NI-488.2™
Function Reference Manual for Macintosh

August 1995 Edition

Part Number 320898A-01

© Copyright 1995 National Instruments Corporation.
All Rights Reserved.

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 9 879 9422, Austria 0662 45 79 90 0, 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, Hong Kong 2645 3186, Italy 02 48301892,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 202 2544,
Netherlands 03480 33466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 20 51 51,
Taiwan 02 377 1200, U.K. 01635 523545

Limited Warranty

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

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

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

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

Copyright

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

Trademarks

NI-488® and NI-488.2™ are trademarks of National Instruments Corporation.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing 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

About This Manual xi
How to Use This Manual Set xi
Organization of This Manual xii
Conventions Used in This Manual xii
Related Documentation xiii
Customer Communication xiii

Chapter 1
NI-488 Functions 1-1
Function Names 1-1
Purpose 1-1
Format 1-1
Input and Output 1-1
Description 1-1
Possible Errors 1-2
List of NI-488 Functions 1-2
IBASK 1-5
IBBNA 1-13
IBCAC 1-15
IBCLR 1-17
IBCMD 1-19
IBCMDA 1-21
IBCONFIG 1-23
IBDEV 1-31
IBDMA 1-33
IBEOS 1-35
IBEOT 1-38
IBFIND 1-40
IBGTS 1-42
IBIST 1-44
IBLINES 1-46
IBLLO 1-48
IBLN 1-49
IBLOC 1-51
IBLOCK 1-53
IBONL 1-55
IBPAD 1-57
IBPCT 1-59
IBPPC 1-61
IBRD 1-63
IBRDA 1-65
IBRDF 1-68
IBRPP 1-70

IBRSC 1-72
IBRSP 1-74
IBRSV 1-76
IBSAD 1-78
IBSIC 1-80
IBSRE 1-82
IBSRQ 1-84
IBSTOP 1-85
IBTMO 1-86
IBTRG 1-88
IBUNLOCK 1-90
IBWAIT 1-92
IBWRT 1-94
IBWRTA 1-96
IBWRTF 1-99

Chapter 2
NI-488 Routines 2-1
Routine Names 2-1
Purpose 2-1
Format 2-1
Input and Output 2-1
Description 2-2
Possible Errors 2-2
List of NI-488.2 Routines 2-2
AllSpoll 2-4
DevClear 2-6
DevClearList 2-7
EnableLocal 2-9
EnableRemote 2-11
FindLstn 2-13
FindRQS 2-15
PassControl 2-17
PPoll 2-18
PPollConfig 2-20
PPollUnconfig 2-22
RcvRespMsg 2-24
ReadStatusByte 2-26
Receive 2-28
ReceiveSetup 2-30
ResetSys 2-32
Send 2-34
SendCmds 2-36
SendDataBytes 2-38
SendIFC 2-40
SendList 2-41
SendLLO 2-43

SendSetup 2-45
SetRWLS 2-47
TestSRQ 2-49
TestSys 2-50
Trigger 2-52
TriggerList 2-53
WaitSRQ 2-55

Chapter 3
Device Manager Functions and Routines 3-1
Function and Routine Names 3-1
Purpose 3-1
Control Number 3-1
Parameter Block Fields 3-1
Description 3-1
Examples 3-2
List of NI-488 Device Manager Functions 3-2
IBASK 3-4
IBBNA 3-6
IBCAC 3-7
IBCLR 3-9
IBCMD 3-10
IBCMDA 3-14
IBCONFIG 3-16
IBDEV 3-17
IBDMA 3-19
IBEOS 3-20
IBEOT 3-24
IBFIND 3-26
IBGTS 3-28
IBIST 3-30
IBLINES 3-32
IBLLO 3-34
IBLN 3-35
IBLOC 3-37
IBLOCK 3-39
IBONL 3-41
IBPAD 3-44
IBPCT 3-46
IBPPC 3-47
IBRD 3-50
IBRDA 3-53
IBRPP 3-57
IBRSC 3-59
IBRSP 3-60
IBRSV 3-62
IBSAD 3-64

IBSIC ... 3-66
IBSRE ... 3-67
IBSTOP ... 3-69
IBTMO ... 3-71
IBTRG ... 3-75
IBUNLOCK ... 3-76
IBWAIT ... 3-78
IBWRT ... 3-82
IBWRTA ... 3-85
NI-488 Programming Examples for the Device Manager ... 3-88
Example Program—High-Level Device Manager Calls ... 3-88
Example Program—Low-Level Device Manager Calls ... 3-90
Example Program—Accessing the GPIB Driver from THINK Pascal ... 3-93
List of NI-488.2 Device Manager Routines ... 3-99
NI-488.2 Routine Descriptions ... 3-100
AllSpoll ... 3-101
DevClear ... 3-102
DevClearList ... 3-103
EnableLocal ... 3-104
EnableRemote ... 3-105
FindLstn ... 3-106
FindRQS ... 3-108
PassControl ... 3-109
PPoll ... 3-110
PPollConfig ... 3-111
PPollUnconfig ... 3-112
RcvRespMsg ... 3-113
ReadStatusByte ... 3-114
Receive ... 3-115
ReceiveSetup ... 3-116
ResetSys ... 3-117
Send ... 3-118
SendCmds ... 3-119
SendDataBytes ... 3-120
SendIFC ... 3-121
SendList ... 3-122
SendLLO ... 3-123
SendSetup ... 3-124
SendRWLS ... 3-125
TestSRQ ... 3-126
TestSys ... 3-127
Trigger ... 3-128
TriggerList ... 3-129
WaitSRQ ... 3-130
Device Manager GPIB Programming Example ... 3-131
Device Manager Example Program—NI-488.2 Routines ... 3-131

Appendix A
Multiline Interface Messages A-1

Appendix B
Status Word Conditions B-1

Appendix C
Error Codes and Solutions C-1

Appendix D
Customer Communication D-1

Glossary G-1

Index I-1

Tables

Table 1-1. List of NI-488 Device-Level Functions 1-2
Table 1-2. List of NI-488 Board-Level Functions 1-3
Table 1-3. ibask Board Configuration Parameter Options 1-7
Table 1-4. ibask Device Configuration Parameter Options 1-11
Table 1-5. ibconfig Board Configuration Parameter Options 1-25
Table 1-6. ibconfig Device Configuration Parameter Options 1-29
Table 1-7. EOS Configurations 1-36
Table 1-8. Timeout Code Values 1-87
Table 1-9. Wait Mask Layout 1-93

Table 2-1. List of NI-488.2 Routines 2-2

Table 3-1. NI-488 Device Manager Control Calls 3-2
Table 3-2. Data Transfer Termination Method 3-20
Table 3-3. Parallel Poll Commands 3-58
Table 3-4. Timeout Code Values 3-72
Table 3-5. Wait Mask Layout 3-79
Table 3-6. Device Manager NI-488.2 Routines 3-99

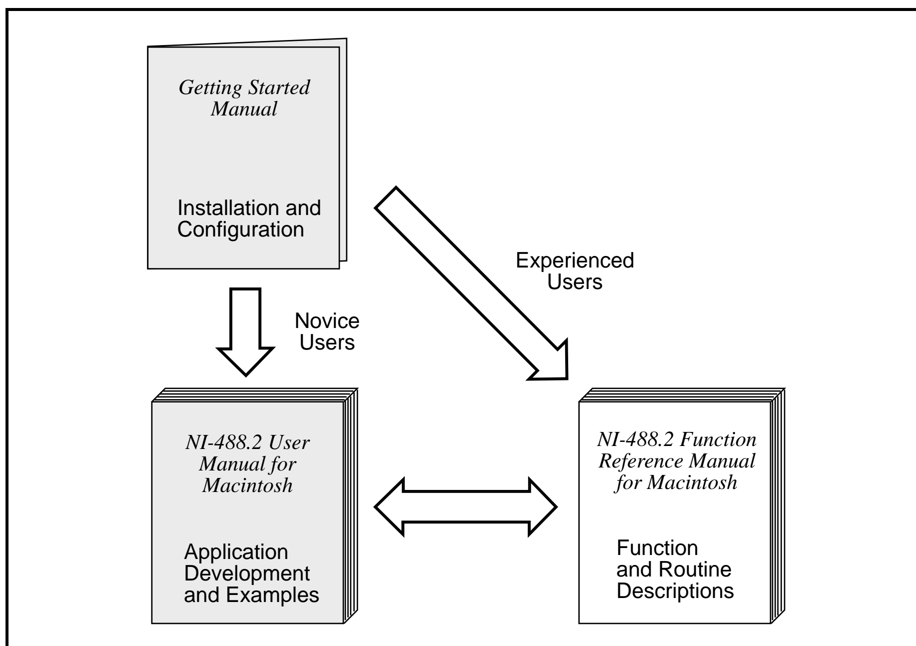
Table B-1. Status Word Bits B-1

Table C-1. GPIB Error Codes C-1

About This Manual

This manual describes the features and functions of the NI-488.2 software for Macintosh. This manual assumes that you are already familiar with the Macintosh operating system.

How to Use This Manual Set



Use the getting started manual that came with your kit to install and configure your GPIB hardware and NI-488.2 software.

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

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

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *NI-488 Functions,* lists the available NI-488 functions and describes the purpose, format, input and output parameters, and possible errors for each function.
- Chapter 2, *NI-488.2 Routines,* lists the available NI-488.2 routines and describes the purpose, format, input and output parameters, and possible errors for each routine.
- Chapter 3, *Device Manager Functions and Routines,* describes the purpose, format, input and output parameters, and possible errors for each Device Manager function and routine.
- Appendix A, *Multiline Interface Messages*, contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions. These multiline interface messages are sent and received with ATN TRUE.
- Appendix B, *Status Word Conditions*, gives a detailed description of the conditions reported in the status word, `ibsta`.
- Appendix C, *Error Codes and Solutions*, lists a description of each error, some conditions under which it might occur, and possible solutions.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual.

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
bold italic	Bold italic text denotes a note, caution, or warning.
`monospace`	Text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is

	also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, constants, variables, filenames, and extensions, and for statements and comments taken from program code.
< >	Angle brackets enclose the name of a key on the keyboard—for example, <Shift>.
IEEE 488 and IEEE 488.2	*IEEE 488* and *IEEE 488.2* refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.

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

Related Documentation

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

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*
- *Inside Macintosh*, Addison-Wesley Publishing Company, Reading, MA, 1994
- *Macintosh Programmer's Workshop, Version 3.3*, Apple Computer, Inc., Cupertino, CA, 1993
- *Metrowerks CodeWarrior User’s Guide*, Metrowerks, Inc., Mooers, NY
- *Microsoft QuickBASIC*, Microsoft Corp., Redmond, WA, 1988
- *THINK C User's Manual*, Symantec Corp., Bedford, MA

Customer Communication

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

Chapter 1
NI-488 Functions

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

While using the functions, you might find it helpful to refer to Chapter 2, *Developing Your Application*, and Chapter 5, *GPIB Programming Techniques,* in the *NI-488.2 User Manual for Macintosh.*

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

- MPW C version 3.0 or higher , THINK C version 4.0 or higher, and Metrowerks CodeWarrior 1.1 or higher
- Microsoft QuickBASIC version 1.0 or higher

Input and Output

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

Description

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

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
`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
`iblock`	Lock access to a GPIB-ENET board or device
`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

(continues)

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

Function	Purpose
`ibrsp`	Conduct a serial poll
`ibsad`	Change or disable the secondary address
`ibstop`	Abort asynchronous I/O operation
`ibtmo`	Change or disable the I/O timeout period
`ibtrg`	Trigger selected device
`ibunlock`	Unlock access to a GPIB-ENET board or 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
`iblock`	Lock access to a GPIB-ENET board or device

(continues)

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

Function	Purpose
`ibonl`	Place the interface board online or offline
`ibpad`	Change the primary address
`ibppc`	Parallel poll configure
`ibrd`	Read data from a device into a user buffer
`ibrda`	Read data asynchronously from a device into a user buffer
`ibrdf`	Read data from a device into a file
`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
`ibunlock`	Unlock access to a GPIB-ENET board or 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

IBASK

Board Level
Device Level

Purpose

Return information about software configuration parameters.

Format

QuickBASIC

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

C

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

Input

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

Output

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

Description

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

IBASK

Board Level
Device Level

IBASK (Continued)

Possible Errors

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

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

Constants	Values	Constants	Values
• IbaAUTOPOLL	0x0007	• IbaPAD	0x0001
• IbaBaud	0x0204	• IbaParity	0x0205
• IbaCICPROT	0x0008	• IbaPP2	0x0010
• IbaComPort	0x0208	• IbaPPC	0x0005
• IbaDataBits	0x0207	• IbaReadAdjust	0x0013
• IbaDMA	0x0012	• IbaRsv	0x0021
• IbaEndBitIsNormal	0x001A	• IbaSAD	0x0002
• IbaEOSchar	0x000F	• IbaSC	0x000A
• IbaEOScmp	0x000E	• IbaSRE	0x000B
• IbaEOSrd	0x000C	• IbaStopBits	0x0206
• IbaEOSwrt	0x000D	• IbaTIMING	0x0011
• IbaEOT	0x0004	• IbaTMO	0x0003
• IbaHSCableLength	0x001F	• IbaWriteAdjust	0x0014
• IbaIst	0x0020		

IBASK

Board Level
Device Level

IBASK (Continued)

Table 1-3. ibask Board Configuration Parameter Options

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

(continues)

IBASK

Board Level
Device Level

IBASK (Continued)

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

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

(continues)

IBASK

Board Level
Device Level

IBASK (Continued)

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

Options (Constants)	Options (Values)	Returned Information
IbaTIMING	0x0011	The current bus timing of the board. 1 = Normal timing (T1 delay of 2 µs.) 2 = High speed timing (T1 delay of 500 ns.) 3 = Very high speed timing (T1 delay of 350 ns.)
IbaDMA	0x0012	zero = The board will not use DMA for GPIB transfers. non-zero = The board will use DMA for GPIB transfers. See `ibdma`.
IbaReadAdjust	0x0013	0 = Read operations do not have pairs of bytes swapped. 1 = Read operations have each pair of bytes swapped.
IbaWriteAdjust	0x0014	0 = Write operations do not have pairs of bytes swapped. 1 = Write operations have each pair of bytes swapped.
IbaEndBitIsNormal	0x001A	zero = The END bit of `ibsta` is set only when EOI or EOI plus the EOS character is received. If the EOS character is received without EOI, the END bit is not set. non-zero = The END bit is set whenever EOI, EOS, or EOI plus EOS is received.
IbaHSCableLength	0x001F	0 = High-speed data transfer (HS488) is disabled. 1 to 15 = High-speed data transfer (HS488) is enabled. The number returned represents the number of meters of GPIB cable in your system. See the NI-488.2 user manual for information about high-speed data transfers (HS488).
IbaIst	0x0020	The individual status (`ist`) bit of the board.

(continues)

IBASK

Board Level
Device Level

IBASK (Continued)

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

Options (Constants)	Options (Values)	Returned Information
IbaRsv	0x0021	The current serial poll status byte of the board.
IbaBaud	0x0204	If your GPIB interface is a GPIB-232CT-A, then this option returns the baud rate that the NI-488.2 software is configured to use when communicating with the interface.
IbaParity	0x0205	If your GPIB interface is a GPIB-232CT-A, then this option returns the parity that the NI-488.2 software is configured to use when communicating with the interface.
IbaStopBits	0x0206	If your GPIB interface is a GPIB-232CT-A, then this option returns the number of stop bits that the NI-488.2 software is configured to use when communicating with the interface.
IbaDataBits	0x0207	If your GPIB interface is a GPIB-232CT-A, then this option returns the number of data bits that the NI-488.2 software is configured to use when communicating with the interface.
IbaComPort	0x0208	If your GPIB interface is a GPIB-232CT-A, then this option returns the serial port number that the NI-488.2 software is configured to use when communicating with the interface.

IBASK

Board Level
Device Level

IBASK (Continued)

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

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

Table 1-4. ibask Device Configuration Parameter Options

Options (Constants)	Options (Values)	Returned Information
IbaPAD	0x0001	The current primary address of the device. See `ibpad.`
IbaSAD	0x0002	The current secondary address of the device. See `ibsad.`
IbaTMO	0x0003	The current I/O timeout of the device. See `ibtmo.`
IbaEOT	0x0004	zero = The GPIB EOI line is not asserted at the end of a write operation. non-zero = EOI is asserted at the end of a write operation. See `ibeot.`
IbaEOSrd	0x000C	zero = The EOS character is ignored during read operations. non-zero = Read operation is terminated by the EOS character. See `ibeos.`

(continues)

IBASK

Board Level
Device Level

IBASK (Continued)

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

Options (Constants)	Options (Values)	Returned Information
IbaEOSwrt	0x000D	zero = The EOI line is not asserted when the EOS character is sent during a write operation. non-zero = The EOI line is asserted when the EOS character is sent during a write operation. See `ibeos`.
IbaEOScmp	0x000E	zero = A 7-bit compare is used for all EOS comparisons. non-zero = An 8-bit compare is used for all EOS comparisons. See `ibeos`.
IbaEOSchar	0x000F	The current EOS character of the device. See `ibeos`.
IbaReadAdjust	0x0013	0 = Read operations do not have pairs of bytes swapped. 1 = Read operations have each pair of bytes swapped.
IbaWriteAdjust	0x0014	0 = Write operations do not have pairs of bytes swapped. 1 = Write operations have each pair of bytes swapped.
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.
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

QuickBASIC

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

C

```
short ibbna (short ud,char bname [])
```

Input

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

Output

Function Return	The value of `ibsta`

Description

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

IBBNA **Device Level** IBBNA (Continued)

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 Chapter 5 of the *NI-488.2 User Manual for Macintosh*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The access board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBCAC **Board Level** IBCAC

Purpose

Become Active Controller.

Format

QuickBASIC

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

C

short `ibcac` (short `ud`, short `v`)

Input

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

Output

Function Return	The value of `ibsta`

Description

Using `ibcac`, the designated GPIB 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-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBCLR — Device Level — IBCLR

Purpose

Clear a specific device.

Format

QuickBASIC

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

C

```
short ibclr (short ud)
```

Input

`ud`	A device unit descriptor

Output

Function Return	The value of `ibsta`

Description

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

IBCLR **Device Level** IBCLR (Continued)

Possible Errors

EARG	`ud` is a valid descriptor but does not refer to a device.
EBUS	There are no devices connected to the GPIB.
ECIC	The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBCMD **Board Level** IBCMD

Purpose

Send GPIB commands.

Format

QuickBASIC

`CALL ibcmd (ud%,cmd$)`

or

`CALL ibcmd (ud%,cmd%(0),{cnt%|cnt&})`

C

short `ibcmd (`short `ud,`char `*cmd,`long `cnt)`

Input

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

Output

Function Return	The value of `ibsta`

Description

`ibcmd` sends `cnt` bytes from `cmd` over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable `ibcnt`. 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.

IBCMD

Board Level

IBCMD (Continued)

Possible Errors

EABO	The timeout period expired before all of the command bytes were sent.
EARG	`ud` is valid but does not refer to an interface board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners are on the GPIB.
EOIP	Asynchronous I/O is in progress.

IBCMDA Board Level IBCMDA

Purpose

Send GPIB commands asynchronously.

Format

QuickBASIC

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

or

```
CALL ibcmda (ud%,cmd%(0),{cnt%|cnt&})
```

C

```
short ibcmda (short ud,char *cmd,long cnt)
```

Input

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

Output

Function Return	The value of `ibsta`

Description

`ibcmda` sends `cnt` bytes from `cmd` over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable `ibcnt`. 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.

IBCMDA **Board Level** IBCMDA

(Continued)

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

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

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

Possible Errors

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

IBCONFIG

Board Level
Device Level

Purpose

Change the software configuration parameters.

Format

QuickBASIC

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

C

short `ibconfig` (short `ud`, unsigned short `option`, unsigned short `value`)

Input

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

Output

Function Return	The value of `ibsta`

Description

`ibconfig` changes the configuration item to the specified value for the selected board or device. `option` may be any of the defined constants in Table 1-5 and `value` must be valid for the parameter that you are configuring. The previous setting of the configured item is return in `iberr`.

IBCONFIG

Board Level
Device Level

IBCONFIG (Continued)

Possible Errors

EARG	Either `option` or `value` is not valid. See Table 1-5.
ECAP	The driver is not able to make the requested change.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
EOIP	Asynchronous I/O is in progress.

Table 1-5 lists the options you can use with `ibconfig` when `ud` is a board descriptor or a board index. The following is an alphabetical list of the `option` constants included in Table 1-5.

Constants	Values	Constants	Values
• IbcAUTOPOLL	0x0007	• IbcPPC	0x0005
• IbcCICPROT	0x0008	• IbcPPollTime	0x0019
• IbcDMA	0x0012	• IbcREADDR	0x0006
• IbcEndBitIsNormal	0x001A	• IbcReadAdjust	0x0013
• IbcEOSchar	0x000F	• IbcSAD	0x0002
• IbcEOScmp	0x000E	• IbcSC	0x000A
• IbcEOSrd	0x000C	• IbcSendLLO	0x0017
• IbcEOSwrt	0x000D	• IbcSRE	0x000B
• IbcEOT	0x0004	• IbcTIMING	0x0011
• IbcHSCableLength	0x001F	• IbcTMO	0x0003
• IbcPAD	0x0001	• IbcUnAddr	0x001B
• IbcPP2	0x0010	• 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 `NI-488 Config`.
IbcSAD	0x0002	Changes the secondary address of the board. Identical to `ibsad`. Default determined by `NI-488 Config`.
IbcTMO	0x0003	Changes the I/O timeout limit of the board. Identical to `ibtmo`. Default determined by `NI-488 Config`.
IbcEOT	0x0004	Changes the data termination mode for write operations. Identical to `ibeot`. Default determined by `NI-488 Config`.
IbcPPC	0x0005	Configures the board for parallel polls. Identical to board-level `ibppc`. Default: zero.
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 `NI-488 Config`.
IbcAUTOPOLL	0x0007	zero = Disable automatic serial polling. non-zero = Enable automatic serial polling. Default determined by `NI-488 Config`. Refer to the *NI-488.2 User Manual for Macintosh* for more information about automatic serial polling.

(continues)

IBCONFIG

Board Level
Device Level

IBCONFIG (Continued)

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

Options (Constants)	Options (Values)	Legal Values
IbcCICPROT	0x0008	zero = Disable the CIC protocol. non-zero = Enable the CIC protocol. Default determined by `NI-488 Config`. Refer to the *NI-488.2 User Manual for Macintosh* for more information about the CIC protocol.
IbcSC	0x000A	Request or release system control. Identical to `ibrsc`. Default determined by `NI-488 Config`.
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 `NI-488 Config`.
IbcEOSwrt	0x000D	zero = Do not assert EOI with the EOS character during write operations. non-zero = Assert EOI with the EOS character during writes operations. Default determined by `NI-488 Config`.
IbcEOScmp	0x000E	zero = Use 7 bits for the EOS character comparison. non-zero = Use 8 bits for the EOS character comparison. Default determined by `NI-488 Config`.
IbcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character. Default determined by `NI-488 Config`.

(continues)

IBCONFIG

Board Level
Device Level

IBCONFIG (Continued)

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. Refer to the *NI-488.2 User Manual for Macintosh* for more information about parallel polling.
IbcTIMING	0x0011	1 = Normal timing (T1 delay of 2 µs). 2 = High-speed timing (T1 delay of 500 ns). 3 = Very high-speed timing (T1 delay of 350 ns). Default determined by `NI-488 Config`. The T1 delay is the GPIB source handshake timing.
IbcDMA	0x0012	Identical to `ibdma`. Default determined by `NI-488 Config`.
IbcReadAdjust	0x0013	0 = No byte swapping. 1 = Swap pairs of bytes during a read. Default: zero.
IbcWriteAdjust	0x0014	0 = No byte swapping. 1 = Swap pairs of bytes during a write. Default: zero.
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.

