* and *SDC*.
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.

L

LAD or Listen Address	See *MLA*.
Language Interface	Code that enables an application program that uses NI-488M functions or NI-488.2M routines to access the driver.
Listen Address	See *MLA*.
Listener	A GPIB device that receives data messages from a Talker.
LLO	Local Lockout is the GPIB command used to tell all devices that they may or should ignore remote (GPIB) data messages or local (front panel) controls, depending on whether the device is in local or remote program mode.
Low-Level Function	A rudimentary board or device function that performs a single operation. See *High-Level Function*.

M

Make File	Utility that compiles and links programs.
Memory-Resident	Resident in RAM.
MLA (My Listen Address)	A GPIB command used to address a device to be a Listener. There are 31 primary addresses.

MSA	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. There are 31 primary addresses.
Multitasking	The concurrent processing of more than one program or task. OS/2 provides a multitasking environment so that multiple applications can execute at the same time.

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*.

O

Open Device or Board	One that has been enabled or placed online by a system or language open function.

P

Parallel Poll	The process of polling all configured devices at once and reading a composite poll response. See *Serial Poll*.
Parallel Poll Configure	See *PPC*.
Parallel Poll Disable	See *PPD*.

Parallel Poll Enable	See *PPE*.
Parallel Poll Unconfigure	See *PPU*.
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
REN (Remote Enable)	A GPIB line controlled by the System Controller but used by the CIC to place devices in remote program mode.
RQS	Request Service

S

SDC	Selected Device Clear is the GPIB command used to reset internal or device functions of an addressed Listener. See *DCL* and *IFC*.
sec	second
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 data byte sent by a device when it is serially polled.
Status Word	See *ibsta*.
Synchronous	Refers to the relationship between the NI-488.2M driver functions and a process when executing driver functions is predictable; the process is blocked until the OS/2 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-488.2M 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

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W