(continues)

IBCONFIG

Board Level
Device Level

IBCONFIG (Continued)

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

Options (Constants)	Options (Values)	Legal Values
IbcPPollTime	0x0019	0 = Use the standard duration (2 µs) when conducting a parallel poll. 1 to 17 = Use a variable length duration when conducting a parallel poll. The duration represented by 1 to 17 corresponds to the `ibtmo` values. Default: zero.
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.
IbcHSCableLength	0x001F	0 = High-speed data transfer (HS488) is disabled. 1 to 15 = The number of meters of GPIB cable in your system. The NI-488.2 software uses this information to select the appropriate high-speed data transfer (HS488) mode. Default determined by `NI-488 Config`. See the *NI-488.2 User Manual for Macintosh* for information about high-speed data transfers (HS488).

IBCONFIG

Board Level
Device Level

IBCONFIG (Continued)

Table 1-6 lists the options you can use with `ibconfig` when `ud` is a device descriptor or a device index. The following is an alphabetical list of the option constants included in Table 1-6.

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

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 `NI-488 Config`.
IbcSAD	0x0002	Changes the secondary address of the device. Identical to `ibsad`. Default determined by `NI-488 Config`.
IbcTMO	0x0003	Changes the device I/O timeout limit. Identical to `ibtmo`. Default determined by `NI-488 Config`.
IbcEOT	0x0004	Changes the data termination method for writes. Identical to `ibeot`. Default determined by `NI-488 Config`.
IbcEOSrd	0x000C	non-zero = Terminate reads when the EOS character is read. Default determined by `NI-488 Config`.

(continues)

IBCONFIG

Board Level
Device Level

IBCONFIG (Continued)

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 `NI-488 Config.`
IbcEOScmp	0x000E	zero = Use seven bits for the EOS character comparison. non-zero = Use 8 bits for the EOS character comparison. Default determined by `NI-488 Config.`
IbcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character. Default determined by `NI-488 Config.`
IbcReadAdjust	0x0013	0 = No byte swapping. 1 = Swap pairs of bytes during a read. Default: zero.
IbcWriteAdjust	0x0014	0 = No byte swapping. 1 = Swap pairs of bytes during a write. Default: zero.
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.

IBDEV **Device Level** IBDEV

Purpose

Open and initialize a device descriptor.

Format

QuickBASIC

```
CALL ibdev (board.index%,pad%,sad%,tmo%,eot%,eos%,ud%)
```

C

`ud =` short `ibdev (`short `boardindex,`short `pad,` short `sad,` short `tmo,` short `eot,`
short `eos)`

Input

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

Output

`ud`	Returned device descriptor

Description

`ibdev` acquires a device descriptor to use in subsequent device-level NI-488 functions. It opens and initializes a device descriptor and configures it according to the input parameters.

For more details on the meaning and effect of each input parameter, see the corresponding NI-488 functions for `ibbna`, `ibpad`, `ibsad`, `ibtmo`, `ibeot`, and `ibeos`.

IBDEV Device Level IBDEV (Continued)

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

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

Possible Errors

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

IBDMA **Board Level** IBDMA

Purpose

Enable or disable DMA.

Format

QuickBASIC

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

C

short `ibdma (`short `ud,`short `v)`

Input

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

Output

Function Return	The value of `ibsta`

Description

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

IBDMA **Board Level** IBDMA (Continued)

Possible Errors

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

IBEOS

Board Level
Device Level

Purpose

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

Format

QuickBASIC

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

C

short `ibeos` (short `ud`, short `v`)

Input

`ud`	A board or device descriptor
`v`	EOS mode and character information

Output

Function Return	The value of `ibsta`

Description

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

IBEOS

Board Level
Device Level

IBEOS
(Continued)

Table 1-7. EOS Configurations

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

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

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

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

For more information on the termination of I/O operations refer to the *NI-488.2 User Manual for Macintosh.*

Possible Errors

EARG The high byte of `v` contains invalid bits.

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

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

EOIP Asynchronous I/O is in progress.

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 */
```

IBEOT

Board Level
Device Level

Purpose

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

Format

QuickBASIC

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

C

```
short ibeot ( short ud,short v)
```

Input

ud	A board or device descriptor
v	Enables or disables the end of transmission assertion of EOI

Output

Function Return	The value of `ibsta`

Description

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

For more information on the termination of I/O operations refer to the *NI-488.2 User Manual for Macintosh.*

IBEOT

Board Level
Device Level

IBEOT (Continued)

Possible Errors

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

IBFIND

Board Level
Device Level

Purpose

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

Format

QuickBASIC

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

C

```
ud = short ibfind (char udname [])
```

Input

udname	A user-configured device or board name

Output

ud	Returned device descriptor

Description

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

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

If `ibfind` is unable to get a valid descriptor, a -1 is returned; the ERR bit is set in `ibsta` and `iberr` contains EDVR.

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

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 Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

EDVR Either `udname` is not recognized as a board or device name or the NI-488.2 driver is not installed.

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

IBGTS **Board Level** IBGTS

Purpose

Go from Active Controller to Standby.

Format

QuickBASIC

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

C

short `ibgts` (short `ud`, short `v`)

Input

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

Output

Function Return	The value of `ibsta`

Description

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

Before performing an `ibgts` with shadow handshake, call the `ibeos` function to establish proper EOS modes.

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

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-488.2 driver is not installed.

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

EOIP Asynchronous I/O is in progress.

IBIST **Board Level** IBIST

Purpose

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

Format

QuickBASIC

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

C

short `ibist (` short `ud,` short `v)`

Input

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

Output

Function Return	The value of `ibsta`

Description

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

For more information on parallel polling, refer to the *NI-488.2 User Manual for Macintosh*.

IBIST **Board Level** IBIST (Continued)

Possible Errors

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

IBLINES

Board Level
Device Level

Purpose

Return the status of the eight GPIB control lines.

Format

QuickBASIC

```
CALL iblines (ud%,gpib.lines%)
```

C

short `iblines (` short `ud,`unsigned short `*gpib_lines);`

Input

`ud`	Board or device descriptor

Output

`gpiblines` or `linesbufName`	Returns GPIB control line state information
Function Return	The value of `ibsta`

Description

`iblines` returns the state of the GPIB control lines in `gpiblines` or `linesbufName`. 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

 © National Instruments Corp.

IBLINES

Board Level
Device Level

IBLINES (Continued)

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

Possible Errors

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

Example

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

IBLLO

Board Level
Device Level

Purpose

Place devices in Local Lockout.

Format

QuickBASIC

```
CALL ibllo (ud%)
```

C

```
short ibllo (short ud)
```

Input

`ud`	Board or device descriptor

Output

Function Return	The value of `ibsta`

Description

The `ibllo` function asserts REN and sends the message LLO, which places devices in the Local Lockout state when they are addressed to listen. This usually inhibits recognition of front panel input.

`ibllo` sends the following commands.

- REN asserted
- Local Lockout (LLO)

Possible Errors

ECIC	The interface board is not Controller-In-Charge.
ESAC	The interface board is not System Controller.

IBLN

Board Level
Device Level

Purpose

Check for the presence of a device on the bus.

Format

QuickBASIC

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

C

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

Input

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

Output

`listen` or `listenFlagName`	Indicates whether or not a device is present
Function Return	The value of `ibsta`

Description

`ibln` determines whether there is a listening device at the GPIB address designated by the `pad` and `sad` parameters. If `ud` is a board descriptor, then the bus associated with that board is tested for Listeners. If `ud` is a device descriptor, then `ibln` uses the access board associated with that device to test for Listeners. If a Listener is detected, a non-zero value is returned in `listen` or `listenFlagName`. If no Listener is found, zero is returned.

IBLN

Board Level
Device Level

IBLN (Continued)

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

Possible Errors

ECIC — Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

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

EOIP — Asynchronous I/O is in progress.

IBLOC

Board Level
Device Level

Purpose

Go to Local.

Format

QuickBASIC

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

C

```
short ibloc (short ud)
```

Input

`ud`	Board or device descriptor

Output

Function Return	The value of `ibsta`

Description

Board Level

If the board is not in a lockout state (LOK does not appear in the status word, `ibsta`), `ibloc` places the board in local mode. Otherwise, the call has no effect.

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

Device Level

Unless the REN (Remote Enable) line has been unasserted with the `ibsre` function, all device-level functions automatically place the specified device in remote program mode. `ibloc` is used to move devices temporarily from a remote program mode to a local mode until the next device function is executed on that device.

IBLOC

Board Level
Device Level

IBLOC (Continued)

Possible Errors

EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBLOCK

Board Level
Device Level

IBLOCK

Purpose

Lock access to a GPIB-ENET board or device.

Format

QuickBASIC

```
CALL iblock (ud%)
```

C

```
short iblock (short ud)
```

Input

ud	A board or device descriptor

Output

Function Return	The value of ibsta

Description

`iblock` is used to obtain exclusive access to a GPIB-ENET interface.

Board Level

The `iblock` function blocks other processes from accessing the interface designated by `id` while the lock is in effect. The interface is released via an `ibunlock` function call made with the same board descriptor.

Device Level

The `iblock` function blocks other processes from accessing the device designated by `id` while the lock is in effect. The device lock is released via an `ibunlock` function call made with the same device descriptor.

IBLOCK

Board Level
Device Level

IBLOCK
(Continued)

Recommended Usage

In general, the `iblock` function should be used to gain critical access to a GPIB-ENET board or device when multiple processes might be accessing the same board or device. While locked, the software guarantees that subsequent calls made from the privileged board or device are completed without interruption.

Refer also to *IBUNLOCK.*

Possible Errors

EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ELCK	Occurs if the GPIB-ENET board or device being locked is already locked by another process.

IBONL

Board Level
Device Level

Purpose

Place the device or interface board online or offline.

Format

QuickBASIC

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

C

short `ibonl (`short `ud,`short `v)`

Input

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

Output

Function Return	The value of `ibsta`

Description

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

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

IBONL

Board Level
Device Level

IBONL (Continued)

Possible Errors

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

IBPAD

Board Level
Device Level

Purpose

Change the primary address.

Format

QuickBASIC

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

C

short `ibpad (` short `ud,` short `v)`

Input

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

Output

Function Return	The value of `ibsta`

Description

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

IBPAD

Board Level
Device Level

IBPAD (Continued)

Possible Errors

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

IBPCT **Device Level** IBPCT

Purpose

Pass control to another GPIB device with Controller capability.

Format

QuickBASIC

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

C

short `ibpct (` short `ud)`

Input

`ud`	Device descriptor

Output

Function Return	The value of `ibsta`

Description

`ibpct` passes Controller-in-Charge status to the device indicated by `ud.` The access board automatically unasserts the ATN line and goes to Controller Idle State. This function assumes that the device has Controller capability.

IBPCT **Device Level** IBPCT (Continued)

Possible Errors

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

EBUS No devices are connected to the GPIB.

ECIC The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

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

EOIP Asynchronous I/O is in progress.

IBPPC

Board Level
Device Level

Purpose

Parallel poll configure.

Format

QuickBASIC

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

C

short `ibppc` (short `ud`, short `v`)

Input

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

Output

Function Return	The value of `ibsta`

Description

Board Level

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

IBPPC

Board Level
Device Level

IBPPC
(Continued)

Device Level

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

For more information on parallel polling, refer to the *NI-488.2 User Manual for Macintosh*.

Possible Errors

EARG	`v` does not contain a valid PPE or PPD message.
EBUS	Device level: No devices are connected to the GPIB.
ECAP	Board level: The board is not configured to perform local parallel poll configuration (see `ibconfig`, option `IbcPP2`).
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRD

Board Level
Device Level

Purpose

Read data from a device into a user buffer.

Format

QuickBASIC

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

or

```
CALL ibrd (ud%,{a%(0)|a!(0)|a#(0)},{cnt% | cnt&})
```

C

short `ibrd (`short `ud,`char `*rd,` long `cnt)`

Input

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

Output

`rd` or `rdbufName`	Address of buffer into which data is read
Function Return	The value of `ibsta`

Description

Board Level

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

IBRD

Board Level
Device Level

IBRD (Continued)

Device Level

If `ud` is a device descriptor, `ibrd` addresses the GPIB, reads up to `cnt` bytes of data, and places the data into the buffer specified by `rd` or `rdbufName`. 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 `ibcnt`.

Possible Errors

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

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

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

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

ECIC — Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

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

EOIP — Asynchronous I/O is in progress.

IBRDA

Board Level
Device Level

Purpose

Read data asynchronously from a device into a user buffer.

Format

QuickBASIC

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

or

```
CALL ibrda (ud%,{a%(0)|a!(0)|a#(0)},{cnt%|cnt&})
```

C

short `ibrda` (short `ud`, char `*rd`, long `cnt`)

Input

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

Output

`rd` or `rdbufName`	Address of buffer into which data is read
Function Return	The value of `ibsta`

IBRDA

Board Level
Device Level

IBRDA (Continued)

Description

Board Level

If `ud` is a board descriptor, `ibrda` reads up to `cnt` bytes of data from a GPIB device and places the data into the buffer specified by `rd` or `rdbufName`. 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, the CIC sends the Device Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcnt`.

Device Level

If `ud` is a device descriptor, `ibrda` addresses the GPIB, begins an asynchronous read of up to `cnt` bytes of data from a GPIB device, and places the data into the memory location specified by `rd` or `rdbufName`. 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 `ibcnt`.

Board and Device Level

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

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

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

IBRDA

Board Level
Device Level

IBRDA (Continued)

Possible Errors

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

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

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

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

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

EOIP Asynchronous I/O is in progress.

IBRDF

Board Level
Device Level

Purpose

Read data from a device into a file.

Format

QuickBASIC

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

C

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

Input

`ud`	Board or device descriptor
`flname`	Name of file into which data is read

Output

Function Return	The value of `ibsta`

Description

Board Level

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

IBRDF

Board Level
Device Level

IBRDF (Continued)

Device Level

If `ud` is a device descriptor, `ibrdf` addresses the GPIB, reads up to `cnt` bytes of data from a GPIB device, and places the data into the file specified by `flname`. 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 `ibcnt`.

Possible Errors

EABO	Either `cnt` bytes or END was not received within the timeout period, or `ud` is a board descriptor and Device Clear was received after the read operation began.
EADR	Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB. Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.
EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
EFSO	`ibrdf` could not access `flname`.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRPP

Board Level
Device Level

Purpose

Conduct a parallel poll.

Format

QuickBASIC

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

C

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

Input

`ud`	Board or device descriptor

Output

`ppr` or `pollbufName`	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` or `pollbufName.`

For more information on parallel polling, refer to the *NI-488.2 User Manual for Macintosh.*

IBRPP

Board Level
Device Level

IBRPP (Continued)

Possible Errors

EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBRSC Board Level IBRSC

Purpose

Request or release system control.

Format

QuickBASIC

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

C

short `ibrsc (` short `ud`, short `v)`

Input

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

Output

Function Return	The value of `ibsta`

Description

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

IBRSC **Board Level** IBRSC
(Continued)

Possible Errors

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

IBRSP **Device Level** IBRSP

Purpose

Conduct a serial poll.

Format

QuickBASIC

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

C

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

Input

`ud`	Device descriptor

Output

`spr` or `pollbufName`	Serial poll response byte
Function Return	The value of `ibsta`

Description

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

For more information on serial polling, refer to the *NI-488.2 User Manual for Macintosh.*

IBRSP **Device Level** IBRSP (Continued)

Possible Errors

EABO	The serial poll response could not be read within the serial poll timeout period.
EARG	`ud` is a valid descriptor but does not refer to a device.
EBUS	No devices are connected to the GPIB.
ECIC	The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ESTB	Autopolling is enabled and the serial poll queue has overflowed. Disable automatic serial polling or call `ibrsp` more often to keep the queue from overflowing.

IBRSV **Board Level** IBRSV

Purpose

Request service and change the serial poll status byte.

Format

QuickBASIC

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

C

short `ibrsv` (short `ud`, short `v`)

Input

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

Output

Function Return	The value of `ibsta`

Description

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

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

IBRSV Board Level IBRSV (Continued)

Possible Errors

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

IBSAD

Board Level
Device Level

Purpose

Change or disable the secondary address.

Format

QuickBASIC

`CALL ibsad (ud%,v%)`

C

short `ibsad (` short `ud,` short `v)`

Input

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

Output

Function Return	The value of `ibsta`

Description

`ibsad` changes the secondary GPIB address of the board or device to `v`, an integer in the range 96 to 126 (hex 60 to hex 7E) or zero. If `v` is zero, secondary addressing is disabled. If no error occurs during the call, then the previous secondary address is returned in `iberr`.

IBSAD

Board Level
Device Level

IBSAD (Continued)

Possible Errors

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

IBSIC **Board Level** IBSIC

Purpose

Assert interface clear.

Format

QuickBASIC

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

C

short `ibsic (` short `ud)`

Input

`ud`	Board descriptor

Output

Function Return	The value of `ibsta`

Description

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

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

IBSIC **Board Level** IBSIC
(Continued)

Possible Errors

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

IBSRE **Board Level** IBSRE

Purpose

Set or clear the Remote Enable line.

Format

QuickBASIC

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

C

short `ibsre` (short `ud`, short `v`)

Input

`ud`	Board descriptor
`v`	Indicates whether to set or clear the REN line

Output

Function Return	The value of `ibsta`

Description

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

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

IBSRE Board Level IBSRE (Continued)

Possible Errors

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

IBSRQ **Board Level** IBSRQ

Purpose

Request an SRQ interrupt routine.

Format

QuickBASIC

Not supported.

C

short `ibsrq ( void (*func) (void) )`

Input

`func` C interrupt-handling routine

Description

`ibsrq` establishes a call to the C routine `func` whenever the SRQI bit is set in the status word (`ibsta`). If SRQI is set, the language interface calls `func` 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*** `func` ***from being called.***

IBSTOP

Board Level
Device Level

Purpose

Abort asynchronous I/O operation.

Format

QuickBASIC

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

C

```
short ibstop (short ud)
```

Input

`ud`	Board or device descriptor

Output

Function Return	The value of `ibsta`

Description

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

Possible Errors

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

IBTMO

Board Level
Device Level

Purpose

Change or disable the I/O timeout period.

Format

QuickBASIC

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

C

short `ibtmo` (short `ud`, short `v`)

Input

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

Output

Function Return	The value of `ibsta`

Description

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

IBTMO

Board Level
Device Level

IBTMO (Continued)

Table 1-8. Timeout Code Values

Constant	Value of v	Minimum Timeout
TNONE	0	disabled - no timeout
T10us	1	10 µs
T30us	2	30 µs
T100us	3	100 µs
T300us	4	300 µs
T1ms	5	1 ms
T3ms	6	3 ms
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms
T300ms	10	300 ms
T1s	11	1 s
T3s	12	3 s
T10s	13	10 s
T30s	14	30 s
T100s	15	100 s
T300s	16	300 s
T1000s	17	1000 s

Possible Errors

EARG v is invalid.

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

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

IBTRG **Device Level** IBTRG

Purpose

Trigger selected device.

Format

QuickBASIC

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

C

short `ibtrg` (short `ud`)

Input

`ud`	Device descriptor

Output

Function Return	The value of `ibsta`

Description

`ibtrg` sends the Group Execute Trigger (GET) message to the device described by `ud`.

IBTRG **Device Level** IBTRG (Continued)

Possible Errors

EARG	`ud` is a valid descriptor but does not refer to a device.
EBUS	No devices are connected to the GPIB.
ECIC	The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh*.
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

IBUNLOCK

Board Level
Device Level

Purpose

Unlock access to a GPIB-ENET board or device.

Format

QuickBASIC

```
CALL ibunlock (ud%)
```

C

```
short ibunlock ( short ud)
```

Input

`ud`	A board or device descriptor

Output

Function Return	The value of `ibsta`

Description

The `ibunlock` function releases the lock on the board or device connection requested by `iblock.`

Board Level

When the `iblock` function has been used to lock access to a board, an `ibunlock` function call made with the same board descriptor unlocks access to the board.

Device Level

When the `iblock` function has been used to lock access to a device, an `ibunlock` function call made with the same device descriptor unlocks access to the device.

IBUNLOCK

Board Level
Device Level

IBUNLOCK (Continued)

Recommended Usage

In general, use `ibunlock` to release your lock on a board or device connection. It is recommended that `ibunlock` be used immediately after critical board or device accesses are made to a locked interface.

Refer also to *IBLOCK*.

Possible Errors

EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ELCK	Occurs if the GPIB-ENET board or device being locked is locked by another process.

IBWAIT

Board Level
Device Level

Purpose

Wait for GPIB events.

Format

QuickBASIC

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

C

short `ibwait` (short `ud`, short `mask`)

Input

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

Output

Function Return	The value of `ibsta`

Description

`ibwait` monitors the events specified by `mask` and delays processing until one or more of the events occurs. If the wait mask is zero, `ibwait` returns immediately with the updated `ibsta` status word. If TIMO is set in the wait mask, `ibwait` returns when the timeout period has elapsed (if one or more of the other specified events have not already occurred). If TIMO is not set in the wait mask, then the function waits indefinitely for one or more of the specified events to occur. The `ibwait` mask bits are identical to the `ibsta` bits and they are described in Table 1-9. If `ud` is a device descriptor, the only valid wait mask bits are TIMO, END, RQS and CMPL. If `ud` is a board descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period using the `ibtmo` function.

IBWAIT

Board Level
Device Level

IBWAIT (Continued)

Possible Errors

EARG The bit set in `mask` is invalid.

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

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

ESRQ Device level: If RQS is set in the wait mask, then ESRQ indicates that the *Stuck SRQ* condition exists. For more information on serial polling, refer to the *NI-488.2 User Manual for Macintosh.*

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
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

Purpose

Write data to a device from a user buffer.

Format

QuickBASIC

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

or

```
CALL ibwrt (ud%,{a%(0)|a&(0)|a!(0)|a#(0)},{cnt%|cnt&})
```

C

```
short ibwrt (short ud,char *wrt,long cnt)
```

Input

`ud`	Board or device descriptor
`wrt`	Address of the buffer containing the bytes to write
`cnt`	Number of bytes to be written

Output

Function Return	The value of `ibsta`

Description

Board Level

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

IBWRT

Board Level
Device Level

IBWRT (Continued)

Device Level

If `ud` is a device descriptor, `ibwrt` addresses the GPIB and writes `cnt` bytes from the memory location specified by `wrt` 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 `ibcnt`.

Possible Errors

EABO	Either `cnt` bytes were not sent within the timeout period, or a Device Clear message was received after the read operation began.
EADR	Board level: The GPIB is not correctly addressed. Use `ibcmd` to address the GPIB. Device level: A conflict exists between the device GPIB address and the GPIB address of the device access board. Use `ibpad` and `ibsad`.
EBUS	Device level: No devices are connected to the GPIB.
ECIC	Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*
EDVR	Either `ud` is invalid or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners were detected on the bus.
EOIP	Asynchronous I/O is in progress.

IBWRTA

Board Level
Device Level

Purpose

Write data asynchronously to a device from a user buffer.

Format

QuickBASIC

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

or

```
CALL ibwrt (ud%,{a%(0)|a&(0)|a!(0)|a#(0)},{cnt%|cnt&})
```

C

```
short ibwrta (short ud, char *wrt, long cnt)
```

Input

`ud`	Board or device descriptor
`wrt`	Address of the buffer containing the bytes to write
`cnt`	Number of bytes to be written

Output

Function Return	The value of `ibsta`

Description

Board Level

If `ud` is a board descriptor, `ibwrta` begins an asynchronous write of `cnt` bytes of data from the buffer pointed to by `wrt` 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, the CIC sends the Device

IBWRTA

Board Level
Device Level

IBWRTA (Continued)

Clear message on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcnt`.

Device Level

If `ud` is a device descriptor, `ibwrta` addresses the GPIB and writes `cnt` bytes from the buffer `wrt` 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 `ibcnt`.

Board and Device Level

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

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

- `ibwait` — If the returned `ibsta` mask has the CMPL bit set, then the driver and application are resynchronized.
- `ibstop` — The I/O is canceled; the driver and application are resynchronized.
- `ibonl` — The I/O is canceled and the interface is reset; the driver and application are resynchronized.

IBWRTA

Board Level
Device Level

IBWRTA (Continued)

Possible Errors

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

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

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

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

ECIC — Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

ENOL — No Listeners were detected on the bus.

EOIP — Asynchronous I/O is in progress.

IBWRTF

Board Level
Device Level

Purpose

Write data to a device from a file.

Format

QuickBASIC

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

C

```
short ibwrtf (short ud,char flname[])
```

Input

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

Output

Function Return	The value of `ibsta`

Description

Board Level

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

IBWRTF

Board Level
Device Level

IBWRTF
(Continued)

Device Level

If `ud` is a device descriptor, `ibwrtf` addresses the GPIB and writes `cnt` bytes from the file `flname` 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 `ibcnt`.

Possible Errors

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

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

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

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

ECIC Device level: The access board is not CIC. See the *Device-Level Calls and Bus Management* section in Chapter 5 of the *NI-488.2 User Manual for Macintosh.*

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

EFSO `ibwrtf` could not access `flname`.

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

EOIP Asynchronous I/O is in progress.

Chapter 2
NI-488.2 Routines

This chapter lists the available NI-488.2 routines and 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, *Developing Your Application*, and Chapter 5, *GPIB Programming Techniques*, in the *NI-488.2 User Manual for Macintosh.*

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

- MPW C version 3.0 or higher , THINK C version 4.0 or higher, and Metrowerks CodeWarrior version 1.1 or higher
- Microsoft QuickBASIC version 1.0 or higher

Input and Output

The input and output parameters for each routine are listed. Most of the NI-488.2 routines have an input parameter which is either a single address or a list of addresses. The address parameter is a 16-bit integer that has two components: the low byte is a valid primary address (0 to 30), and the high byte is a valid secondary address (`NO_SAD(0)` or 96 to 126). A list of addresses is an array of single addresses. You must mark the end of this list with the constant `NOADDR`. An empty address list is either an array with only the `NOADDR` constant in it, or a `NULL` pointer.

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 ID to be a Listener in preparation for `RcvRespMsg`
`ResetSys`	Reset and initialize IEEE 488.2-compliant devices
`Send`	Send data bytes to a device

(continues)

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

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

AllSpoll AllSpoll

Purpose

Serial poll all devices.

Format

QuickBASIC

```
CALL AllSpoll (board%,addresslist%(0),resultlist%(0))
```

C

void `AllSpoll` (short `board`, short `addresslist` [], short `resultlist` [])

Input

`board`	The interface board number
`addresslist`	A list of device addresses that is terminated by `NOADDR`

Output

`resultlist`	A list of serial poll response bytes corresponding to device addresses in `addresslist`

Description

`AllSpoll` serial polls all of the devices described by `addresslist`. It stores the poll responses in `resultlist` and the number of responses in `ibcnt`.

AllSpoll

AllSpoll (Continued)

Possible Errors

EABO	One of the devices timed out instead of responding to the serial poll; `ibcnt` contains the index of the timed-out device.
EARG	An invalid address (out of range) appears in `addresslist`; `ibcnt` is the index of the invalid address in the `addresslist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

DevClear

Purpose

Clear a single device.

Format

QuickBASIC

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

C

```
void DevClear ( short board,short address)
```

Input

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

Description

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

Possible Errors

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

DevClearList DevClearList

Purpose

Clear multiple devices.

Format

QuickBASIC

```
CALL DevClearList (board%,addresslist%(0))
```

C

```
void DevClearList (short board,short addresslist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses terminated by `NOADDR` that you want to clear

Description

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

DevClearList DevClearList

(Continued)

Possible Errors

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

EnableLocal EnableLocal

Purpose

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

Format

QuickBASIC

```
CALL EnableLocal (board%,addresslist%(0))
```

C

```
void EnableLocal (short board, short addresslist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses that is terminated by `NOADDR`

Description

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

EnableLocal

Possible Errors

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

EnableRemote

Purpose

Enable remote GPIB programming for devices.

Format

QuickBASIC

```
CALL EnableRemote (board%,addresslist%(0))
```

C

```
void EnableRemote (short board,short addresslist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses that is terminated by `NOADDR`

Description

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

EnableRemote

EnableRemote (Continued)

Possible Errors

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

FindLstn FindLstn

Purpose

Find listening devices on the GPIB.

Format

QuickBASIC

```
CALL FindLstn (board%,addresslist%(0),resultlist%(0),limit%)
```

C

```
void FindLstn (short board,short addresslist [],short resultlist [],short
        limit)
```

Input

`board`	The interface board number
`addresslist`	A list of primary addresses that is terminated by `NOADDR`
`limit`	Total number of entries that can be placed in `resultlist`

Output

`resultlist`	Addresses of all listening devices found by `FindLstn` are placed in this array.

Description

`FindLstn` tests all of the primary addresses in `addresslist` as follows:

If a device is present at a primary address given in `addresslist`, then the primary address is stored in `resultlist`. Otherwise, all secondary addresses of the primary address are tested, and the addresses of any devices found are stored in `resultlist`. No more than limit addresses are stored in `resultlist`; `ibcnt` contains the actual number of addresses stored in `resultlist`.

FindLstn

FindLstn (Continued)

Possible Errors

EARG	An invalid primary address (out of range) appears in `addresslist`; `ibcnt` is the index of the invalid address in the `addresslist` array.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ETAB	The number of devices found on the GPIB exceed `limit`.

FindRQS FindRQS

Purpose

Determine which device is requesting service.

Format

QuickBASIC

```
CALL FindRQS (board%,addresslist%(0),result%)
```

C

```
void FindRQS (short board,short addresslist[],short *result)
```

Input

`board`	The interface board number
`addresslist`	List of device addresses that is terminated by `NOADDR`

Output

`result`	Serial poll response byte of the device that is requesting service

Description

`FindRQS` serial polls the devices described by `addresslist`, in order, until it finds a device which is requesting service. The serial poll response byte is then placed in `result`. `ibcnt` contains the index of the device requesting service in `addresslist`. If none of the devices are requesting service, then the index corresponding to `NOADDR` in `addresslist` is returned in `ibcnt` and ETAB is returned in `iberr`.

FindRQS

(Continued)

Possible Errors

EARG	An invalid address (out of range) appears in `addresslist`; `ibcnt` is the index of the invalid address in the `addresslist` array.
EBUS	No devices are connected to the GPIB.
ECIC	`board` is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	`board` is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.
ETAB	None of the devices in `addresslist` are requesting service or `addresslist` contains only `NOADDR`. `ibcnt` contains the index of `NOADDR` in `addresslist`.

PassControl PassControl

Purpose

Pass control to another device with Controller capability.

Format

QuickBASIC

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

C

```
void PassControl (short board,short address)
```

Input

`board`	The interface board number
`address`	Address of the device to which you want to pass control

Description

`PassControl` sends the Take Control (TCT) GPIB message to the device described by `address`. That device becomes Controller-In-Charge and `board` is no longer CIC.

Possible Errors

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

PPoll PPoll

Purpose

Perform a parallel poll on the GPIB.

Format

QuickBASIC

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

C

void `PPoll` (short `board`, short *`result`)

Input

`board`	The interface board number

Output

`result`	The parallel poll result

Description

`PPoll` conducts a parallel poll and the result is placed in `result`. Each of the eight bits of `result` represents the status information for each device configured for a parallel poll. The interpretation of the status information is based on the latest parallel poll configuration command sent to each device (see `PPollConfig` and `PPollUnconfig`). The Controller can use parallel polling to obtain one-bit, device-dependent status messages from up to eight devices simultaneously.

For more information on parallel polling, refer to the *NI-488.2 User Manual for Macintosh.*

PPoll

PPoll (Continued)

Possible Errors

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

PPollConfig PPollConfig

Purpose

Configure a device to respond to parallel polls.

Format

QuickBASIC

```
CALL PPollConfig (board%,address%,dataline%,sense%)
```

C

void `PPollConfig` (short `board` , short `address` , short `dataline` , short `sense`)

Input

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

Description

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

For more information on parallel polling, refer to the *NI-488.2 User Manual for Macintosh.*

PPollConfig

PPollConfig (Continued)

Possible Errors

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

PPollUnconfig PPollUnconfig

Purpose

Unconfigure devices for parallel polls.

Format

QuickBASIC

```
CALL PPollUnconfig (board%,addresslist%(0))
```

C

```
void PPollUnconfig ( short board,short addresslist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses that is terminated by `NOADDR`

Description

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

For more information on parallel polling, refer to the *NI-488.2 User Manual for Macintosh.*

PPollUnconfig

PPollUnconfig (Continued)

Possible Errors

EARG — An invalid address (out of range) appears in `addresslist`; `ibcnt` is the index of the invalid address in the `addresslist` array.

EBUS — No devices are connected to the GPIB.

ECIC — The interface board is not the Controller-In-Charge; see `SendIFC`.

EDVR — Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.

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

EOIP — Asynchronous I/O is in progress.

RcvRespMsg RcvRespMsg

Purpose

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

Format

QuickBASIC

```
CALL RcvRespMsg (board%,data$,cnt%,termination%)
```

C

void `RcvRespMsg` (short `board`, char `data` [], long `cnt`, short `termination`)

Input

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

Output

`data`	Stores the received data bytes

Description

`RcvRespMsg` reads up to `cnt` bytes from the GPIB and places these bytes into `data`. Data bytes are read until either `cnt` data bytes have been read or the termination condition is detected. If the termination condition is `STOPend`, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when the 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable `ibcnt`.

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

RcvRespMsg

RcvRespMsg (Continued)

Possible Errors

EABO	The I/O timeout period elapsed before all the bytes were received.
EADR	The interface board is not in the listen-active state; use `ReceiveSetup` to address the GPIB properly.
EARG	The `termination` parameter is invalid. It must be either `STOPend` or an 8-bit EOS character.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
EOIP	Asynchronous I/O is in progress.

ReadStatusByte ReadStatusByte

Purpose

Serial poll a single device.

Format

QuickBASIC

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

C

void `ReadStatusByte (`short `board,`short `address,`short `*result)`

Input

`board`	The interface board number
`address`	A device address

Output

`result`	Serial poll response byte

Description

`ReadStatusByte` serial polls the device described by `address`. The response byte is stored in `result`.

ReadStatusByte

ReadStatusByte (Continued)

Possible Errors

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

Receive

Receive

Purpose

Read data bytes from a device.

Format

QuickBASIC

```
CALL Receive (board%,address%,data$,cnt%,termination%)
```

C

```
void Receive (short board,short address,char data [],unsigned long cnt, short
                termination)
```

Input

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

Output

`data`	Stores the received data bytes

Description

`Receive` addresses the device described by `address` to talk and the interface board to listen. Then up to `cnt` bytes are read and placed into the buffer. Data bytes are read until either `cnt` bytes have been read or the termination condition is detected. If the termination condition is `STOPend`, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when the 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable `ibcnt`.

Receive

Receive (Continued)

Possible Errors

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

ReceiveSetup ReceiveSetup

Purpose

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

Format

QuickBASIC

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

C

```
void ReceiveSetup ( short board,short address)
```

Input

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

Description

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

ReceiveSetup

ReceiveSetup (Continued)

Possible Errors

EARG — The `address` parameter is invalid (out of range).

EBUS — No devices are connected to the GPIB.

ECIC — The interface board is not the Controller-In-Charge; see `SendIFC`.

EDVR — Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.

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

EOIP — Asynchronous I/O is in progress.

ResetSys ResetSys

Purpose

Reset and initialize IEEE 488.2-compliant devices.

Format

QuickBASIC

```
CALL ResetSys (board%,addresslist%(0))
```

C

void `ResetSys` (short `board`,short `addresslist` [])

Input

`board`	The interface board number
`addresslist`	A list of device addresses that is terminated by `NOADDR`

Description

The reset and initialization take place in three steps. The first step resets the GPIB by asserting the Remote Enable (REN) line and then the Interface Clear (IFC) line. The second step clears all of the devices by sending the Universal Device Clear (DCL) GPIB message. The final step causes IEEE 488.2-compliant devices to perform device-specific reset and initialization. This step is accomplished by sending the message `"*RST\n"` to the devices described by `addresslist.`

ResetSys

ResetSys (Continued)

Possible Errors

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

Send

Purpose

Send data bytes to a device.

Format

QuickBASIC

```
CALL Send (board%,address%,data%,cnt%,eotmode%)
```

C

void `Send (`short `board`, short `address`, char `data []`, long `cnt`, short `eotmode)`

Input

`board`	The interface board number
`address`	Address of a device to which data is sent
`data`	The data bytes to be sent
`cnt`	Number of bytes to be sent
`eotmode`	The data termination mode: `DABend`, `NULLend`, or `NLend`

Description

`Send` addresses the device described by `address` to listen and the interface board to talk. Then `cnt` bytes from `data` are sent to the device. The last byte is sent with the EOI line asserted if `eotmode` is `DABend`. The last byte is sent *without* the EOI line asserted if `eotmode` is `NULLend`. If `eotmode` is `NLend` then a new line character (`'\n'`) is sent with the EOI line asserted after the last byte of `data`. The actual number of bytes transferred is returned in the global variable `ibcnt`.

Send

Send (Continued)

Possible Errors

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

SendCmds SendCmds

Purpose

Send GPIB command bytes.

Format

QuickBASIC

```
CALL SendCmds (board%,commands$,cnt%)
```

C

```
void SendCmds (short board,char commands[],unsigned long cnt)
```

Input

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

Description

`SendCmds` sends `cnt` command bytes from `commands` over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable `ibcnt`. Refer to Appendix A, *Multiline Interface Messages*, for a listing of the defined interface messages.

Use command bytes to configure the state of the GPIB, not to send instructions to GPIB devices. Use `Send` or `SendList` to send device-specific instructions.

SendCmds

SendCmds (Continued)

Possible Errors

EABO	The I/O timeout period expired before all of the command bytes were sent.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No devices are connected to the GPIB.
EOIP	Asynchronous I/O is in progress.

SendDataBytes SendDataBytes

Purpose

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

Format

QuickBASIC

```
CALL SendDataBytes (board%,data$,cnt%,eotmode%)
```

C

void `SendDataBytes` (short `board`, char `data` [], long `cnt`, short `eotmode`)

Input

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

Description

`SendDataBytes` sends `cnt` number of bytes from the buffer to devices which are already addressed to listen. The last byte is sent with the EOI line asserted if `eotmode` is `DABend`; the last byte is sent *without* the EOI line asserted if `eotmode` is `NULLend`. If `eotmode` is `NLend` then a new line character (`'\n'`) is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global variable `ibcnt`.

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

SendDataBytes

SendDataBytes (Continued)

Possible Errors

EABO	The I/O timeout period expired before all of the bytes were sent.
EADR	The 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 `data` is empty and the `eotmode` is `DABend`.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners are on the GPIB to accept the data bytes; use `SendSetup` to address the GPIB properly.
EOIP	Asynchronous I/O is in progress.

SendIFC SendIFC

Purpose

Reset the GPIB by sending interface clear.

Format

QuickBASIC

```
CALL SendIFC (board%)
```

C

void `SendIFC` (short `board`)

Input

`board`	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. It also ensures that the connected devices are all unaddressed and that the interface functions of the devices are in their idle states.

Possible Errors

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

SendList SendList

Purpose

Send data bytes to multiple GPIB devices.

Format

QuickBASIC

```
CALL SendList (board%,addresslist%(0),data$,cnt%,eotmode%)
```

C

```
void SendList (short board,short addresslist [],char data [],long cnt, short
               eotmode)
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses to send data to
`data`	The data bytes to be sent
`cnt`	Number of bytes transmitted
`eotmode`	The data termination mode: `DABend`, `NULLend`, or `NLend`.

Description

`SendList` addresses the devices described by `addresslist` to listen and the interface board to talk. Then, `cnt` bytes from buffer are sent to the devices. The last byte is sent with the EOI line asserted if `eotmode` is `DABend`. The last byte is sent *without* the EOI line asserted if `eotmode` is `NULLend`. If `eotmode` is `NLend`, then a new line character (`'\n'`) is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global variable `ibcnt`.

SendList

SendList (Continued)

Possible Errors

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

SendLLO SendLLO

Purpose

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

Format

QuickBASIC

```
CALL SendLLO (board%)
```

C

void `SendLLO (` short `board)`

Input

`board`	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 most cases, use `SetRWLS` to place devices in Remote With Lockout State.

SendLLO

SendLLO (Continued)

Possible Errors

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

SendSetup SendSetup

Purpose

Set up devices to receive data in preparation for `SendDataBytes`.

Format

QuickBASIC

```
CALL SendSetup (board%,addresslist%(0))
```

C

```
void SendSetup (short board,short addresslist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses that is terminated by `NOADDR`

Description

`SendSetup` makes the devices described by `addresslist` listen-active and makes the interface board talk-active. This call is usually followed by `SendDataBytes` to actually transfer data from the interface board to the devices. `SendSetup` is particularly useful to set up the addressing before making multiple calls to `SendDataBytes`; it eliminates the need to readdress the devices for every block of data.

SendSetup

SendSetup (Continued)

Possible Errors

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

SetRWLS SetRWLS

Purpose

Place devices in Remote With Lockout State.

Format

QuickBASIC

```
CALL SetRWLS (board%,addresslist%(0))
```

C

void `SetRWLS (` short `board,` short `addresslist` [])

Input

`board`	The interface board number
`addresslist`	A list of device addresses terminated by `NOADDR`

Description

`SetRWLS` places the devices described by `addresslist` 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.

SetRWLS

SetRWLS (Continued)

Possible Errors

EARG — An invalid address (out of range) appears in `addresslist` (`ibcnt` is the index of the invalid address in the `addresslist` array), or the `addresslist` is empty.

EBUS — No devices are connected to the GPIB.

ECIC — The interface board is not the Controller-In-Charge; see `SendIFC`.

EDVR — Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.

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

EOIP — Asynchronous I/O is in progress.

ESAC — The interface board is not configured as System Controller.

TestSRQ TestSRQ

Purpose

Determine the current state of the GPIB Service Request (SRQ) line.

Format

QuickBASIC

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

C

void `TestSRQ (` short `board,` short `*result)`

Input

`board`	The interface board number

Output

`result`	State of the SRQ line: non-zero if the line is asserted, zero if the line is not asserted

Description

`TestSRQ` returns the current state of the GPIB SRQ line in `result.` If SRQ is asserted, then `result` contains a non-zero value. Otherwise, `result` contains a zero. Use `TestSRQ` to get the current state of the GPIB SRQ line. Use `WaitSRQ` to wait until SRQ is asserted.

Possible Errors

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

TestSys

Purpose

Cause IEEE 488.2-compliant devices to conduct self tests.

Format

QuickBASIC

```
CALL TestSys (board%,addresslist%(0),resultlist%(0))
```

C

```
void TestSys (short board,short addresslist [],short resultlist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses terminated by `NOADDR`

Output

`resultlist`	A list of test results; each entry corresponds to an address in `addresslist`

Description

`TestSys` sends the `"*TST\n"` message to the IEEE 488.2-compliant devices described by `addresslist`. The `"*TST\n"` message instructs them to conduct their self-test procedures. A 16-bit test result code is read from each device and stored in `resultlist`. A test result of `0\n` indicates that the device passed its self test. Any other value indicates that the device failed its self test. Refer to the manual that came with your device to determine the meaning of the failure code. A test result of -1 indicates that the I/O timeout period elapsed before the device sent its result code. `ibcnt` 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; `ibcnt` 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 `addresslist` (`ibcnt` is the index of the invalid address in the `addresslist` array), or the `addresslist` is empty.
EBUS	No devices are connected to the GPIB.
ECIC	The interface board is not the Controller-In-Charge; see `SendIFC`.
EDVR	Either `board` is invalid (out of range) or the NI-488.2 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners are on the GPIB.
EOIP	Asynchronous I/O is in progress.

Trigger Trigger

Purpose

Trigger a device.

Format

QuickBASIC

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

C

void `Trigger (` short `board,` short `address)`

Input

`board`	The interface board number
`address`	Address of a device to be triggered

Description

`Trigger` sends the Group Execute Trigger (GET) GPIB message to the device described by `address`. If `address` is the constant `NOADDR`, the Group Execute Trigger message is sent to all devices that are currently listen-active on the GPIB.

Possible Errors

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

TriggerList TriggerList

Purpose

Trigger multiple devices.

Format

QuickBASIC

```
CALL TriggerList (board%,addresslist%(0))
```

C

```
void TriggerList (short board,short addresslist [])
```

Input

`board`	The interface board number
`addresslist`	A list of device addresses terminated by `NOADDR`

Description

`TriggerList` sends the Group Execute Trigger (GET) GPIB message to the devices included in `addresslist`. If `addresslist` contains only `NOADDR`, the Group Execute Trigger message is sent to all devices that are currently listen-active on the GPIB.

TriggerList

TriggerList (Continued)

Possible Errors

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

WaitSRQ WaitSRQ

Purpose

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

Format

QuickBASIC

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

C

void `WaitSRQ` (short `board`, short *`result`)

Input

`board`	The interface board number

Output

`result`	State of the SRQ line: non-zero if line is asserted, zero if line not asserted

Description

`WaitSRQ` waits until either the GPIB SRQ line is asserted or the timeout period has expired (see `ibtmo`). When `WaitSRQ` returns, `result` contains a non-zero value if SRQ is asserted. Otherwise, `result` contains a zero. Use `TestSRQ` to get the current state of the GPIB SRQ line. Use `WaitSRQ` to wait until SRQ is asserted.

Possible Errors

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

Chapter 3
Device Manager Functions and Routines

This chapter describes the purpose, format, input and output parameters, and possible errors for each Device Manager function and routine.

While using the functions and routines, you might find it helpful to refer to Chapter 2, *Developing Your Application*, and Chapter 5, *GPIB Programming Techniques,* in the *NI-488.2 User Manual for Macintosh.*

Function and Routine Names

The calls in this chapter are listed alphabetically—NI-488 functions first, then NI-488.2 routines.

Purpose

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

Control Number

Thecontrol number given for each function or routine is a constant that indicates which GPIB function call to make.

Parameter Block Fields

The function and routine descriptions include parameter block fields that describe the parameters of the specific GPIB functions.

Description

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

Examples

All examples are from the C language interface, using high-level Device Manager calls. All languages that permit toolbox calls can access the NI-488.2 driver in a similar manner.

List of NI-488 Device Manager Functions

The following table contains an alphabetical list of each NI-488 control call.

Table 3-1. NI-488 Device Manager Control Calls

Constant	Fields	Cntrl No.	Description
FC_ibLOCK	id	79	Lock access to GPIB-ENET device or board
FC_ibUNLOCK	id	80	Unlock access to GPIB-ENET device or board
ibASK	id, controlVar, IOBufPtr	81	Return information about software configuration parameters
ibBNA	id, IOBufPtr	73	Change access board of device
ibCAC	id, controlVar	0	Become Active Controller
ibCLR	id	22	Clear specified device
ibCMD	id, IOBufPtr, IOCount	1	Send commands asynchronously from string
ibCMDA	id, IOBufPtr, IOCount	39	Send commands from string
ibCONFIG	id, controlVar, IOCount	71	Change the driver configuration parameters
ibDEV	id	32	Open device by index and return unit descriptor
ibDMA	id, controlVar	16	Enable/disable DMA
ibEOS	id, controlVar	20	Change/disable End-Of-String (EOS) mode
ibEOT	id, controlVar	17	Enable/disable END message
ibFIND	id, IOBufPtr	25	Open device by name and return unit descriptor

(continues)

 © National Instruments Corp.

Table 3-1. NI-488 Device Manager Control Calls (Continued)

Constant	Fields	Cntrl No.	Description
ibGTS	id, controlVar	2	Go from Active Controller to Standby
ibIST	id, controlVar	3	Set/clear ist
ibLINES	id, IOBufPtr	42	Read state of GPIB handshake and control lines
ibLLO	id	15	Place devices in Local Lockout
ibLN	id, controlVar, IOCount, IOBufPtr	31	Check for presence of a device on the bus
ibLOC	id	4	Go to Local
ibONL	id, controlVar	5	Place device online/offline
ibPAD	id, controlVar	18	Change primary address
ibPCT	id	23	Pass control
ibPPC	id, controlVar	9	Parallel Poll Configure
ibRD	id, IOBufPtr, IOCount	6	Read data to string
ibRDA	id, IOBufPtr, IOCount	37	Read data asynchronously to string
ibRPP	id	7	Conduct a parallel poll
ibRSC	id, controlVar	8	Request/release system control
ibRSP	id	24	Return serial poll byte
ibRSV	id, controlVar	11	Request service
ibSAD	id, controlVar	19	Change secondary address
ibSIC	id	12	Send Interface Clear
ibSRE	id, controlVar	13	Set/clear Remote Enable line
ibSTOP	id	40	Abort asynchronous operation
ibTMO	id, controlVar	26	Change/disable time limit
ibTRG	id	21	Trigger selected devices
ibWAIT	id, controlVar	10	Wait for a selected event
ibWRT	id, IOBufPtr, IOCount	14	Write data from string
ibWRTA	id, IOBufPtr, IOCount	38	Write data asynchronously from string

IBASK

Purpose

Return information about software configuration parameters.

Control Number

81

Parameter Block Fields

```
short    id            ->  | board or device id
short    controlVar    ->  | option
short    IOBufPtr      ->  | value
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is a device or an interface board. `controlVar` selects the configuration item whose value is being returned. `IOBufPtr` points to a short integer where the current value of the selected configuration item is returned.

Device Function Example

Determine SAD of device.

```
paramBlk->id = devID;              /*  devID returned by       */
                                   /*  ibFIND control call     */
paramBlk->controlVar = ibaSAD;     /*  get SAD of device       */
paramBlk->IOBufPtr = &myint;       /*  return SAD of device    */
                                   /*  here                    */

osErr = Control (refNum,ibASK,&paramBlk);

/* ibASK = 81                                                  */
/* The ERR bit on ibsta is set if the ask option is out of     */
/* range or the interface is not capable of returning the      */
/* specified information.                                      */
```

IBASK IBASK

(Continued)

Board Function Examples

Determine PAD of board.

```
paramBlk->id = devID;              /*  devID returned by        */
                                   /*  ibFIND control call      */
paramBlk->controlVar = ibaPAD;     /*  get PAD of board  */
paramBlk->IOBufPtr = &myint;       /*  return PAD of board      */
                                   /*  here                     */

osErr = Control (refNum,ibASK,&paramBlk);

/* ibASK = 81                                                   */
/* The ERR bit on ibsta is set if the ask option is out of    */
/* range or the interface is not capable of returning the     */
/* specified information.                                       */
```

IBBNA IBBNA

Purpose

Change access board of device.

Control Number

73

Parameter Block Fields

```
short    id          <-  | board or device id
Ptr      IOBufPtr    ->  | pointer to board or device
                       name
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

`id` specifies a device. `bdname` specifies the new access board to be used in all device calls to that device. `ibbna` is needed only to alter the board assignment from its configuration setting.

The assigned board is used in all subsequent device functions used with that device until `ibbna` is called again, `ibonl` or `ibfind` is called, or the system is restarted.

When `ibbna` is called and an error does not occur, the previous access board number is stored in `ibcnt`.

IBCAC IBCAC

Purpose

Become Active Controller.

Control Number

0

Parameter Block Fields

```
short    id            ->  | board id
short    controlVar    ->  | 0 or 1
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is an interface board. If `controlVar` is non-zero, the GPIB Controller takes control synchronously with respect to data transfer operations. Otherwise, the GPIB Controller takes control immediately (and possibly asynchronously).

To take control synchronously, the GPIB interface asserts the ATN signal in such a way as to ensure that data being transferred on the GPIB is not corrupted. If a data handshake is in progress, the take-control action is postponed until the handshake is complete; if a handshake is not in progress, the take-control action is done immediately. Synchronous take control is not guaranteed if an `ibRD` or `ibWRT` operation completed with a timeout or error.

Asynchronous take-control should be used in situations where it appears to be impossible to gain control synchronously (for example, after a timeout error).

Most applications do not need to use the `ibCAC` function. Functions such as `ibCMD` and `ibRPP`, which require that the GPIB interface take control, do so automatically.

The ECIC error results if the GPIB interface is not Controller-In-Charge (CIC).

Board Function Examples

1. Take control immediately without regard to other handshakes in progress.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND    */
                               /*  control call                */
paramBlk->controlVar = 0;      /*  take control immediately    */
                               /*  without regard to any data  */
                               /*  handshake in progress       */
osErr = Control(refNum, ibCAC, &paramBlk);
/*  ibCAC = 0 */
```

IBCAC

IBCAC (Continued)

2. Take control synchronously and assert ATN following a read operation.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND  */
                               /*  control call              */
paramBlk->IOBufPtr = buf;
paramBlk->IOCount = 512;
osErr = Control(refNum, ibRD, & paramBlk);
/*  ibRD = 6  */
paramBlk->controlVar = 1;      /*  take control synchronously*/
osErr = Control(refNum, ibCAC, & paramBlk);
/*  ibCAC = 0 */
```

IBCLR IBCLR

Purpose

Clear specified device.

Control Number

22

Parameter Block Fields

```
short    id      ->  | Device id
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is a device.

The `ibCLR` function clears the internal or device functions of a specified device. On exit, all devices are unaddressed.

`ibCLR` calls the board `ibCMD` function to send the following commands using the designated access board.

- Listen address of the device
- Secondary address of the device, if applicable
- Selected Device Clear (SDC)
- Untalk (UNT) and Unlisten (UNL)

Other command bytes can be sent as necessary.

Refer to *IBCMD* for additional information. Also refer to the discussions of device functions and clearing the device and the GPIB in Chapter 2, *Developing Your Application*, in the *NI-488.2 User Manual for Macintosh.*

Device Function Example

Clear the device.

```
paramBlk->id = devID;          /*  devID returned by ibFIND   */
                               /*  control call               */
osErr = Control(refNum,ibCLR,&paramBlk);  /*  ibCLR = 22     */
```

IBCMD IBCMD

Purpose

Send commands from string.

Control Number

1

Parameter Block Fields

short	id	->	board id
Ptr	IOBufPtr	->	buffer pointer
long	IOCount	->	byte count
short	ibsta	<-	status variables
short	iberr	<-	returned
long	ibcnt	<-	

`id` is an interface board. `IOBufPtr` contains the commands to be sent over the GPIB. `IOCount` is the number of bytes to be sent over the GPIB.

The `ibCMD` function transmits interface messages (commands) over the GPIB. These commands, which are listed in Appendix A, *Multiline Interface Messages*, include device talk and listen addresses, secondary addresses, serial and parallel poll configuration messages, and device clear and trigger instructions. The `ibCMD` function also passes GPIB control to another device. Do *not* use this function to transmit programming instructions to devices. To transmit programming instructions and other device-dependent information, use the read and write functions.

The `ibCMD` operation terminates on any of the following events.

- All commands are successfully transferred.
- An error is detected.
- The time limit is exceeded.
- A Take Control (TCT) command is sent.
- An Interface Clear (IFC) message is received from the System Controller (not the GPIB interface).

After termination, the `ibcnt` variable contains the number of commands sent. A short count can occur on any event but the first.

An ECIC error results if the GPIB interface is not CIC. If it is not Active Controller, the GPIB interface takes control and asserts ATN prior to sending the command bytes. The GPIB interface remains Active Controller afterward.

IBCMD

IBCMD (Continued)

In the examples that follow, GPIB commands and addresses are coded as printable ASCII characters. When the hex values to be sent over the GPIB correspond to printable ASCII characters, it is simplest to use the ASCII characters to specify the values. Refer to Appendix A, *Multiline Interface Messages,* for conversions of hex values to ASCII characters.

Board Function Examples

1. Unaddress all Listeners with the Unlisten command (ASCII _), and address a Talker at hex 46 (ASCII F) and a Listener at hex 31 (ASCII 1).

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND */
                                /*  control call             */
paramBlk->IOCount = 3;          /*  3 bytes to transfer      */
buf[0] = '?';                   /*  UNL                      */
buf[1] = 'F'                    /*  TAD of 0x46              */
buf[2] = '1';                   /*  LAD of 0x31              */
paramBlk->IOBufPtr = buf;
osErr = Control(refNum, ibCMD, &paramBlk);
/*  ibCMD = 1   */
```

2. Same as Example 1 except Listener has secondary address of hex 6E (ASCII n).

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND */
                                /*  control call             */
paramBlk->IOCount = 4;          /*  4 bytes to transfer      */
buf[0] = '?';                   /*  UNL                      */
buf[1] = 'F';                   /*  TAD of 0x46              */
buf[2] = '1';                   /*  LAD of 0x31              */
buf[3] = 'n';                   /*  SAD of 0x6E              */
paramBlk->IOBufPtr = buf;
osErr = Control(refNum, ibcmd, &paramBlk);
/*  ibCMD = 1 */
```

3. Clear all GPIB devices (that is, reset internal functions) with the Device Clear command (hex 14, an unprintable ASCII character).

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND */
                                /*  control call             */
paramBlk->IOCount = 1;
buf[0] = 0x14;                  /*  DCL                      */
osErr = Control(refNum,ibCMD,&paramBlk);
/*  ibCMD = 1 */
```

IBCMD

IBCMD (Continued)

4. Clear two devices with listen addresses of hex 21 (ASCII !) and hex 28 (ASCII (left parenthesis)) with the Selected Device Clear command (hex 04).

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND */
                               /*  control call             */
paramBlk->IOCount = 4;         /*  4 bytes to transfer      */
buf[0] = '?';                  /*  UNL                      */
buf[1] = '!';                  /*  LAD of 0x21              */
buf[2] = '(';                  /*  LAD of 0x28              */
buf[3] = 4;                    /*  SDC                      */
paramBlk->IOBufPtr = buf;
osErr = Control(refNum,ibCMD,&paramBlk);
/*  ibCMD = 1 */
```

5. Trigger any device previously addressed to listen with the Group Execute Trigger command (hex 08).

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND
*/
                                 /*  control call             */
paramBlk->IOCount = 1;           /*  1 byte to transfer       */
buf[0] = 0x8;                    /*  GET                      */
paramBlk->IOBufPtr = buf;
osErr = Control(refNum,ibCMD,&paramBlk);
/*  ibCMD = 1 */
```

IBCMD IBCMD

(Continued)

---

6. Unaddress all Listeners and serial poll the device at talk address hex 52 (ASCII R) using the Serial Poll Enable (hex 18, an unprintable ASCII character) and Serial Poll Disable (hex 19, another unprintable ASCII character) commands. (The GPIB interface listen address is hex 20 or ASCII blank.)

```
paramBlk->id = brdID;              /* brdID returned by ibFIND
*/
                                   /* control call           */
paramBlk->IOCount = 4;
buf[0] = '?'                       /* UNL                    */
buf[1] = 'R';                      /* TAD                    */
buf[2] = '  ';                     /* MLA                    */
buf[3] = 0x18                      /* SPE                    */
paramBlk->IOBufPtr = buf;
osErr = Control(refNum,ibCMD,& paramBlk)
/* ibCMD = 1 */
paramBlk->IOCount = 1;
paramBlk->IOBufPtr = rdBuf;
osErr = Control(refNum,ibRD,& paramBlk)
/*  After checking the status byte on rd[0], disable this   */
/*  device and unaddress it with the Untalk (UNT or ASCII _) */
/*  command before pulling the next device.                  */
buf[0] = 0x19;                     /* SPD                    */
buf[1] = '_';                      /* UNT                    */
paramBlk->IOCount = 2
paramBlk->IOBufPtr = buf;
osErr = Control(refNum,ibCMD,&paramBlk):
/* ibCMD = 1  */
```

IBCMDA IBCMDA

Purpose

Send commands asynchronously from string.

Control Number

39

Parameter Block Fields

short	id	->	board id
Ptr	IOBufPtr	->	buffer pointer
long	IOCount	->	byte count
short	ibsta	<-	status variables
short	iberr	<-	returned
long	ibcnt	<-	

`id` is an interface board. `IOBufPtr` contains the commands to be sent over the GPIB. `IOCount` is the number of bytes to be sent over the GPIB.

The `ibCMDA` function transmits interface messages (commands) over the GPIB. These commands, which are listed in Appendix A, *Multiline Interface Messages*, include device talk and listen addresses, secondary addresses, serial and parallel poll configuration messages, and device clear and trigger instructions. You can also use the `ibcmda` function to pass GPIB control to another device. Do *not* use this function to transmit programming instructions to devices. Transmit programming instructions and other device-dependent information with the write function.

Use `ibCMDA` instead of `ibCMD` when the application program must perform other functions while processing the GPIB I/O operation. `ibCMDA` returns after starting the I/O operation. If the number of bytes to send is small and the bytes are accepted quickly by the GPIB device(s), the operation may complete on the initial call. In this case, the CMPL bit is set in `ibsta`. If the operation does not complete on the initial call, monitor the `ibsta` variable after subsequent calls (usually `ibwait` calls) until the I/O is completed. When CMPL becomes set in `ibsta`, indicating that the I/O is complete, the number of bytes sent is reported in the `ibcnt` variable.

After `ibCMDA` is called and before the corresponding CMPL, other GPIB function calls to this board return the error EOIP, with the following exceptions.

- `ibSTOP`
- `ibWAIT`
- `ibONL`

The asynchronous I/O started by `ibCMDA` terminates for the same reasons that `ibCMD` terminates.

IBCMDA IBCMDA

(Continued)

An ECIC error results if the GPIB board is not CIC. If it is not Active Controller, the GPIB board takes control and asserts ATN prior to sending the command bytes. It remains Active Controller afterwards. The ENOL error does *not* occur if there are no Listeners.

Board Function Example

Address several devices for a broadcast message to follow while testing for a high priority event to occur.

```
/*   The interface board brd0 at talk address 0x40 (ASCII @)   */
/*   addresses nine Listeners at addresses 0x31-0x39           */
/*   (ASCII 1-9) to receive the broadcast message.             */

   paramBlk->id = brdID;           /*  brdID returned by       */
                                   /*  ibFIND                  */
                                   /*  control call            */
   paramBlk->IOCount = 11;         /*  11 bytes to transfer    */
   paramBlk->IOBufPtr = "?@123456789";
   osErr = Control(refNum,ibCMDA,&paramBlk);
   /*     ibCMDA = 39       */

/*   Call unspecified routine to test and process a high priority */
/*   event.*/

    do {
        eventtst();

/*   Get status.              */

        ibwait (brd0,0);

/*   Loop until complete.     */

    } while ((ibsta & CMPL) == 0);
```

IBCONFIG IBCONFIG

Purpose

Change the driver configuration parameters.

Control Number

71

Parameter Block Fields

```
short    id           ->  | board id
short    controlVar   ->  | termination character
long     IOCount      ->  | number of bytes to receive
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` specifies a GPIB interface board or a device. `controlVar` is used to select the configurable item in the driver. The configurable item is set to the contents of `IOCount`. The previous contents of the configurable item are returned in `iberr`. If `id` is a GPIB interface board descriptor, `controlVar` takes on the values shown in Table 1-5. If `id` is a device descriptor, `controlVar` has the values shown in Table 1-6.

Board Function Example

```
gpibBlock paramBlk;

pb = &paramBlk;
pb->id = 0;
pb->controlVar = 10;
pb->IOCount = 0;           /* Release System Control */
Control (refNum,71,(Ptr)&paramBlkPtr);
```


© National Instruments Corp.

IBDEV IBDEV

Purpose

Open an unused device and return the unit descriptor.

Control Number

32

Parameter Block Fields

```
short    id      ->  | board index
short    ud      <-  | device descriptor
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is an index from 0 to [(number of boards) - 1] of the board with which the device descriptor must be associated. `ud` is the returned device descriptor that is used in subsequent calls to reference the particular device.

Use this function to return a device descriptor when you want to configure a device dynamically or when the user-configured name or parameters of the device is unknown. Use this function in place of `ibFIND`. To configure the device, call the functions `ibPAD`, `ibSAD`, `ibTMO`, `ibEOS`, and `ibEOT`.

`ibDEV` returns a device descriptor of the first unopened user-configured device that the function finds. The device descriptor is attached to the board if necessary. The function performs the equivalent of `ibONL` to open the specified device and to initialize software parameters to their default user-configuration settings.

To make your source code easier to read, assign the device descriptor to a variable name that suggests the actual name of the device.

The device descriptor of the driver can remain open across invocations of an application. Therefore, return the device descriptor to the pool of unopened descriptors by calling `ibONL` with `controlVar` = 0. If you do not do this, the device descriptor will not be available on the next call to `ibDEV`. The maximum number of device descriptors for each release of the driver is given in the `Read Me` file on the distribution disk.

The device descriptor is valid until `ibONL` is used to place that device offline.

If the `ibDEV` call fails, a negative number is returned in place of the device descriptor. The most probable reason for a failure is that the `board_index` argument passed into `ibDEV` does not correspond with a GPIB board in the computer.

IBDEV

IBDEV (Continued)

Do not call ibFIND to open a device after using ibDEV because ibFIND can open online devices. Doing so changes the ibDEV dynamic configuration of the device to the default configuration (set by the configuration program). If the ERR bit of ibsta is set, the error code in iberr identifies the problem so that your application can take appropriate action.

The error code can be one of the following.

- EDVR — No GPIB driver is loaded (no NI-488 INIT in the System Folder or no GPIB board installed), or too many devices are open by previous calls to ibDEV and ibFIND. The Macintosh OS error code is returned in ibcnt.
- EARG — If ibDEV returns a negative number, the bus_index is out of range; otherwise, one of the other arguments is out of range.
- ENEB — Nonexistent GPIB board in the slot associated with bus_index.
- ECAP — The driver has no capability for this call. Ask the user to install the NI-488 INIT.

Board Function Example

Assign a device descriptor for an instrument to the variable ud.

```
paramBlkPtr = &gpibBlk;          /*  Set up like ibfind*/
gpibBlk.statusBlk = &statusBlk;
gpibBlk.id = 3;
/*   Board index                 */
osErr = Control (refNum,ibDEV,(Ptr)&paramBlkPtr);
if (osErr){
/*   Handle OSError here.        */
}
ud = gpibBlk.id;
```

IBDMA IBDMA

Purpose

Enable or disable DMA.

Control Number 16

Parameter Block Fields

```
short    id           ->  | board id
short    controlVar   ->  | 0 or 1
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is an interface board. If `controlVar` is non-zero, DMA transfers between the NI-488.2 driver and memory are used for read and write operations. If `controlVar` is zero, programmed I/O is used instead of DMA I/O.

The assignment made by this function remains in effect until `ibDMA` is called again, the `ibONL` or `ibFIND` function is called, or the system is restarted.

When `ibDMA` is called and an error does not occur, the previous value of `controlVar` is stored in `iberr`.

On systems without DMA capability, calling this function when `controlVar` is zero has no effect, and calling `controlVar` with a non-zero value results in an EDMA error.

Board Function Examples

1. Enable DMA transfers.

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND */
                                 /*  control call             */
paramBlk->controlVar = 1;        /*  enable DMA               */
osErr = Control (refNum,ibDMA,&paramBlk);
/*  ibDMA = 16 */
```

2. Disable DMA and use programmed I/O exclusively.

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND
*/
                                 /*  control call            */
paramBlk -> controlVar = 0;      /*  disable DMA             */
osErr = Control (refNum,ibDMA,&paramBlk);
/*  ibDMA = 16  */
```

IBEOS

IBEOS

Purpose

Change or disable EOS termination mode.

Control Number

20

Parameter Block Fields

```
short    id           ->  | board or device id
short    controlVar   ->  | 0, or EOS char
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is a device or an interface board. `controlVar` is the EOS character and the data transfer termination method according to Table 3-2. `ibEOS` is needed only to alter the value from its configuration setting.

The assignment made by this function remains in effect until `ibEOS` is called again, the `ibONL` or `ibFIND` function is called, or the system is restarted.

If `ibEOS` is called and an error does not occur, the previous value of `controlVar` is stored in `iberr`.

Table 3-2. Data Transfer Termination Method

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

Methods A and C determine how read operations terminate. If only Method A is chosen, reads terminate when the low seven bits of the byte that is read match the low seven bits of the EOS character. If Methods A and C are chosen, a full 8-bit comparison is used.

IBEOS IBEOS

(Continued)

Methods B and C determine when write operations send the END message. If only Method B is chosen, the END message is sent automatically with the EOS byte when the low seven bits of that byte match the low seven bits of the EOS character. If Methods B and C are chosen, a full 8-bit comparison is used.

Note: ***Defining an EOS byte for a device or board does not cause the driver to automatically send that byte when performing an*** `ibWRT`***. To send the EOS byte, your application program must include the EOS byte in the data string it defines***.

Device IBEOS Function

If `id` is a device, the options coded in `controlVar` are used for all device reads and writes in which that device is specified.

Board IBEOS Function

If `id` is a board, the options coded in `controlVar` become associated with all board reads and writes.

Refer also to *IBEOT*.

Device Function Example

Terminate read on linefeed character.

```
paramBlk->id = devID;              /*  devID returned by        */
                                   /*  ibFIND control call      */
paramBlk->controlVar = REOS | 0xA;   /*terminate read on LF   */
                                   /*  (8 bit compare)          */
osErr = Control (refNum,ibEOS,&paramBlk);
/*       ibEOS = 20   */
/*   The END bit on ibsta is set if the read terminated on    */
/*   the EOS character.  The value of ibcnt shows the number  */
/*   of bytes received.                                       */
```

IBEOS IBEOS
(Continued)

Board Function Examples

1. Program the interface board `brd0` to terminate a read on detection of the linefeed character ('\n' == hex 0A) that is expected to be received within 512 bytes.

```
paramBlk->id = brd0                  /*  brd0 returned by ibFIND    */
                                     /*  control call               */
paramBlk->controlVar = REOS | 'n';
osErr = Control (refNum,ibEOS,&paramBlk);
/*   assume board has been addressed; do board read                 */
paramBlk->IOBufPtr = buf;
paramBlk->IOCount = 512;
osErr = Control (refNum,ibRD,&paramBlk);
/*  ibRD = 6     */
/*  The END bit on ibsta is set if the read terminated on the   */
/*  EOS character.  The value of ibcnt shows the number of bytes*/
/*   received.                                                   */
```

2. Program the interface board `brd0` to terminate read operations on the 8-bit value hex 82 rather than the 7-bit character hex 0A.

```
paramBlk->id = brd0;                 /*  brd0 returned by ibFIND    */
                                     /*  control call               */
paramBlk->controlVar = REOS | BIN | 0x82;
osErr = Control (refNum,ibEOS,&paramBlk);
/*   Assume board has been addressed; do board read                 */
paramBlk->IOBufPtr = buf;
paramBlk->IOCount = 512;
osErr = Control (refNum,ibRD,&paramblk)
/*   ibrd = 6                        */
/*  The END bit on ibsta is set if the read terminated on the   */
/*  EOS character.  The value of ibcnt shows the number of bytes*/
/*   received.                                                   */
```

3. Disable read termination on receiving the EOS character for operations involving the interface board `brd0`.

```
paramBlk->id = brd0;                 /*  brd0 returned by ibFIND    */
                                     /*  control call               */
paramBlk->controlVar = 0;            /*  disable EOS modes          */
osErr = Control (refNum,ibEOS,&paramBlk);
```

 © National Instruments Corp.

IBEOS IBEOS

(Continued)

4. Send END when the linefeed character is written for operations involving the interface board `brd0`.

```
paramBlk->id = brdID               /*  brdID returned by ibFIND   */
                                   /*  control call               */
paramBlk->controlVar = XEOS | '\n';    /*EOS info for ibeos     */
osErr = Control (refNum,ibEOS,&paramBlk);
```

5. Send END with linefeeds and terminate reads on linefeeds for operations involving the interface board `brd0`.

```
paramBlk->id = brdID               /*  brdID returned by ibFIND
*/
                                   /*  control call            */
paramBlk->controlVar = REOS | XEOS | 0x0A;
/*  EOS info for  */
/*  ibeos         */
osErr = Control (refNum,ibEOS,&paramBlk);
```

IBEOT IBEOT

Purpose

Enable or disable END termination message on write operations.

Control Number

17

Parameter Block Fields

```
short    id            ->  | board or device id
short    controlVar    ->  | 0 or 1
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is a device or an interface board. If `controlVar` is non-zero, the END message is sent automatically with the last byte of each write operation. If `controlVar` is zero, END is not sent. `ibEOT` is needed only to alter the value from its configuration setting.

The END message is sent when the GPIB EOI signal is asserted during a data transfer and it is used to identify the last byte of a data string without having to use an EOS character. `ibEOT` is used primarily to send variable length binary data.

The assignment made by this function remains in effect until `ibEOT` is called again, the `ibONL` or `ibFIND` function is called, or the system is restarted.

If `ibEOT` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Device IBEOT Function

If `id` is a device, the END termination message method that is selected is used on all device I/O write operations to that device.

Board IBEOT Function

If `id` is an interface board, the method that is selected is used on all board I/O write operations, regardless of which device is written to.

Refer also to *IBEOS*.

IBEOT

IBEOT (Continued)

Device Function Example

Send the END message with the last byte of all subsequent write operations to this device.

```
paramBlk->id = devID;             /*  devID returned by ibFIND   */
                                  /*  control call               */
paramBlk->controlVar = 1;         /*  send END with last byte    */
osErr = Control (refNum,ibEOT,&paramBlk);
/*  ibEOT = 17   */
```

Board Function Examples

1. Stop sending END with the last byte for calls directed to the interface board `brd0`.

```
paramBlk->id = brdID;             /*  brdID returned by ibFIND   */
                                  /*  control call               */
paramBlk->controlVar = 0;         /*  disable setting of EOI     */
osErr = Control (refNum,ibEOT,&paramBlk);
/*  ibEOT = 17    */
```

2. Send the END message with the last byte of all subsequent write operations directed to the interface board `brd0`.

```
paramBlk->id = brdID;             /*  brdID returned by ibFIND   */
                                  /*  control call               */
paramBlk->controlVar = 1;         /*  send END with last byte    */
osErr = Control (refNum,ibEOT,&paramBlk);
/*  ibEOT = 17    */
```

IBFIND IBFIND

Purpose

Open
device and return the unit descriptor associated with the given name.

Control Number

25

Parameter Block Fields

Ptr	IOBufPtr	->	pointer to board or device name
short	id	<-	board or device id
short	ibsta	<-	status variables
short	iberr	<-	returned
long	ibcnt	<-	

`IOBufPtr` is a string containing a default or configured device or board name. `id` is the returned unit descriptor, which is used in subsequent calls to reference the device or board.

`ibFIND` opens individual boards or devices, and returns a board or device `id` to be used on all subsequent calls for that board or device. Calling `ibFIND` associates a variable name in the application program with a particular default or configured device or board name. The name used in the `bdname` argument must match exactly the default or configured device or board name. The number returned must be assigned to a variable that is used in all references to that device or board in GPIB function calls.

`ibFIND` performs the equivalent of `ibONL` to open the specified device or board and to initialize software parameters to their default configuration settings. Select a variable name that suggests the actual name of the device or board to simplify programming.

The unit descriptor is valid until `ibONL` is used to place that device or interface board offline.

If the `ibFIND` call fails, a negative number is returned rather than the unit descriptor. The most probable reason for a failure is that the string argument passed into `ibFIND` does not exactly match the default or configured device or board name.

The `ibFIND` function returns status in `ibsta` and an error code in `iberr`. If the ERR bit of `ibsta` is set, the error code in `iberr` identifies the problem so that your application can take appropriate action. The error code can be one of the following.

- EDVR No GPIB driver is loaded (no `NI-488 INIT` in the `System Folder` or no GPIB board installed). The Macintosh OS error code is returned in `ibcnt`.

IBFIND IBFIND
(Continued)

- EARG The board or device name is too long, or no such name exists.
- ENEB Nonexistent GPIB board associated with board or device name.

Device Function Example

Get device descriptor for device `dev1` and put device online.

```
paramBlk = &gpibBlk;
paramBlk.statusBlk = &statusBlk;
s[0] = 'd';
s[1] = 'e';
s[2] = 'v';
s[3] = '1';
s[4] = '\0';                    /*  NULL terminated string     */
paramBlk->IOBufPtr = s;
osErr = Control (refNum,ibFIND,&paramBlk);
/*  ibFind = 25  */
devID = paramBlk->id;           /*  devID used for all         */
                                /*  subsequent device calls to */
                                /*  "dev1"                     */
```

Board Function Example

Get board descriptor for board `gpib1` and put board online.

```
paramBlk = &gpibBlk;
paramBlk -> idstatusBlk = &statusBlk;
s[0] = 'g';
s[1] = 'p';
s[2] = 'i;
s[3] = 'b';
s[4] = '1';                        /*  NULL terminated string  */
s[5] = '\0';
paramBlk -> idIOBufPtr = s;
osErr = Control (refNum,ibFIND,&paramBlk);
/*  ibFIND = 25  */
brdID = paramBlk->id;
```

IBGTS IBGTS

Purpose

Go from Active Controller to Standby.

Control Number

2

Parameter Block Fields

```
short    id           ->  | board id
short    controlVar   ->  | 0 or 1
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is an interface board (value returned from `ibFIND`). If `controlVar` is non-zero, the GPIB interface shadow handshakes the data transfer as an Acceptor, and when the END message is detected, the GPIB interface enters a Not Ready For Data (NRFD) handshake holdoff state on the GPIB. If `controlVar` is zero, no shadow handshake or holdoff occurs.

The `ibGTS` function causes the GPIB interface to go to the Controller Standby state and to unassert the ATN signal if it initially is the Active Controller. `ibGTS` permits GPIB devices to transfer data without involving the GPIB interface in the transfer.

If the shadow handshake option is activated, the GPIB interface participates in data handshake as an Acceptor without actually reading the data. It monitors the transfers for the END (EOI or EOS character) message and holds off subsequent transfers. This permits the GPIB interface to take control synchronously on a subsequent operation, such as `ibCMD` or `ibRPP`.

Before performing an `ibGTS` with shadow handshake, call the `ibEOS` function to establish the proper EOS character and to disable EOS detection if the EOS character in use by the Talker is not known.

The ECIC error results if the GPIB interface is not CIC.

Refer also to *IBCAC*.

IBGTS IBGTS
(Continued)

Board Function Example

Turn the ATN line off with the `ibGTS` function after unaddressing all Listeners with the Unlisten (ASCII _) command, addressing a Talker at hex 46 (ASCII F), and addressing a Listener at hex 31 (ASCII 1) to permit the Talker to send data messages.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND    */
                                /*  control call                */
paramBlk->controlVar = 1;       /*  take control synchronously  */
osErr = Control (refNum,ibGTS,&paramBlk);
/*  ibGTS = 2   */
```

IBIST IBIST

Purpose

Set or clear the individual status bit for parallel polls.

Control Number

3

Parameter Block Fields

```
short    id             ->  | board id
short    controlVar     ->  | 0 or 1
short    ibsta          <-  | status variables
short    iberr          <-  | returned
long     ibcnt          <-  |
```

`id` is an interface board. If `controlVar` is non-zero, the individual status bit is set. If `controlVar` is zero, the bit is cleared.

The `ibIST` function is used when the GPIB interface participates in a parallel poll that is conducted by another device that is the Active Controller. The Active Controller conducts a parallel poll by asserting the EOI signal to send the Identify (IDY) message. While this message is active, each device configured to participate in the poll responds by asserting a predetermined GPIB data line either TRUE or FALSE, depending on the value of its local `ist` bit. The GPIB interface, for example, can be assigned to drive the DIO3 data line TRUE if `ist` = 1, and FALSE if `ist` = 0; conversely, it can be assigned to drive DIO3 TRUE if `ist` = 0, and FALSE if `ist` = 1.

The relationship between the value of `ist`, the line that is driven, and the sense at which the line is driven is determined by the Parallel Poll Enable (PPE) message in effect for each device. The GPIB interface can receive this message either locally, via the `ibPPC` function, or remotely, via a command from the Active Controller. Once the PPE message is executed, the `ibIST` function changes the sense at which the line is driven during the parallel poll, thereby allowing the GPIB interface to convey a one-bit, device-dependent message to the Controller.

When `ibIST` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Refer also to *IBPPC*.

IBIST

IBIST (Continued)

Board Function Examples

1. Set the individual status bit.

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND   */
                                 /*  control call               */
paramBlk->controlVar = 1;        /*  ist bit set                */
osErr = Control (refNum,ibIST,&paramBlk);
/*  ibIST = 3    */
```

2. Clear the individual status bit.

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND   */
                                 /*  control call               */
paramBlk->controlVar = 0;        /*  ist bit cleared            */
osErr = Control (refNum,ibIST,&paramBlk);
/*  ibIST = 3   */
```

IBLINES IBLINES

Purpose

Read the state of the GPIB handshake and control lines.

Control Number

42

Parameter Block Fields

```
short    id          ->  | a bus or a device ud
Ptr      IOBufPtr    ->  | address for Bus Lines
                         | response word
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

If `id` is a device, the access bus of the device is used. `IOBufPtr` is a pointer to an integer in which the state of the bus lines is returned. The upper byte of the response contains the eight bus lines whose bit positions are defined below. The lower byte of the response is a validity mask, that is, NRFD is set if
`(gpib_lines&BUS_NRFD)&&(gpib_lines& (BUS_NRFD>>8))`.

If `gpib_lines&(BUS_NRFD>>8)` is FALSE, the hardware has no capability for returning the state of NRFD.

Bus Lines Bit Positions

BUS_EOI	hex 8000	Occurs briefly during last byte of data.
BUS_ATN	hex 4000	Controller is asserting Attention.
BUS_SRQ	hex 2000	Device is requesting service.
BUS_REN	hex 1000	SC is putting devices in remote state.
BUS_IFC	hex 0800	Occurs briefly during interface clear.
BUS_NRFD	hex 0400	Handshake: Not Ready for Data.
BUS_NDAC	hex 0200	Handshake: Not Data Accepted.
BUS_DAV	hex 0100	Handshake: Data Available.

 © National Instruments Corp.

IBLINES IBLINES

(Continued)

Device Function Example

Obtain the GPIB lines response (`gpib_lines`) from the bus associated with the device `scanner` and test for NRFD asserted.

```
paramBlk->id = scanner;           /*  scanner returned by ibFIND  */
                                  /*  control call                */
paramBlk->IOBufPtr = &gpib_lines; /* spb is char storage          */
                                  /*  for response byte           */
osErr = Control (refNum,ibLINES,&paramBlk);
/*  ibLINES = 42 */
if (gpib_lines&BUS_NRFD)&&(gpib_lines&(BUS_NRFD>>8))
/*   The application would then take action appropriate to   */
/*   NRFD asserted.  */
```

IBLLO IBLLO

Purpose

Place devices in Local Lockout state.

Control Number

15

Parameter Block Fields

```
short     id       ->  | bus or device ud
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a bus or a device.

The `ibLLO` function asserts REN and sends the message LLO, which places devices in the Local Lockout state when they are addressed to listen. This usually inhibits recognition of front panel input.

`ibLLO` sends the following commands.

- REN asserted
- Local Lockout (LLO)

Board Function Example

Place devices on the bus `gpib1` in the Local Lockout state.

```
paramBlk->id = gpib1;            /*  gpib1 returned by ibFIND   */
                                 /*  control call               */
osErr = Control (refNum,ibLLO,&paramBlk);                    /*
ibLLO = 15           */
/*   Place any device on gpib1 in Local Lockout state by        */
/*   addressing it to listen with ibCMD.                         */
```

Device Function Example

Put the device `analyz` in the Local Lockout state.

```
paramBlk->id = analyz;           /*  analyz returned by ibFIND  */
                                 /*  control call               */
osErr = Control (refNum,ibLLO,&paramBlk);
/*  ibLLO = 15    */
/*   Place analyz in Local Lockout state by sending it any      */
/*   message with ibWRT.                                        */
```

IBLN IBLN

Purpose

Check for presence of a device on the bus.

Control Number

31

Parameter Block Fields

```
short   id          ->  | board index
short   controlVar  ->  | primary address
long    IOCount     ->  | secondary address
Ptr     IOBufPtr    ->  | pointer to short listen
flag                    |
short   id          ->  | device id
short   ibsta       <-  | status variables
short   iberr       <-  | returned
long    ibcnt       <-  |
```

`id` is a device or an interface board. The field `controlVar` (the primary GPIB device address) should contain a number in the range 0 through 30. The field `IOCount` (the secondary GPIB device address) should contain a number in the range hex 60 through 7E.

The function `ibLN` returns a non-zero value in the variable `listen` if a Listener is at the GPIB address.

The `sad` parameter can be one of the constants `NO_SAD` or `ALL_SAD` in addition to any secondary address. You can test for a Listener using only GPIB primary addressing by setting `sad` to `NO_SAD`. In this case, `ibLN` sends only the primary address. Because there are 961 secondary addresses and `ibLN` waits for NDAC to settle for up to 10 msec, `ibLN` can scan the secondary address space to reduce the total delay of a find-all-Listeners algorithm. `ibLN` can test all secondary addresses associated with a single primary address (a total of 31 device addresses) when you set `sad` to `ALL_SAD`. In this case, `ibLN` sends the primary address and all secondary addresses before waiting for NDAC to settle. If the listen flag is TRUE, you need to search only the 31 secondary addresses associated with a single primary address to locate the Listener. The two constants are defined in the file `decl.h`.

The `ibLN` function addresses any device to listen at the given GPIB address specified by `pad` and `sad`. The board is placed in Standby Controller state by unasserting ATN, and returns to the listen variable state of NDAC, which is asserted if a device is listening.

If `id` is a device, `ibLN` tests for a Listener on the board associated with `ud`.

Refer also to *IBFIND* and *IBDEV*.

IBLN

IBLN
(Continued)

Device and Board Function Example

Test for a GPIB Listener at primary address 2 and secondary address hex 60.

```
short listen;               /*  listen flag declared as short   */
paramBlkPtr->id = ud;
paramBlkPtr->controlVar = 2;
paramBlkPtr->IOCount = 0x60;
paramBlkPtr->IOBufPtr = (Ptr)&listen;
osErr = Control (refNum,ibLN,(Ptr)&paramBlkPtr);
if (osErr){
   /*   Handle OSError here. */
}
if( !listen){
   /*   Error: No device is at this address.                */
}
```

IBLOC

Purpose

Go to Local.

Control Number

4

Parameter Block Fields

```
short    id      ->  | board or device id
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is a device or an interface board.

Unless the Remote Enable line has been unasserted with the `ibSRE` function, all device 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.

Device IBLOC Function

`ibLOC` places the indicated device in local mode by calling `ibCMD` to send the following command sequence.

- Unlisten (UNL)
- Listen address of the device
- Secondary address of the device, if necessary
- Go To Local (GTL)

Other command bytes can be sent as necessary.

On exit, all devices are unaddressed.

Board IBLOC Function

If `id` is an interface board and it is not locked in remote state, the board is placed in a local state by sending the local message Return To Local (`RTL`). (The LOK bit of the status word indicates whether the board is in a Lockout state.) The `ibLOC` function simulates a front panel RTL switch when the computer is used as an instrument.

IBLOC

IBLOC (Continued)

Device Function Example

Return the device `dvm` to local state.

```
paramBlk->id = devID;          /*  devID returned by ibFIND   */
                               /*  control call               */
osErr = Control (refNum,ibLOC,&paramBlk);
/*  ibLOC = 4         */
```

Board Function Example

Return the interface board `brd0` to local state.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND   */
                               /*  control call               */
osErr = Control (refNum,ibLOC,&paramBlk);
/*  ibLOC = 4         */
```

IBLOCK IBLOCK

Purpose

Lock access to a GPIB-ENET board or device.

Control Number

79

Parameter Block Fields

```
short    id      ->  | board or device id
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is a device or an interface board. `ibLOCK` is used to obtain exclusive access to a GPIB -ENET interface.

Board IBLOCK Function

The `ibLOCK` function blocks other processes from accessing the interface designated by `id` while the lock is in effect. The interface is released via an `ibUNLOCK` function call made with the same board descriptor.

Device IBLOCK Function

The `ibLOCK` function blocks other processes from accessing the device designated by `id` while the lock is in effect. The device lock is released via an `ibUNLOCK` function call made with the same device descriptor.

Recommended Usage

In general, the `ibLOCK` function should be used to gain critical access to a GPIB-ENET board or device when multiple processes might be accessing the same board or device. While locked, the software guarantees that subsequent calls made from the privleged board or device are completed without interruption.

IBLOCK IBLOCK

(Continued)

The ELCK error occurs if the GPIB-ENET board or device being locked is already locked by another process.

Refer also to *IBUNLOCK*.

Board/Device Function Example

Lock the board or device connection specified by `brdID`.

```
paramBlk->id = brdID; /* brdID returned by ibFIND              */
                      /* control call                          */
osErr = Control (refNum,ibSIC,&paramBlk);   /*FC_ibLOCK = 79  */
```

IBONL IBONL

Purpose

Place the device or interface board online or offline.

Control Number

5

Parameter Block Fields

```
short    id           ->  | board or device id
short    controlVar   ->  | 0 or 1
short    ibsta        <-  | status variable
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is a device or an interface board. If `controlVar` is non-zero, the device or interface board is enabled for operation (online). If `controlVar` is zero, it is held in a reset, disabled mode (offline).

Taking a device or interface board offline can be thought of as disconnecting its GPIB cable from the other devices.

Calling `ibONL` with `controlVar` non-zero when the device or interface board is already online simply restores all configuration settings to their defaults.

Device Function Examples

1. Disable the device `plotter`.

```
paramBlk->id = devID;             /*  devID returned by ibFIND   */
                                  /*  control call               */
paramBlk->controlVar = 0;         /*  disable device             */
osErr = Control (refNum,ibONL,&paramBlk);
/*  ibONL = 5         */
```

IBONL IBONL

(Continued)

2. Enable the device `plotter` after taking it offline temporarily.

```
paramBlk = &gpibBlk;
paramBlk.statusBlk = &statusBlk;
s[0] = 'P';
s[1] = 'L';
s[2] = 'O';
s[3] = 'T';
s[4] = 'T';
s[5] = 'E';
s[6] = 'R';
s[7] = '\0';                        /*  NULL terminated string */
paramBlk -> idIOBufPtr = s;
osErr = Control (refNum, ibFIND, &paramBlk);
/*  ibFIND = 25  */
plotter = paramBlk->id;
/*   ibonl with v non-zero is automatically performed as     */
/*   part of ibfind.                                         */
```

3. Restore the configuration settings of the device `plotter` to their defaults.

```
paramBlk->id = plotter;      /*  plotter returned by ibFIND   */
                             /*  control call                 */
paramBlk->controlVar = 1;
osErr = Control (refNum, ibONL, &paramBlk);
/*  ibONL = 5    */
```

Board Function Examples

1. Disable the interface board with descriptor `brdID`.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 1;
osErr = Control (refNum,ibONL,&paramBlk);
/*  ibONL = 5    */
```

IBONL IBONL (Continued)

2. Enable the interface board `gpib1` after taking it offline temporarily.

```
paramBlk = &gpibBlk;
paramBlk -> idstatusBlk = &statusBlk;
s[0] = 'g';
s[1] = 'p';
s[2] = 'i';
s[3] = 'b';
s[4] = '\0';                    /*  NULL terminated string    */
paramBlk -> idIOBufPtr = s;
osErr = Control (refNum, ibFIND, &paramBlk);
/*  ibFIND = 25   */
brdID = paramBlk->id;
/*   ibonl with v non-zero is automatically performed as part*/
/*  of ibfind.   */
```

3. Restore the configuration settings of the interface board to their defaults.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 1;
osErr = Control (refNum,ibONL,&paramBlk);
/*  ibONL = 5    */
```

IBPAD IBPAD

Purpose

Change primary address.

Control Number

18

Parameter Block Fields

```
short    id            ->  | board or device id
short    controlVar    ->  | new primary address
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is a device or an interface board. `controlVar` is the primary GPIB address of the device or interface board. `ibPAD` alters the primary address value from its configuration setting.

Only the low five bits of `controlVar` are significant and they must not all have the value 1. Thus, 31 valid GPIB addresses exist, ranging from 0 to 30. An EARG error results if the value of `controlVar` is not in this range.

The assignment made by this function remains in effect until `ibPAD` is called again, the `ibONL` or `ibFIND` function is called, or the system is restarted.

If `ibPAD` is called and an error does not occur, the previous value of `controlVar` is stored in `iberr`.

Device IBPAD Function

If `id` is a device, `ibPAD` determines the talk and listen addresses based on the value of `v` for all I/O directed to that device. A device listen address is formed by adding hex 20 to the primary address; the talk address is formed by adding hex 40 to the primary address. Consequently, a primary address of hex 10 corresponds to a listen address of hex 30 and a talk address of hex 50. The actual GPIB address of any device is set within that device, either with hardware switches or a software program. Refer to the device documentation for instructions.

Board IBPAD Function

If `id` is a board, `ibPAD` programs the interface board to respond to the primary talk and listen address indicated by `controlVar`.

Refer also to *IBSAD* and *IBONL*.

IBPAD IBPAD

(Continued)

Device Function Example

Change the primary GPIB listen and talk address of the device `plotter` from the configuration setting to hex 2A and hex 4A, respectively.

```
paramBlk->id = devID;          /*  devID returned by ibFIND   */
                               /*  control call               */
paramBlk->controlVar = 10;     /*  primary address of 10      */
osErr = Control (refNum,ibPAD,&paramBlk);
/*  ibPAD = 18   */
```

Board Function Example

Change the primary GPIB listen and talk address of the interface board `brd0` from the configuration setting to hex 27 and hex 47, respectively.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND   */
                               /*  control call               */
paramBlk->controlVar = 7;      /*  primary address of 7       */
osErr = Control (refNum,ibPAD,&paramBlk);
/*  ibPAD = 18   */
```

IBPCT IBPCT

Purpose

Pass Control.

Control Number

23

Parameter Block Fields

```
short    id       ->  | device id
short    ibsta    <-  | status variables
short    iberr    <-  | returned
long     ibcnt    <-  |
```

`id` is a device.

The `ibPCT` function passes CIC authority to the specified device from the access board assigned to that device. The board automatically goes to Controller Idle State (CIDS). This function assumes that the device has Controller capability.

`ibPCT` calls the board `ibCMD` function to send the following commands.

- Talk address of the device
- Secondary address of the device, if applicable
- Take Control (TCT)

Other command bytes can be sent as necessary.

Refer to *IBCMD* for additional information.

Device Function Example

Pass control to device `devID`.

```
paramBlk->id = devID;          /*  devID returned by ibFIND   */
                               /*  control call               */
osErr = Control (refNum,ibPCT,&paramBlk);
/*  ibPCT = 23   */
```

IBPPC IBPPC

Purpose

Parallel Poll Configure.

Control Number

9

Parameter Block Fields

```
short    id           ->  | board or device id
short    controlVar   ->  | PPC byte (0 or 96 - 127)
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is a device or an interface board (returned from `ibFIND`). `controlVar` must be a valid parallel poll enable/disable command, or the value zero.

If `ibPPC` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Device IBPPC Function

If `id` is a device, the `ibPPC` function enables or disables the device from responding to parallel polls.

`ibPPC` calls the board `ibCMD` function to send the following commands.

- Listen address of the device
- Secondary address of the device, if applicable
- Parallel Poll Configure (PPC)
- Parallel Poll Enable (PPE) or Disable (PPD)

Other command bytes can be sent if necessary.

Each of the 16 PPE messages specifies the GPIB data line (DIO1 through DIO8) and sense (one or zero) that the device must use when responding to the Identify (IDY) message during a parallel poll. The assigned message is interpreted by the device along with the current value of the individual status bit (`ist`) to determine if the selected line is driven TRUE or FALSE. For example, if PPE = hex 64, DIO5 is driven TRUE if `ist` = 0, and FALSE if `ist` = 1; if PPE = hex 68, DIO1 is driven TRUE if `ist` = 1, and FALSE if `ist` = 0. Any PPD message or zero value cancels the PPE message in effect.

IBPPC

IBPPC (Continued)

The PPE and PPD messages that are sent, and the meaning of a parallel poll response (returned by the function `ibrpp`) are system-dependent protocol matters that you determine.

On exit, all devices are unaddressed.

Board IBPPC Function

If `id` is an interface board, the board itself is programmed to respond to a parallel poll by setting its local poll enable (lpe) message to the value of `controlVar`.

Refer also to *IBCMD* and *IBIST*.

Device Function Examples

1. Configure the device `dvm` to respond to a parallel poll by sending data line DIO5 TRUE (`ist` = 0).

```
paramBlk -> devID = id;         /*  devID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 96;      /*  ppc byte              */
osErr = Control (refNum,ibPPC,&paramBlk);
    /*           ibPPC = 9       */
```

2. Configure the device `dvm` to respond to a parallel poll by sending data line DIO1 TRUE (`ist` = 1).

```
paramBlk->id = devID;           /*  devID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 0x68;    /*  ppc byte              */
osErr = Control (refNum,ibPPC,&paramBlk);
/*  ibPPC = 9    */
```

3. Cancel the parallel poll configuration of the device `dvm`.

```
paramBlk->id = devID;           /*  devID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 0x70;    /*  ppc byte              */
osErr = Control (refNum,ibPPC,&paramBlk);
/*  ibPPC = 9    */
```

IBPPC IBPPC
(Continued)

Board Function Example

Configure the interface board `gpib1` to respond to a parallel poll by sending data line DIO5 TRUE (`ist` = 0).

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND */
                                /*  control call             */
paramBlk->controlVar = 0x64;    /*  ppc byte                 */
osErr = Control (refNum,ibPPC,&paramBlk);
/*  ibPPC = 9    */
```

IBRD IBRD

Purpose

Read data from a device to string.

Control Number

6

Parameter Block Fields

```
short    id          ->  | board or device id
Ptr      IOBufPtr    ->  | buffer pointer
long     IOCount     ->  | byte count
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

`id` is a device or an interface board. `IOBufPtr` identifies the storage buffer for data bytes that are read from the GPIB. `IOCount` is the number of bytes to be read from the GPIB.

The `ibRD` function reads `cnt` bytes of data from a GPIB device. Device and board operations of the function are described below.

Device IBRD Function

If `id` is a device, the following board steps are performed automatically to read from the device.

1. The `ibCMD` function is called to address the device to talk and the access board to listen.
2. The board `ibRD` function is called to read the data from the device, as explained in the following discussion, *Board IBRD Function*.

Other command bytes can be sent as necessary.

When the device `ibRD` function returns, `ibsta` holds the latest device status; `ibcnt` is the actual number of data bytes read from the device; and `iberr` is the first error detected if the ERR bit in `ibsta` is set.

Board IBRD Function

If `id` is an interface board, the `ibRD` function attempts to read from a GPIB device that is assumed to be properly initialized and addressed.

IBRD IBRD
(Continued)

If the access board is CIC, the `ibCMD` function must be called prior to `ibRD` to address a device to talk, and address the board to listen. Otherwise, the device on the GPIB that is the CIC must perform the addressing.

If the access board is Active Controller, the board is first placed in Standby Controller state with ATN off and remains there after the read operation is completed. Otherwise, the read operation begins immediately.

An EADR error results if the board is CIC but has not been addressed to listen with the `ibCMD` function. An EABO error results if the board is not the CIC and is not addressed to listen within the time limit. An EABO error also results if the device that is to talk is not addressed or the operation does not complete within the time limit.

The board `ibRD` operation terminates on any of the following events.

- The allocated buffer becomes full.
- An error is detected.
- The time limit is exceeded.
- An END message is detected.
- An EOS character is detected (if this option is enabled).
- A Device Clear (DCL) or Selected Device Clear (SDC) command is received from another device which is the CIC.

After termination, `ibcnt` contains the number of bytes read. A short count can occur on any event but the first.

Device Function Example

Read 56 bytes of data from the device `tape`.

```
paramBlk -> = devID;          /*  devID returned by ibFIND   */
                              /*  control call               */
paramBlk->IOCount = 56;
paramBlk->IOBufPtr = buf;     /*  large enough for 56 bytes  */
osErr = Control (refNum,ibRD,&paramBlk);
/*  ibRD = 6    */
```

IBRD IBRD
(Continued)

Board Function Examples

1. Read 56 bytes of data from a device at talk address hex 4C (ASCII L), and then unaddress it (the NI-488.2 driver listen address is hex 20 or ASCII blank).

```
paramBlk->id = brdID;
paramBlk->IOCount = 3;
buf[0] = '?';          /*  UNL    */
buf[1] = 'L';          /*  TAD    */
buf[2] = '  ';         /*  MLA    */
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
paramBlk->IOCount = 56;
paramBlk->IOBufPtr = rdBuf;
osErr = Control (refNum,ibRD,&paramBlk);
buf[0] = '_';
buf[1] = '?';
paramblk -> IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
```

2. To terminate the read on an EOS character, see *IBEOS* examples.

IBRDA IBRDA

Purpose

Read data asynchronously to string.

Control Number

37

Parameter Block Fields

```
short   id          ->  | board or device id
Ptr     IOBufPtr    ->  | buffer pointer
long    IOCount     ->  | byte count
short   ibsta       <-  | status variables
short   iberr       <-  | returned
long    ibcnt       <-  |
```

`id` is a device or an interface board. `IOBufPtr` identifies the storage buffer for data bytes that are read from the GPIB. `IOCount` is the number of bytes to be read from the GPIB.

The `ibRDA` function reads `cnt` bytes of data from a GPIB device.

`ibRDA` is used in place of `ibRD` when the application program must perform other functions while processing the GPIB I/O operation. `ibRDA` returns after starting the I/O operation. If the number of bytes to read is small and the bytes are transmitted quickly by the GPIB device, the operation may complete on the initial call. In this case, the CMPL bit is set in `ibsta`. If the operation does not complete on the initial call, monitor the `ibsta` variable after subsequent calls (usually `ibWAIT` calls) to verify that the I/O is complete. When CMPL becomes set in `ibsta`, indicating that the I/O is complete, the number of bytes read is reported in the `ibcnt` variable.

Device IBRDA Function

If `id` is a device, the following board steps are performed automatically to read from the device.

1. The `ibCMD` function is called to address the device to talk, and address the access board to listen.
2. The board `ibRDA` function is called to read the data from the device, as explained in the following discussion, *Board IBRDA Function*.

IBRDA IBRDA (Continued)

Other command bytes can be sent as necessary.

When the device `ibRDA` function returns, `ibsta` holds the latest device status; `iberr` is the first error detected (if the ERR bit in `ibsta` is set).

When the I/O completes and the CMPL bit is set in `ibsta`, the driver automatically unaddresses all devices (if device unaddressing is enabled).

Board IBRDA Function

If `id` is an interface board, the `ibRDA` function attempts to read from a GPIB device that is assumed to be properly initialized and addressed.

If the board is CIC, the `ibCMD` function must be called prior to `ibRDA` to address the device to talk, and address the board to listen. Otherwise, the device on the GPIB that is the CIC must perform the addressing.

If the board is Active Controller, the board is first placed in Standby Controller state with ATN off and remains there after the read operation is completed. Otherwise, the read operation begins immediately.

An EADR error results if the interface board is CIC but has not addressed itself as a Listener with the `ibCMD` function.

`ibRDA` returns immediately even when no error condition exists. When the CMPL bit is set in `ibsta` (indicating that the I/O is complete), `ibcnt` indicates the number of bytes received.

After the `ibRDA` call and the before corresponding CMPL, other GPIB function calls to this device or to any other device with the same access board, or any board calls to the access board itself, return the error EOIP, with the following exceptions.

- `ibSTOP` To cancel the asynchronous I/O
- `ibWAIT` To monitor other GPIB conditions
- `ibONL` To cancel the I/O and reset the interface

IBRDA IBRDA (Continued)

The board ibRDA operation terminates on any of the following events.

- The allocated buffer becomes full.
- An error is detected.
- The time limit is exceeded.
- An END message is detected.
- An EOS character is detected (if this option is enabled).
- A Device Clear (DCL) or Selected Device Clear (SDC) command is received from another device which is the CIC.

After termination, ibcnt contains the number of bytes read. A short count can occur on any event but the first.

Device Function Example

Read 56 bytes of data from the device tape while performing other processing.

```
paramBlk -> = devID;          /* devID returned by ibFIND   */
                              /* control call               */
paramBlk->IOCount = 56;
paramBlk->IOBufPtr = buf;     /* large enough for 56 bytes  */
osErr = Control (refNum,ibRDA,&paramBlk);
/* ibRD = 37    */

do {
/*   Perform other processing here then wait for I/O completion */
/*   or timeout.              */
ibwait (tape,TIMO | CMPL);
/*   If CMPL or ERR is not set, continue processing.*/
} while (!(ibsta & CMPL));
```

IBRDA IBRDA

(Continued)

Board Function Example

Read 56 bytes of data from a device at talk address hex 4C (ASCII L), and then unaddress it (the NI-488.2 driver listen address is hex 20 or ASCII blank).

```
paramBlk->id = brdID;
paramBlk->IOCount = 3;
buf[0] = '?';                           /*   UNL                 */
buf[1] = 'L';                           /*   TAD                 */
buf[2] = '  ';                          /*   MLA                 */
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
paramBlk->IOCount = 56;
paramBlk->IOBufPtr = rdBuf;
osErr = Control (refNum,ibRDA,&paramBlk);
/*   Perform other processing here then wait for I/O completion*/
/*   or timeout.*/

     ibwait (brd0,TIMO | CMPL);

/*   Check ibsta to see what the read terminated on: */
/*   CMPL, END, TIMO, or ERR (not done here). Data is stored */
/*   in rd.*/
/*   Unaddress the Talker and Listener.                      */
buf[0] = '_';
buf[1] = '?';
paramblk -> IOBufPtr = buf;
osErr = Control (refNum, ibCMD, &paramBlk);
```

IBRPP IBRPP

Purpose

Conduct a parallel poll.

Control Number

7

Parameter Block Fields

```
short      id        ->  | board or device id
paramBlk   IOBufPtr  ->  | buffer pointer
short      ibsta     <-  | status variables
short      iberr     <-  | returned
long       ibcnt     <-  |
```

`id` is a device or an interface board. `IOBufPtr` identifies the address where the parallel poll response byte is stored.

Device IBRPP Function

If `id` is a device, all devices on its GPIB are polled in parallel using the access board of that GPIB. This is done by executing the board `ibRPP` function with the appropriate access board specified.

Board IBRPP Function

If `id` is a board, the `ibRPP` function causes the identified board to conduct a parallel poll of previously configured devices by sending the IDY message (ATN and EOI both asserted) and reading the response from the GPIB data lines.

An ECIC error results if the GPIB interface is not CIC. If the GPIB interface is Standby Controller, it takes control and asserts ATN (becomes Active) prior to polling. The GPIB interface remains Active Controller afterward.

In the examples that follow, some of the GPIB commands and addresses are coded as printable ASCII characters. The simplest means of specifying values is to use printable ASCII characters. Refer to Appendix A, *Multiline Interface Messages*, for conversions of hex values to ASCII characters.

Five commands relevant to parallel polls are shown in Table 3-3.

IBRPP

IBRPP (Continued)

Table 3-3. Parallel Poll Commands

Command	Hex Value	Meaning
PPC	05	Parallel Poll Configure
PPU	15	Parallel Poll Unconfigure
PPE	60	Parallel Poll Enable
PPD	70	Parallel Poll Disable
S	08	Parallel Poll Sense Bit

Device Function Example

Remotely configure the device `lcrmtr` to respond positively on DIO3 if its individual status bit is 1, and then parallel poll all configured devices.

```
paramBlk -> = devID;         /*  devID returned by ibFIND     */
                             /*  control call                 */
paramBlk->IOBufPtr = &ppc;   /*  address of byte for ppr value   */
osErr = Control (refNum,ibRPP,&paramBlk);   /*    ibRPP = 7    */
```

Board Function Examples

1. Remotely configure the device at listen address hex 23 to respond positively on DIO3 if its individual status bit is 1, and then parallel poll all configured devices.

```
paramBlk->id = brdID;          /*  LAD                          */
paramBlk->IOCount = 4;
buf[0] = 0x23;                 /*  LAD                          */
buf[1] = PPC;
buf[2] = PPE | S | Z;          /*  send PPR3 if ist = 1         */
buf[3] = UNL;
paramBlk->IOBufPtr = buf
osErr = Control (refNum, ibCMD, &paramBlk);
paramBlk->IOBufPtr = &ppr;
osErr = Control (refNum,ibRPP,&paramBlk);
```

2. Disable and unconfigure all GPIB devices from parallel polling using the PPU command.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND     */
                               /*  control call                 */
paramBlk->IOCount = 1;
buf[0] = 0x15;
paramBlk->IOBufPtr = buf;
osErr = Control (refNum, ibCMD, &paramBlk); /*ibCMD = 1       */
```

IBRSC IBRSC

Purpose

Request or release system control.

Control Number 8

Parameter Block Fields

```
short    id          ->  | board id
short    controlVar  ->  | 0 or 1
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

`id` is an interface board. If `controlVar` is non-zero, functions requiring System Controller capability are subsequently allowed. If `controlVar` is zero, functions requiring System Controller capability are disallowed.

The `ibRSC` function is used to enable or disable the capability of the GPIB interface driver to send the Interface Clear (IFC) and Remote Enable (REN) messages to GPIB devices, using the `ibSIC` and `ibSRE` functions respectively. The interface board must not be System Controller to respond to Interface Clear sent by another Controller.

In most applications, the GPIB interface driver is always the System Controller. In some applications, however, the GPIB interface driver is never the System Controller. In either case, the `ibRSC` function is used only if the computer is not going to be System Controller for the duration of the program execution. While the IEEE 488 standard does not specifically allow schemes in which System Control can be passed dynamically from one device to another, the `ibRSC` function would be used in such a scheme.

If `ibRSC` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Board Function Example

Request to be System Controller if the interface board `brd0` is not currently so designated.

```
paramBlk -> = brdID;             /*  brdID returned by ibFIND  */
                                 /*  control call              */
paramBlk->controlVar = 1;        /*  allow System Controller   */
                                 /*  capability                */
osErr = Control (refNum,ibRSC,&paramBlk); /*ibRSC = 8           */
```

IBRSP IBRSP

Purpose

Return serial poll byte.

Control Number

24

Parameter Block Fields

```
short    id           ->  | device id
Ptr      IOBufPtr     ->  | address for serial poll
                          response byte
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is a device. `IOBufPtr` is the variable in which the poll response is stored.

The `ibRSP` function serial polls one device and obtains its status byte, or obtains a previously stored status byte. If bit 6 (the hex 40 bit) of the response is set, the status response is positive, that is, the device is requesting service. On exit, all devices are unaddressed.

When automatic serial polling is enabled (using the configuration program), the specified device may have been polled previously. If it has been polled and a positive response was obtained, the RQS bit of that device status word is set, and a call to `ibRSP` returns the previously acquired status byte. If the RQS bit of the status word is not set when `ibRSP` is called, the device is serial polled.

When a poll is conducted, the following sequence of events occurs.

1. `ibCMD` sends the following commands.
   - Talk address of the device
   - Secondary address of the device, if applicable
   - Listen address of the access board
   - Secondary address of the access board, if applicable
   - Serial Poll Enable (SPE)

   Other command bytes can be sent as necessary.

2. `ibRD` reads one response byte from the device and stores it in `spr`.

IBRSP

IBRSP (Continued)

3. `ibCMD` sends the following commands.

    - Untalk (UNT) and Unlisten (UNL)
    - Serial Poll Disable (SPD)

The interpretation of the response in `IOBufPtr`, except the RQS bit, is device-specific. For example, the polled device might set a particular bit in the response byte to indicate that it has data to transfer, and another bit to indicate a need for reprogramming. Consult your device documentation for an interpretation of the response byte.

Refer to *IBCMD* and *IBRD* for additional information.

Device Function Example

Obtain the serial poll response (`spr`) byte from the device `tape`.

```
paramBlk->id = devID;           /*  devID returned by ibFIND  */
                                /*  control call              */
paramBlk->IOBufPtr = &spb;      /*  spb is char storage for   */
                                /*  response byte             */
osErr = Control (refNum,ibRSP,&paramBlk);
/*  ibRSP = 24   */
```

IBRSV IBRSV

Purpose

Request service and/or set or change serial poll status byte.

Control Number

11

Parameter Block Fields

```
short    id           ->  | device id
short    controlVar   ->  | 0 or serial poll status
byte
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is an interface board. `controlVar` is the response or status byte that the GPIB interface driver provides when serial polled by another device that is the GPIB CIC. If bit 6 (the hex 40 bit) is set, the GPIB interface driver also requests service from the Controller by asserting the GPIB SRQ line.

The `ibRSV` function requests service from the Controller using the Service Request (SRQ) signal, and provides a system-dependent status byte when the Controller serial polls the GPIB interface driver.

If `ibRSV` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Board Function Examples

1. Set the serial poll status byte to hex 41, which simultaneously requests service from an external CIC.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND */
                                /*  control call             */
paramBlk->controlVar = 0x41;    /*  set bits 6 & 0 & GPIB SRQ*/
                                /*  line                     */
osErr = Control (refNum,ibRSV,&paramBlk); /*ibRSV = 11       */
```

IBRSV IBRSV

(Continued)

2. Change the status byte as in Example 1, without requesting service.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 1;       /*  set bit 0                 */
osErr = Control (refNum, ibRSV, &paramBlk); /*ibRSV = 11      */
```

IBSAD IBSAD

Purpose

Change or disable secondary address.

Control Number

19

Parameter Block Fields

```
short    id           ->  | board or device id
short    controlVar   ->  | new secondary address
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is a device or an interface board. If `controlVar` is a number between 96 and 127, that number becomes the secondary GPIB address of the device or interface board. If `controlVar` is 128 or 0, secondary addressing is disabled. `ibSAD` is used only to alter the value from its configuration setting.

The assignment made by this function remains in effect until `ibSAD` is called again, the `ibONL` or `ibFIND` function is called, or the system is restarted.

If `ibSAD` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Device IBSAD Function

If `id` is a device, the function enables or disables extended GPIB addressing for the device. If secondary addressing is enabled, `ibSAD` records the secondary GPIB address of that device to be used in subsequent device I/O function calls.

Board IBSAD Function

If `id` is an interface board, the `ibSAD` function enables or disables extended GPIB addressing; when enabled, it assigns the secondary address of the GPIB interface.

Refer also to *IBPAD* and *IBONL*.

IBSAD IBSAD
(Continued)

Device Function Examples

1. Change the secondary GPIB address of the device `plotter` from its current value to hex 6A.

```
paramBlk->id = devID;           /*  devID returned by ibFIND   */
                                /*  control call               */
paramBlk->controlVar = 0x6A;    /*  secondary address of 0x6A  */
osErr = Control (refNum,ibSAD,&paramBlk); /* ibSAD = 19        */
```

2. Disable secondary addressing for the device `dvm`.

```
paramBlk->id = devID;           /*  devID returned by ibFIND   */
                                /*  control call               */
paramBlk->controlVar = 0;      /*  disable secondary addressing*/
osErr = Control (refNum,ibSAD,&paramBlk);   /* ibSAD = 19     */
```

Board Function Examples

1. Change the secondary GPIB address of the interface board `gpib1` from its current value to hex 6A.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND    */
                               /*  control call                */
paramBlk->controlVar = 0x6A;   /*  secondary address of 0x6A   */
osErr = Control (refNum,ibSAD,&paramBlk);   /* ibSAD = 19     */
```

2. Disable secondary addressing for the interface board `brd0`.

```
paramBlk->id = brdID;          /*  brdID returned by ibFIND    */
                               /*  control call                */
paramBlk->controlVar = 0;      /*  disable secondary addressing*/
osErr = Control (refNum,ibSAD,&paramBlk);   /* ibSAD = 19     */
```

IBSIC

Purpose

Send Interface Clear for 100 µsec.

Control Number

12

Parameter Block Fields

```
short    id          ->  | board id
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

`id` is an interface board.

The `ibSIC` function causes the GPIB interface driver to assert the IFC signal for at least 100 µsec if it has System Controller authority. This action initializes the GPIB and makes the interface board CIC and Active Controller with ATN asserted. This function is generally used when a bus fault condition is suspected.

The IFC signal resets only the GPIB interface functions of bus devices and not the internal device functions. Device functions are reset with the Device Clear (DCL) and Selected Device Clear (SDC) commands. To determine the effect of these messages, consult the device documentation.

The ESAC error occurs if the GPIB interface does not have System Controller capability.

Refer also to *IBRSC*.

Board Function Example

Initialize the GPIB and make it CIC and Active Controller at the beginning of a program.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND  */
                                /*  control call              */
osErr = Control (refNum,ibSIC,&paramBlk); /* ibSIC = 12      */
```

IBSRE IBSRE

Purpose

Send or clear Remote Enable line.

Control Number

13

Parameter Block Fields

```
short    id            ->  | board id
short    controlVar    ->  | 0 or 1
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is an interface board. If `controlVar` is non-zero the Remote Enable (REN) signal is asserted. If `controlVar` is zero, the signal is unasserted.

The `ibSRE` function turns the REN signal on and off. REN is used by devices to select between local and remote modes of operation. REN enables the remote mode. A device does not actually enter remote mode until it receives its listen address.

The ESAC error occurs if the GPIB interface driver is not System Controller.

If `ibSRE` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Refer also to *IBRSC*.

Board Function Examples

1. Place the device at listen address hex 23 (ASCII #) in remote mode with local ability to return to local mode.

```
paramBlk->id = brdID;           /*  brdID returned by ibFIND  */
                                /*  control call              */
paramBlk->controlVar = 1;       /*  assert Remote Enable      */
osErr = Control (refNum,ibSRE,&paramBlk); /* ibSRE = 13       */
buf[0] = 0x23;
paramBlk->IOCount = 1;
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
```

IBSRE

IBSRE (Continued)

2. To exclude the local ability of the device to return to local mode, send the Local Lockout command (hex 11), or include it in the command string in Example 1.

```
paramBlk->id = brdID;             /*brdID returned by ibFIND */
                                  /*  control call           */
paramBlk->controlVar = 1;         /*  assert Remote Enable    */
osErr = Control (refNum,ibSRE,&paramBlk);   /* ibSRE = 13 */
paramBlk->id = brdID;             /*brdID returned by ibFIND */
                                  /*  control call           */
paramBlk->IOCount = 1;
buf[0] = 0x11;                    /*  LLO                    */
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);   /* ibCMD = 1     */
buf[0] = 0x23;
paramBlk->IOCount = 1;
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
```

or

```
paramBlk->id = brdID;             /*brdID returned by ibFIND */
                                  /*  control call           */
paramblk -> controlVar = 1;       /*  REN TRUE               */
osErr = Control (refNum,ibCMD,&paramBlk);   /* ibSRE = 13 */
paramBlk->IOCount = 2
buf[0] = '#';                     /*  LAD                    */
buf[1] = 0x11;                    /*  LLO                    */
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);   /* ibCMD = 1     */
```

3. Return all devices to local mode.

```
paramBlk->id = brdID;             /*brdID returned by ibFIND */
                                  /*  control call           */
paramBlk->controlVar = 0;         /*  unassert Remote Enable  */
osErr = Control (refNum,ibSRE,&paramBlk);   /* ibSRE = 13 */
```

IBSTOP IBSTOP

Purpose

Abort asynchronous operation.

Control Number

40

Parameter Block Fields

```
short    id          ->  | board or device id
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

`id` is a device or an interface board. `ibSTOP` terminates asynchronous read, write, and command operations in progress.

If an asynchronous operation is aborted before completion, the ERR bit in the status word is set and an EABO error is returned. No error indication results if the operation successfully completes before `ibSTOP` is called.

Device IBSTOP Function

If `id` is a device, `ibSTOP` attempts to terminate unfinished asynchronous I/O operations to that device.

Board IBSTOP Function

If `id` is a board, `ibSTOP` attempts to terminate unfinished asynchronous I/O operations to that board.

Device Function Example

Stop asynchronous operations associated with the device `rdr`.

```
paramBlk->id = rdr;                 /*  rdr returned by ibFIND  */
                                    /*  control call            */
osErr = Control (refNum,ibSTOP,&paramBlk);  /*ibSTOP = 40      */
```

IBSTOP IBSTOP (Continued)

Board Function Example

Stop asynchronous operations associated with the interface board `brd0`.

```
paramBlk->id = brd;                  /*  brd returned by ibFIND  */
                                     /*  control call            */
osErr = Control (refNum,ibSTOP,&paramBlk);  /* ibSTOP = 40    */
```

IBTMO IBTMO

Purpose

Change or disable time limit.

Control Number

26

Parameter Block Fields

```
short    id            ->  | board or device id
short    controlVar    ->  | timeout value from table
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is a device or an interface board. `controlVar` is a code specifying the time limit as shown in Table 3-4.

IBTMO

Table 3-4. Timeout Code Values

Value of controlVar	Minimum Timeout
0	disabled
1	1 msec
2	1 msec
3	1 msec
4	1 msec
5	1 msec
6	3 msec
7	10 msec
8	30 msec
9	100 msec
10	300 msec
11	1 sec
12	3 sec
13	10 sec
14	30 sec
15	100 sec
16	300 sec
17	1000 sec

IBTMO

IBTMO (Continued)

Note: ***If*** `controlVar` ***is zero, no limit is in effect.***

`ibTMO` is needed only to alter the value from its configuration setting.

The assignment made by this function remains in effect until `ibTMO` is called again, the `ibONL` or `ibFIND` function is called, or the system is restarted.

The `ibTMO` function changes the length of time that the following functions wait for the embedded I/O operation to finish or for the specified event to occur before returning with a timeout indication.

- `ibCMD`
- `ibRD`
- `ibWRT`

The `ibTMO` function also changes the length of time that device functions wait for commands to be accepted. If a device does not accept commands within the time limit, the EBUS error is returned.

If `ibTMO` is called and an error does not occur, the previous value of the Controller is stored in `iberr`.

Device IBTMO Function

If `id` is a device, the new time limit is used in subsequent device functions directed to that device.

Board IBTMO Function

If `id` is a board, the new time limit is used in subsequent board functions directed to that board.

Refer also to *IBWAIT*.

IBTMO IBTMO (Continued)

Device Function Example

Change the time limit to approximately 300 msec for calls directed to the device specified by `devID`.

```
paramBlk->id = devID;              /*  devID returned by ibFIND*/
                                   /*  control call            */
paramBlk->controlVar = 13;         /*  timeout limit of 10 sec */
osErr = Control (refNum,ibTMO,&paramBlk);    /*ibTMO = 26 */
```

Board Function Example

Change the time limit to approximately 20 msec for calls directed to the interface board specified by `brdID`.

```
paramBlk->id = brdID;              /*  brdID returned by ibFIND*/
                                   /*  control call            */
paramBlk->controlVar = 7;          /*  timeout limit of 10 msec*/
osErr = Control (refNum,ibTMO,&paramBlk);   /* ibTMO = 26 */
```

IBTRG IBTRG

Purpose

Trigger selected device.

Control Number 21

Parameter Block Fields

```
short    id      ->  | device id
short    ibsta   <-  | select variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is a device.

The `ibTRG` function addresses and triggers the specified device, then unaddresses all devices on the GPIB.

`ibTRG` calls the board `ibCMD` function to send the following commands.

- Listen address of the device
- Secondary address of the device, if applicable
- Group Execute Trigger (GET)

Other command bytes can be sent as necessary.

Refer to the *IBCMD* function for additional information.

Device Function Example

Trigger the device `analyze.`

```
paramBlk->id = devID;          /*  device id returned by ibFIND*/
                               /*  control call                */
osErr = Control (refNum,ibTRG,&paramBlk);  /*ibTRG = 21       */
```

IBUNLOCK IBUNLOCK

Purpose

Unlock access to a GPIB-ENET board or device.

Control Number

80

Parameter Block Fields

```
short    id      ->  | board or device id
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is a device or an interface board. `ibUNLOCK` is used to release exclusive access to a GPIB-ENET interface.

The `ibUNLOCK` function releases the lock on the board or device connection requested by `ibLOCK`.

Board IBUNLOCK Function

When the `ibLOCK` function has been used to lock access to a board, an `ibUNLOCK` function call made with the same board descriptor unlocks access to the board.

Device IBUNLOCK Function

When the `ibLOCK` function has been used to lock access to a device, an `ibUNLOCK` function call made with the same device descriptor unlocks access to the device.

Recommended Usage

In general, use `ibUNLOCK` to release your lock on a board or device connection. It is recommeded that `ibUNLOCK` be used immediately after critical board or device accesses are made to a locked interface.

Refer also to *IBLOCK*.

IBUNLOCK IBUNLOCK
(Continued)

Board/Device Function Example

Unlock the board or device connection specified by `brdID`.

```
paramBlk->id = brdID; /* brdID returned by ibFIND          */
                      /* control call                      */
osErr = Control (refNum,ibSIC,&paramBlk);/* FC_ibUNLOCK = 80 */
```

IBWAIT IBWAIT

Purpose

Wait for selected event.

Control Number

10

Parameter Block Fields

```
short    id           ->  board or device id
short    controlVar   ->  wait mask
short    ibsta        <-  status variables
short    iberr        <-  returned
long     ibcnt        <-
```

`id` specifies a device or an interface board. `controlVar` is a bit mask with the same bit assignments as the status word, `ibsta.` Each mask bit is set or cleared to wait or not wait, respectively, for the corresponding event to occur.

The `ibWAIT` function is used to monitor the events selected in `mask` and to delay processing until any of these events occurs. The events and bit assignments are shown in Table 3-5.

IBWAIT IBWAIT

(Continued)

Table 3-5. Wait Mask Layout

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

IBWAIT

ibWAIT also updates all conditions of the status word, which can be read in the ibsta variable.

If mask = 0 or mask = hex 8000 (the ERR bit), the function returns immediately.

If the TIMO bit is the value 0, or the time limit is set to the value 0 with the ibTMO function, timeouts are disabled. Disable timeouts only when setting mask = 0 or when you are certain the selected event will occur; otherwise, the processor may wait indefinitely for the event to occur.

Device IBWAIT Function

If id is a device, only the ERR, TIMO, END, RQS, and CMPL bits of the wait mask and status word are applicable. When an ibWAIT function is called for the RQS bit, the access board of the specified device serial polls all devices on its GPIB each time the GPIB SRQ line is asserted. The responses are saved until the specified device returns the status byte that indicates that it is requesting service (bit hex 40 is set in the serial poll response byte). Notice that an ibWAIT on RQS should be used only on those devices that respond to serial polls. If the TIMO bit of the mask is also set, ibWAIT returns if SRQ is not asserted within the timeout period of the device. Serial polls are conducted with the board functions ibCMD and ibRD.

Board IBWAIT Function

If id is a board, all bits of the wait mask and status word are applicable except RQS.

Device Function Example

Wait indefinitely for the device logger to request service.

```
paramBlk->id = devID;           /*  devID returned by ibFIND   */
                                /*  control call               */
paramBlk->controlVar = RQS; /*  mask = 0x800                   */
osErr = Control (refNum,ibWAIT,&paramBlk);     /*ibWAIT=10     */
```

Board Function Examples

1. Wait for a service request or a timeout.

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND */
                                 /*  control call             */
paramBlk->controlVar = 0x4004;   /*  timo and lacs            */
osErr = Control (refNum,ibWAIT,&paramBlk);    /* ibWAIT = 10 */
```

IBWAIT

IBWAIT (Continued)

2. Update the current status for `ibsta`.

```
paramBlk->id = brdID;              /*  brdID returned by ibFIND*/
                                   /*  control call            */
paramBlk->controlVar = 0;          /*  update the current status*/
                                   /*  for ibsta               */
osErr = Control (refNum,ibWAIT,&paramBlk);     /*ibWAIT = 10 */
```

3. Wait indefinitely until control is passed from another CIC.

```
paramBlk->id = brdID;              /*  brdID returned by ibFIND*/
                                   /*  control call            */
paramBlk->controlVar = CIC;        /*  mask = 0x20             */
osErr = Control (refNum,ibWAIT,&paramBlk);     /*ibWAIT = 10 */
```

4. Wait indefinitely until addressed to talk or listen by another CIC.

```
paramBlk->id = brdID;              /*  brdID returned by ibFIND*/
                                   /*  control call            */
paramBlk->controlVar = TACS | LACS;  /*mask = 0x0C             */
osErr = Control (refNum,ibWAIT,&paramBlk);     /*ibWAIT = 10 */
```

IBWRT IBWRT

Purpose

Write data from string.

Control Number

14

Parameter Block Fields

```
short    id          ->  board or device id
Ptr      IOBufPtr    ->  buffer pointer
long     IOCount     ->  byte count
short    ibsta       <-  status variables
short    iberr       <-  returned
long     ibcnt       <-
```

`id` is a device or an interface board. `IOBufPtr` contains the data to be sent over the GPIB. `IOCount` is the number of bytes to be sent over the GPIB.

The `ibWRT`
function writes `IOCount` bytes of data to a GPIB device.

Device IBWRT Function

If `id` is a device, the following board steps are performed automatically to write to the device.

1. The `ibCMD` function is called to address the device to listen, and the access board to talk.
2. The board `ibWRT` function is called to write the data to the device, as explained in the following discussion, *Board IBWRT Function*.
3. The `ibCMD` function is called to unaddress the access board using the Untalk command, and all devices using the Unlisten command.

Other command bytes can be sent as necessary.

When the device `ibWRT` function returns, `ibsta` holds the latest device status; `ibcnt` is the actual number of data bytes written to the device; and `iberr` is the first error detected if the ERR bit in `ibsta` is set.

Board IBWRT Function

If `id` is an interface board, the `ibWRT` function attempts to write to a GPIB device that is assumed to be properly initialized and addressed.

IBWRT IBWRT
(Continued)

If the access board is CIC, the `ibCMD` function must be called prior to `ibWRT` to address the device to listen and the board to talk. Otherwise, the device on the GPIB that is the CIC must perform the addressing.

If the access board is Active Controller, the board is first placed in Standby Controller state with ATN off and remains there after the write operation is completed. Otherwise, the write operation begins immediately.

An EADR error results if the board is CIC but has not been addressed to talk with the `ibcmd` function. An EABO error results if the board is not the CIC and is not addressed to talk within the time limit. An EABO error also results if the operation does not complete within the time limit.

The board `ibWRT` operation terminates on any of the following events.

- All bytes are transferred.
- An error is detected.
- The time limit is exceeded.
- A Device Clear (DCL) or Selected Device Clear (SDC) command is received from another device that is the CIC.

After termination, `ibcnt` contains the number of bytes written. A short count can occur on any event but the first.

Device Function Examples

1. Write two instruction bytes to the device `dvm`.

```
paramBlk->id = devID;            /*  devID returned by ibFIND */
                                 /*  control call             */
paramBlk->IOCount = 2;
buf[0] = 'F';
buf[1] = 'l';
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibWRT,&paramBlk);   /* ibWRT = 14 */
```

IBWRT

IBWRT (Continued)

2. Write five instruction bytes terminated by a carriage return and a linefeed to the device `ptr`.

```
paramBlk->id = devID;            /*  devID returned by ibFIND */
                                 /*  control call             */
paramBlk->IOCount = 7;
paramBlk->IOBufPtr = "IPZX5\r\n";
osErr = Control (refNum,ibWRT,&paramBlk);   /*ibWRT = 14      */
```

Board Function Example

Write two instruction bytes to a device at listen address hex 2F (ASCII /) and then unaddress it (the NI-488.2 driver talk address is hex 40 or ASCII @).

```
paramBlk->id = brdID;            /*  brdID returned by ibFIND */
                                 /*  control call             */
paramBlk->IOCount = 3;
buf[0] = '?';                    /*  UNL                      */
buf[1] = '@';                    /*  MTA                      */
buf[2] = '/';                    /*  LAD                      */
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
paramBlk->IOCount = 2;
buf[0] = 'F';
buf[1] = '3';
osErr = Control (refNum,ibWRT,&paramBlk);
buf[0] = '_';                    /*  UNT                      */
buf[1] = '?';                    /*  UNL                      */
osErr = Control (refNum,ibCMD,&paramBlk);
```

IBWRTA IBWRTA

Purpose

Write data asynchronously from string.

Control Number

38

Parameter Block Fields

```
short    id          ->  | board or device id
Ptr      IOBufPtr    ->  | buffer pointer
long     IOCount     ->  | byte count
short    ibsta       <-  | status variables
short    iberr       <-  | returned
long     ibcnt       <-  |
```

`id` is a device or an interface board. `IOBufPtr` contains the data to be sent over the GPIB. `IOCount` is the number of bytes to be sent over the GPIB.

The `ibWRTA` function writes `IOCount` bytes of data to a GPIB device.

`ibWRTA` is used in place of `ibWRT` when the application program must perform other functions while processing the GPIB I/O operation. `ibWRTA` returns after starting the I/O operation. If the number of bytes to write is small and they are received quickly by the GPIB device, the operation may complete on the initial call. In this case, the CMPL bit will be set in `ibsta`. If the operation does not complete on the initial call, monitor the `ibsta` variable after subsequent calls (usually `ibwait` calls) to verify that the I/O is complete. When CMPL becomes set in `ibsta`, indicating that the I/O is complete, the number of bytes written is reported in the `ibcnt` variable.

Device IBWRTA Function

If `id` is a device, the following board steps are performed automatically to write to the device.

1. The `ibCMD` function is called to address the device to listen, and the assigned interface board to talk.
2. The board `ibWRTA` function is called to write the data to the device, as explained in the following discussion, *Board IBWRTA Function*.

Other command bytes can be sent as necessary.

IBWRTA IBWRTA (Continued)

When the device `ibWRTA` function returns, `ibsta` holds the latest device status. If the ERR bit in `ibsta` is set, `iberr` is the first error detected. When the I/O completes and the CMPL bit is set in `ibsta`, the driver automatically unaddresses all devices (if device unaddressing is enabled).

Board IBWRTA Function

If `id` is an interface board, the `ibWRTA` function attempts to write to a GPIB device that is assumed to be properly initialized and addressed.

If the board is CIC, the `ibCMD` function must be called prior to `ibWRTA` to address the device to listen, and address the board to talk. Otherwise, the device on the GPIB that is the CIC must perform the addressing.

If the board is Active Controller, the board is first placed in Standby Controller state with ATN off and remains there after the write operation is completed. Otherwise, the write operation begins immediately.

An EADR error results if the board is CIC but has not been addressed to talk with the `ibCMD` function. The ENOL error does *not* occur if there are no Listeners.

Note: *To send an EOS character at the end of the data string, place it there explicitly.*

After the `ibWRTA` call and before the corresponding CMPL, other GPIB function calls to this device or to any other device with the same access board (or any board calls to the access board itself) return the error EOIP, with the following exceptions.

- `ibSTOP` To cancel the asynchronous I/O
- `ibWAIT` To monitor other GPIB conditions
- `ibONL` To cancel the I/O and reset the interface

IBWRTA IBWRTA
(Continued)

Device Function Example

Write two instruction bytes to the device dvm.

```
paramBlk->id = devID;            /*  devID returned by ibFIND */
                                 /*  control call             */
paramBlk->IOCount = 10;
paramBlk->IOBufPtr = "F3R1X5P2G0";
osErr = Control (refNum,ibWRTA,&paramBlk);  /*ibWRTA = 38*/

do {
/*   Perform other processing here, then get status           */
ibwait (dvm,0);
} while (!(ibsta & CMPL));       /*  loop until complete      */
```

Board Function Example

Write ten instruction bytes to a device at listen address hex 2F (ASCII /), while testing for a high priority event to occur, and then unaddress it (the GPIB board talk address is hex 40 or ASCII @).

```
paramBlk->id = brdID;             /*  brdID returned by ibFIND
*/
                                  /*  control call            */
paramBlk->IOCount = 3;
buf[0] = '?';                     /*  UNL                     */
buf[1] = '@';                     /*  MTA                     */
buf[2] = '/';                     /*  LAD                     */
paramBlk->IOBufPtr = buf;
osErr = Control (refNum,ibCMD,&paramBlk);
paramBlk->IOCount = 10;
paramBlk->IOBufPtr = "F2R1X5P2G0";
osErr = Control (refNum,ibWRTA,&paramBlk);
do {

/*   Perform other processing here, then get status.          */

    ibwait (brd0,0);
    } while (!(ibsta & CMPL));    /*  loop until complete     */
   buf[0] = '_';                  /*  UNT                     */
   buf[1] = '?';                  /*  UNL                     */
   osErr = Control (refNum,ibCMD,&paramBlk);
```

NI-488 Programming Examples for the Device Manager

These examples illustrate programming steps that you can use to program a representative IEEE 488 instrument from your personal computer using the NI-488.2 driver functions. The applications are written in C.

Example Program—High-Level Device Manager Calls

```
#include "stdio.h"
#include <Devices.h>

typedef struct StatusBlk{
          short     ibsta;
          short     iberr;
          short     ibret;   /* The four GPIB status variables  */
          long      ibcnt;
} StatusBlk;

typedef struct gpibBlock
{
   StatusBlk *statusBlk;
   ushort  id;
   ushort  controlVar;     /* Control variable for some         */
                           /* functions, to indicate nature     */
                           /* of action                      */
   Ptr     IOBufPtr;       /* Pointer to the buffer in          */
                           /* user area, during I/O calls    */
   ulong   OCount;         /* I/O byte count                    */
   ushort* addr ;
   ushort* result;
   ushort  limit ;
   void    (*srqservice) () ;
} gpibBlock;


enum {
   ibCAC, ibCMD, ibGTS, ibIST, ibLOC,
   ibONL, ibRD,  ibRPP, ibRSC, ibPPC,
   ibWAIT,ibRSV, ibSIC, ibSRE, ibWRT,
   ibLLO, ibDMA, ibEOT, ibPAD, ibSAD,
   ibEOS, ibTRG, ibCLR, ibPCT, ibRSP,
   ibFIND,ibTMO, ibPOKE, res1, res2,
   res3,  ibLN   ibDEV,
   PUT_BUS_DATA,  PUT_DEVICE_DATA,
   PUT_BUS_NAMES, PUT_DEVICE_NAMES,
   ibRDA, ibWRTA,ibCMDA,ibSTOP,ibBUS,
   ibLINES, FC_sendcmds, FC_sendsetup, FC_senddatabytes,
   FC_send, FC_receivesetup, FC_rcvrespmsg, FC_receive,
   FC_devclear, FC_sendlist, FC_devclearlist, FC_enablelocal,
   FC_enableremote, FC_sendrwls, FC_sendllo, FC_trigger,
   FC_passcontrol, FC_readstatusbyte, FC_triggerlist,
```

```
FC_ppollconfig, FC_ppollunconfig, FC_ppoll, FC_testsrq,
FC_waitsrq, FC_resetsys, FC_findrqs, FC_allspoll, FC_findlstn,
FC_testsys,ibCONFIG, ibSRQ, ibBNA, FC_ibdiag, FC_ibrdkey,
FC_ibwrtkey, FC_sendifc,FC_ibtrace, FC_ibbus
    };
```

(**Note:** ibLLO, ibSTAT, ibPOKE are not available to the user.)

```
char *brdName = "gpib0";
char *cmd = "@!";
char *cmd2 = " A";
char *wrt = "123456";


main ()

{
            Int osErr;
            short brd, dev1, refnum, brdID;
            gpibBlock gpibBlk, *gpibBlkPtr;
            StatusBlk status;
            char rd[1000];

            osErr = OpenDriver("\p.GPIB Driver", &refnum);
            if (osErr)
                        printf("\n\nOpenDriver error: %d", osErr);

            gpibBlkPtr = &gpibBlk;    /*  setup parameter block     */
            gpibBlkPtr->statusBlk = &status;

            gpibBlkPtr->IOBufPtr = brdName;/* ibFIND                */
            osErr = Control (refnum, ibFIND, &gpibBlkPtr);
            /* brd descriptor                                       */
            /* in gpibBlkPtr->id                                    */

            brdID = gpibBlkPtr->id;
            /* save if changing between devices and                 */
            /* boards                                               */
            printf("\n ibsta: %x iberr: %d ibcnt: %ld", gpibBlkPtr->
                     statusBlk->ibsta,
                        gpibBlkPtr->statusBlk->iberr, gpibBlkPtr-
>statusBlk->ibcnt);

            osErr = Control (refnum, ibSIC, &gpibBlkPtr);/*ibSIC   */
            printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr->
                     statusBlk->ibsta,
                        gpibBlkPtr->statusBlk->iberr, gpibBlkPtr-
>statusBlk->ibcnt);

            gpibBlkPtr->IOBufPtr = cmd;                 /* ibCMD    */
            gpibBlkPtr->IOCount = 2;
            osErr = Control (refnum, ibCMD, &gpibBlkPtr);
            printf("\n ibsta: %x iberr: %d ibcnt: %ld", gpibBlkPtr->
```

```
                 statusBlk->ibsta,
                   gpibBlkPtr->statusBlk->iberr, gpibBlkPtr-
>statusBlk->ibcnt);

        gpibBlkPtr->IOBufPtr = wrt;    /*     ibWRT              */
        gpibBlkPtr->IOCount = 6;
        osErr = Control (refnum, ibWRT, &gpibBlkPtr);
        printf("\n ibsta: %x iberr: %d ibcnt: %ld", gpibBlkPtr->
                 statusBlk->ibsta,
                   gpibBlkPtr->statusBlk->iberr, gpibBlkPtr-
>statusBlk->ibcnt);

        gpibBlkPtr->IOBufPtr = cmd2;   /*     ibCMD              */
        gpibBlkPtr->IOCount = 2;
        osErr = Control (refnum, ibCMD, &gpibBlkPtr);
        printf("\n ibsta: %x iberr: %d ibcnt: %ld", gpibBlkPtr->
                 statusBlk->ibsta,
                   gpibBlkPtr->statusBlk->iberr, gpibBlkPtr-
>statusBlk->ibcnt);

        gpibBlkPtr->IOBufPtr = rd;     /*     ibRD               */
        gpibBlkPtr->IOCount = 400;
        osErr = Control (refnum, ibRD, &gpibBlkPtr);
        printf("\n ibsta: %x iberr: %d ibcnt: %ld", gpibBlkPtr->
                 statusBlk->ibsta,
                   gpibBlkPtr->statusBlk->iberr, gpibBlkPtr-
>statusBlk->ibcnt);


}
```

Example Program—Low-Level Device Manager Calls

```
#include "stdio.h"
#include <DeviceMgr.h>

/* parameter block for low-level Control/Status calls typedef struct */
typedef struct{
   QElemPtr  qLink;
   short         qType;
   short         ioTrap;
   Ptr       ioCmdAddr;
   ProcPtr   ioCompletion;
   OsErr     ioResult;
   StringPtr ioNamePtr;
   short         ioVRefNum;
   short         ioRefNum;
   short         csCode;
                                   /*   driver routine ID       */
      Ptr                  niParam;  /*  pointer to NBParamBlock */
   } MyCntrlParam;
typedef MyCntrlParam *MyCntrlPtr;
```

```
typedef struct StatusBlk{
       short  ibsta;
       short  iberr;
       short  ibret;           /*  The four GPIB status variables*/
       long   ibcnt;
} StatusBlk;

typedef struct gpibBlock  {
   StatusBlk *statusBlk;
       short  id;
       short   controlVar;     /*  Control variable for some     */
                               /*  functions, to indicate nature */
                               /*  of action                     */
       Ptr          IOBufPtr;  /*  Pointer to the buffer in user */
                               /*  area, during I/O calls byte   */
       long   IOCount;         /*  count                         */
} gpibBlock;

enum {
   ibCAC, ibCMD, ibGTS, ibIST, ibLOC,
   ibONL, ibRD, ibRPP, ibRSC, ibPPC, ibWAIT,
   ibRSV, ibSIC, ibSRE, ibWRT, ibLLO,
   ibDMA, ibEOT, ibPAD, ibSAD, ibEOS,
   ibTRG, ibCLR, ibPCT, ibRSP, ibFIND,
   ibTMO, ibPOKE, ibSTAT};

char *brdName = "gpib4";
char *cmd = "@!";
char *cmd2 = " A";
char *wrt = "123456";
char *drvrName = "\p.GPIB Driver";

main ()

{
   short osErr
   short brd, dev1, GPIBRefNum, brdID;
   gpibBlock gpibBlk, *gpibBlkPtr;
   StatusBlk status;
   MyCntrlPtr cp;
   MyCntrlParam cpparam;
   char rd[500];

       gpibBlkPtr = &gpibBlk;   /*  setup control parameter block */
   gpibBlkPtr->statusBlk = &status;
   cp = &cpparam;
   cp->niParam = gpibBlkPtr;

   cp->ioNamePtr = drvrName;
       osErr = PBOpen(cp, FALSE);   /* refNum is returned in ioRefNum  */
       GPIBRefNum = cp->ioRefNum;   /* save refNum if using other      */
                                    /* drivers                         */
   if (osErr)
```

```
    printf("\nOpen error: %d\n", osErr);

    gpibBlkPtr->IOBufPtr = brdName; /* ibFIND                    */
    cp->csCode = ibFIND;
    osErr = PBControl (cp, FALSE);   /* brd descriptor returned in */
                                    /* gpibBlkPtr->id             */
    brdID = gpibBlkPtr->id;         /* save if changing between   */
                                    /* devices and boards         */
    printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr-
    >statusBlk-
           >ibsta,
       gpibBlkPtr->statusBlk->ibret, gpibBlkPtr->statusBlk->ibcnt);

    cp->csCode = ibSIC;           /*  ibSIC  */
osErr = PBControl (cp, FALSE);
    printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr->
           statusBlk->ibsta,
       gpibBlkPtr->statusBlk->iberr, gpibBlkPtr->statusBlk->ibcnt);

    gpibBlkPtr->IOBufPtr = cmd;   /*  ibCMD  */
gpibBlkPtr->IOCount = 2;
cp->csCode = ibCMD;
osErr = PBControl (cp, FALSE);
    printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr->
           statusBlk->ibsta,
       gpibBlkPtr->statusBlk->iberr, gpibBlkPtr->statusBlk->ibcnt);

    gpibBlkPtr->IOBufPtr = wrt;   /*  ibWRT  */
gpibBlkPtr->IOCount = 6;
cp->csCode = ibWRT;
osErr = PBControl (cp, FALSE);

    printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr->
           statusBlk->ibsta,
       gpibBlkPtr->statusBlk->iberr, gpibBlkPtr->statusBlk->ibcnt);

    gpibBlkPtr->IOBufPtr = cmd2;  /*  ibCMD  */
gpibBlkPtr->IOCount = 2;
cp->csCode = ibCMD;
osErr = PBControl (cp, FALSE);
    printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr->
           statusBlk->ibsta,
       gpibBlkPtr->statusBlk->iberr, gpibBlkPtr->statusBlk->ibcnt);

    gpibBlkPtr->IOBufPtr = rd;    /*  ibRD   */
gpibBlkPtr->IOCount = 1;
cp->csCode = ibRD;
osErr = PBControl (cp, FALSE);
    printf("\n ibsta: %x iberr: %d  ibcnt: %ld", gpibBlkPtr->
           statusBlk->ibsta,
       gpibBlkPtr->statusBlk->iberr, gpibBlkPtr->statusBlk->ibcnt);

}
```

Example Program—Accessing the GPIB Driver from THINK Pascal

```
{This program demonstrates how to access the GPIB driver from THINK}
{Pascal.  Access to the GPIB driver is through the "control" function.}
{Once the GPIB driver is opened, any GPIB function is executed by}
{setting the appropriate parameters and passing the function code to}
{the "control" Macintosh system function.  The Control system call}
{accepts a device driver reference number,a function code, and a}
{parameter block handle.  A handle is a pointer to a pointer.}

const
   ibFIND = 25;  {The function code for performing an ibFIND.}
   ibSIC = 12;   {The function code for performing an ibSIC.}
   ibRD = 6;     {The function code for performing an ibRD.}
   ibWRT = 14;   {The function code for performing an ibWRT.}

   noErr = 0;    {Zero response implies that there is no system error.}

type
   statusBlock = record{The status parameter block as the GPIB driver}
      ibsta : integer; {expects it to look in memory.}
      iberr : integer; {10 bytes.}
      ibret : integer;
      ibcnt : longint;
end;

SBlkPtr = ^statusBlock;{Defines a pointer type to the status block.}
gpibBlock = record       {The GPIB parameter block as the GPIB driver}
   statusPtr : SBlkPtr;{expects it to look in memory.}
   id : integer;         {16 bytes.}
   ControlVar : integer;
   IOBuf : Ptr;
   IOCount : longint;
end;

   gpibBlockPtr = ^GPIBBlock; {Defines a pointer type to the GPIB
block.}
var
   gpib : GPIBBlock;          {Allocates space for a GPIB block.}

   status : statusBlock;      {Allocates space for a status block.}

   gpibpointer : GPIBBlockPtr;  {Defines a pointer to a GPIB block.}

      RefNum : integer;
      osErr : integer;
      bdname : packed array[1..7] of char;
      boardID : integer;
      dvname : packed array[1..7] of char;
      deviceID : integer;
      message : packed array[1..40] of char;
      data : packed array[1..200] of char;
      i : integer;
```

```
begin
   showtext;

{Open GPIB driver.}
{-----------------------------------------------------------------}
{The GPIB driver must be opened before any GPIB operation can take}
{place. The OpenDriver system call accepts a driver name and returns its}
{reference number.From now on, access to that driver is through its}
{reference number.  The returned reference number should be negative.}
{If it is not, then something is wrong.  Check the parameter blocks!}

  osErr := OpenDriver('.GPIB Driver', RefNum);

{Check system error condition.}
{-----------------------------------------------------------------}
{osErr is the error returned by the Macintosh system.}
{If osErr is zero, then there are no system errors.}
{The most common error is -43.  This error occurs when the Macintosh}
{system  is not able to find the GPIB driver.}
{Make sure that "Installer" was used to install the GPIB driver and}
{that the correct name was passed to the OpenDriver system call. The}
{correct name is ".GPIB driver".}
{Notice the period at the beginning of the name.}

   if osErr = noErr then
      begin
         writeln('GPIB driver open operation was successful.');
         writeln('GPIB driver refnum = ', RefNum);
      end
   else
      begin
         writeln('GPIB driver open operation Failed. ');
         writeln('Macintosh system error is =  ', osErr);
      end;

{Setup pointer connections.}
{-----------------------------------------------------------------}
   gpibpointer := @gpib;  {Assign gpibpointer as a pointer to the GPIB}
                          {parameter block.}

  gpibpointer^.statusPtr := @status;{Assign statusPtr, an element of}
                                    {GPIB parameter block, as a}
                                    {pointer to the status block.}
                                    {Open the GPIB board.}
{-----------------------------------------------------------------}
{To open the GPIB board, an ibFIND function must be executed.}
{The ibFIND function accepts a name and returns an id.}
{The pointer to the name is passed in IOBuf.}
{The board id is returned in the GPIB block parameter, id.}
{If the ibFIND operation returns a -1, then the specified name}
{was not found in the GPIB driver.}
```

```
{Run the NI-488 Config application to make sure that the name is in}
{the GPIB driver.  Double check the name array.}

   bdname[1] := 'g';     {In this case, the board is in slot four.}
   bdname[2] := 'p';     {The name is defined by the slot number. So,}
   bdname[3] := 'i';     {the name is "GPIB4."  The name must be}
   bdname[4] := 'b';     {terminated by an ASCIINULL.  That is why}
   bdname[5] := '0';     {chr($00) is included as a terminating}
   bdname[6] := chr($00); {character.}

   gpibpointer^.IOBuf := @bdname;  {Make IOBuf, an element of the GPIB}
                                   {parameter block, point to the board}
   {name.}

{Do the ibFIND operation.}
   osErr := Control (RefNum, ibFIND, Ptr(@gpibpointer));

{Notice that the address of gpibpointer, and not gpibpointer itself,}
{was passed to the control function since it expects a handle, not a}
{simple pointer.  The PTR ( ) is used to cast the handle into the}
{proper pointer type.}

{Check system error condition.}
   if osErr = noErr then
      writeln('The board ibFIND operation was successful.')
   else
      begin
         writeln('The board ibFIND operation Failed. ');
         writeln('Macintosh system error is =  ', osErr);
      end;

   boardID := gpibpointer^.id; {Save the board id.}

{Check GPIB error condition.}
   if boardID < 0 then
      writeln('This name was not found in the GPIB driver table')
   else
      writeln('The board id is ', boardID);

{Issue an interface clear, IFC, by doing an ibSIC.}
{-----------------------------------------------------------------
}
{An ibSIC operation must be done after opening the board.}
{The board id is already in the id parameter of the GPIB block so there}
{is no need to reassign it.}

{gpibpointer^.id := boardID; No need to do this.}

   osErr := Control (RefNum, ibSIC, Ptr(@gpibpointer));

{Check system error condition.}
   if osErr = noErr then
      writeln('The ibSIC operation was successful.')
```

```
    else
        begin
            writeln('The ibSIC operation failed. ');
            writeln('Macintosh system error is =  ', osErr);
        end;

{Check GPIB error condition.}
    if gpibpointer^.statusPtr^.ibsta < 0 then
        writeln('GPIB error =', gpibpointer^.statusPtr^.iberr);

    writeln('GPIB status=', gpibpointer^.statusPtr^.ibsta);

{Open the device.}
{-----------------------------------------------------------------}
{In this case, the oscilloscope being used has a primary address of 1.}
{Since the default names are used, it would be the device with the}
{name: "dev1".  If a different name is used, then it must be assigned}
{to the device name array.  Again, a null ASCII character is needed as}
{a terminator.}

    dvname[1] := 'd';
    dvname[2] := 'e';
    dvname[3] := 'v';
    dvname[4] := '1';
    dvname[5] := chr($00);

{Assign IOBuf to point to the device name.}
    gpibpointer^.IOBuf := @dvname;

{Do the ibFIND operation.}

    osErr := Control (RefNum, ibFIND, Ptr(@gpibpointer));

{Check system error condition.}
    if osErr = noErr then
        writeln('The device ibFIND operation was successful.')
    else
        begin
            writeln('The device ibFIND operation failed. ');
            writeln('Macintosh system error is =  ', osErr);
        end;

    deviceID := gpibpointer^.id; {Save the device id.}

{Check GPIB error condition.}
    if deviceID < 0 then
        writeln('This name was not found in the GPIB driver table')
    else
        writeln('The device id is ', deviceID);

{Send a command to the device.}
{-----------------------------------------------------------------}
```

```
{This particular oscilloscope responds to the query "ID?" by sending}
{back 26 bytes of identification.  This message must be changed to suit}
{the instrument being used.}

{Assign the query to the message array.}
{Notice that no termination string is needed since this is not a name.}
   message[1] := 'I';
   message[2] := 'D';
   message[3] := '?';

{The device id, a pointer to the message and a byte count, must be}
{specified in the GPIB block before executing the ibWRT operation.}
{The device id is still in the GPIB block.  There is no need to}
{reassign it.}

{gpibpointer^.id := deviceID; No need to do this.}

   gpibpointer^.IOBuf := @message; {IOBuf points to the message.}
   gpibpointer^.IOCount := 3; {The message contains three bytes.}

{Send the message using the ibWRT function.}

   osErr := Control (RefNum, ibWRT, Ptr(@gpibpointer));

{Check system error condition.}
   if osErr = noErr then
      writeln('The device ibWRT operation was successful.')
   else
      begin
         writeln('The device ibWRT operation failed. ');
         writeln('Macintosh system error is =  ', osErr);
      end;

{Check GPIB error condition.}
   if gpibpointer^.statusPtr^.ibsta < 0 then
      writeln('GPIB error =', gpibpointer^.statusPtr^.iberr)
   else
      writeln(gpibpointer^.statusPtr^.ibcnt, ' bytes were sent.');

   writeln('GPIB status=', gpibpointer^.statusPtr^.ibsta);

{Read a response from the device.}
{-------------------------------------------------------------------}
{This particular oscilloscope responds to the query "ID?" by sending}
{back 26 bytes of identification.}
{The device id, a pointer to a data buffer and a byte count, must be }
{specified in the GPIB block before executing the ibRD operation.}
{The device id is still in the id parameter of the GPIB block from the}
{last operation.  There is no need to assign it again.}
{gpibpointer^.id := deviceID; No need to do this unless the id}
{parameter in the GPIB block had been changed by issuing commands to}
{a different device or board.}
```

```
{Assign IOBuf as a pointer to a data buffer large enough to hold the
response.}
   gpibpointer^.IOBuf := @data;

{Read up to 200 bytes.  In this case, the read is terminated on byte 26}
{since the oscilloscope asserts the EOI line with the last byte sent.}
{The read mode is set, in the NI-488 Config application, to terminate}
{on count or EOI.}
  gpibpointer^.IOCount := 200;
{Execute the ibRD operation}

   osErr := Control (RefNum, ibRD, Ptr(@gpibpointer));

{Check system error condition}
   if osErr = noErr then
      writeln('The device ibRD operation was successful.')
   else
      begin
         writeln('The device ibRD operation failed. ');
         writeln('Macintosh system error is =  ', osErr);
      end;

{Check GPIB error condition.}
   if gpibpointer^.statusPtr^.ibsta < 0 then
      writeln('GPIB error =', gpibpointer^.statusPtr^.iberr)
   else
      writeln(gpibpointer^.statusPtr^.ibcnt, ' bytes were read.');

   writeln('GPIB status=', (gpibpointer^.statusPtr^.ibsta));

{Display data buffer.}
   writeln('The response is:');
   for i := 1 TO gpibpointer^.statusPtr^.ibcnt do
      write(data[i]);

end.
```

List of NI-488.2 Device Manager Routines

Table 3-6 lists the syntax for each NI-488.2 routine and a brief description of these routines for the Device Manager.

Table 3-6. Device Manager NI-488.2 Routines

Routine	Fields	Cntrl No.	Description
FC_allspoll	id, addr, result	68	Serial poll all devices
FC_devclear	id, addr	50	Clear a single device
FC_devclearlist	id, addr	52	Clear multiple devices
FC_enablelocal	id, addr	53	Enable operations from the front panel of a device
FC_enableremote	id, addr	54	Enable remote GPIB programming of devices
FC_findlstn	id, addr, result, limit	69	Find all Listeners
FC_findrqs	id, addr, result	67	Determine which device is requesting service
FC_passcontrol	id, addr	58	Pass control to another device with Controller capability
FC_ppoll	id, result	63	Perform a parallel poll
FC_ppollconfig	id, addr, limit, controlVar	61	Configure a device for parallel polls
FC_ppollunconfig	id, addr	62	Unconfigure devices for parallel polls
FC_rcvrespmsg	id, IOBufPtr, IOCount, controlVar	48	Read data bytes from addressed device
FC_readstatusbyte	id, addr, result	59	Serial poll a single device to get its status byte
FC_receive	id, addr, IOBufPtr,IOCount, controlVar	49	Read data bytes from a GPIB device
FC_receivesetup	id, addr	47	Prepare a specified device to send data bytes and prepare the GPIB board to read them

(continues)

Table 3-6. Device Manager NI-488.2 Routines (Continued)

Routine	Fields	Cntrl No.	Description
FC_resetsys	id, addr	66	Initialize a GPIB system on three levels
FC_send	id, addr, IOBufPtr, IOCount, controlVar	46	Send data bytes to a single GPIB device
FC_sendcmds	id, IOBufPtr, IOCount	43	Send GPIB command bytes
FC_senddatabytes	id, IOBufPtr, IOCount, controlVar	45	Send data bytes to addressed devices
FC_sendifc	id	77	Clear the GPIB interface functions with IFC
FC_sendlist	id, addr, IOBufPtr, IOCount, controlVar	51	Send data bytes to multiple GPIB devices
FC_sendllo	id	56	Send the Local Lockout message to all devices
FC_sendsetup	id, addr	44	Prepare specified devices to receive data bytes
FC_sendrwls	id, addr	55	Place specified devices in the Remote With Lockout state
FC_testsrq	id, result	64	Determine the current state of the SRQ line
FC_testsys	id, addr, result	70	Cause devices to conduct self-tests
FC_trigger	id, addr	57	Trigger a single device
FC_triggerlist	id, addr	60	Trigger multiple devices
FC_waitsrq	id, result	65	Wait until a device asserts Service Request

NI-488.2 Routine Descriptions

The remainder of this chapter contains a detailed description of each NI-488.2 routine with examples. The descriptions are listed alphabetically for easy reference. In this chapter, short and long types refer to 16-bit and 32-bit integers, respectively.

AllSpoll AllSpoll

Purpose

Serial poll all devices.

Control Number

67

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | pointer to addresslist
short*    result   ->  | pointer to resultlist
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a board number. The parameter `addr` is an array of address integers of any size, terminated by the value NOADDR. The GPIB devices whose addresses are contained in the address array are serial polled, and the responses are stored in the corresponding elements of the `result` array.

If any of the specified devices times out instead of responding to the poll, the error code EABO is returned in `iberr`, and `ibcnt` contains the index of the timed-out device.

Although the `AllSpoll` routine can serial poll any number of GPIB devices, use the `ReadStatusByte` routine to poll only one GPIB device.

Example

```
paramBlk->id = id
paramBlk->addr = listen;   /* address list pointer */
paramBlk->result = res;    /* pointer to result array */
osErr =   Control (refNum,FC_allspoll,&paramBlk);
```

DevClear DevClear

Purpose

Clear a single device.

Control Number 50

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | device address
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a board number. The GPIB Selected Device Clear (SDC) message is sent to the device at the given address. The parameter `addr` is a pointer to the GPIB address of the device to be cleared. It should contain in its low byte the primary GPIB address of the device to be cleared. The high byte should be 0 if the device has no secondary address. Otherwise, it should contain the desired secondary address. If the parameter `addr` contains the constant value NOADDR, the Universal Device Clear message is sent to all devices on the GPIB.

The `DevClear` routine clears either one GPIB device, or all GPIB devices. To send a single message that clears several specified GPIB devices, use the `DevClearList` routine.

Example

```
paramBlk->id = id;
paramBlk->addr = &listen;      /* listen contains the address */
osErr =   Control (refNum,FC_devclear,&paramBlk);
```

 © National Instruments Corp.

DevClearList DevClearList

Purpose

Clear multiple devices.

Control Number

52

Parameter Block Fields

```
short     id      ->  | board id
short*    addr    ->  | device address list pointer
short     ibsta   <-  | status variables
short     iberr   <-  | returned
long      ibcnt   <-  |
```

`id` is a board number. The GPIB devices whose addresses are contained in the address array are cleared. The parameter `listen` is an array for any size of address integers, terminated by the value NOADDR.

Although the `DevClearList` routine can clear any number of GPIB devices, use the `DevClear` routine to clear only one GPIB device.

If the array contains only the value NOADDR, the universal Device Clear message is sent.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;
osErr =   Control (refNum,FC_devclearlist,&paramBlk);
```

EnableLocal

EnableLocal

Purpose

Enable operations from the front panel of a device.

Control Number

53

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | device address list pointer
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a board number. The GPIB devices whose addresses are contained in the `addr` array are placed in local mode by addressing the devices as Listeners and sending the GPIB Go To Local command. The parameter `addr` is an array for any size of address integers, terminated by the value NOADDR.

If the array contains only the value NOADDR, Remote Enable (REN) becomes unasserted, immediately placing all GPIB devices in local mode.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;
osErr =   Control (refNum,FC_enablelocal,&paramBlk);
```

EnableRemote EnableRemote

Purpose

Enable remote GPIB programming of devices.

Control Number

54

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | device address list pointer
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a board number. The GPIB devices whose addresses are contained in the `addr` array are placed in remote mode by asserting Remote Enable (REN) and addressing the devices as Listeners. The parameter `addr` is an array for any size of address integers, terminated by the value NOADDR.

If the array contains only the value NOADDR, no addressing is performed, and Remote Enable (REN) becomes asserted.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;
osErr =   Control (refNum,FC_enableremote,&paramBlk);
```

FindLstn FindLstn

Purpose

Find all Listeners.

Control Number 69

Parameter Block Fields

```
short    id      ->  board id
short*   addr    ->  pointer to addresslist
short*   result  ->  pointer to resultlist
short    limit   ->  number of entries in
                   resultlist
short    ibsta   <-  status variables
short    iberr   <-  returned
long     ibcnt   <-
```

`id` is a board number. `addr` contains a list of primary GPIB addresses, terminated by the value NOADDR. These addresses are tested in turn for the presence of a listening device. If Listeners are found, the addresses are entered into `result`. If no listening device is detected at a particular primary address, all the secondary addresses associated with that primary address are tested, and detected Listeners are entered into `result`. The `limit` argument specifies how many entries should be placed into the `result` array.

If more Listeners are present on the bus, the list is truncated after `limit` entries have been detected, and the error ETAB is reported in `iberr`. The variable `ibcnt` will contain the number of addresses placed into `result`.

Because there may be multiple secondary addresses that respond as Listeners for any given primary address, the `result` array in general should be larger than the `addr` array. In any event, the `result` array (with `limit` being the maximum possible results) must be large enough to accommodate all expected listening devices because no check is made for overflow of the array.

Because most GPIB devices have the ability to listen, this routine is normally used to detect the presence of devices at particular addresses. Once detected, they usually can be interrogated by identification messages to identify them.

FindLstn

FindLstn (Continued)

Example

```
paramBlk->id = id;
paramBlk->addr = pads;
paramBlk->result = result;
paramBlk->limit = limit;
osErr =   Control (refNum,FC_findlstn,&paramBlk);
```

FindRQS FindRQS

Purpose

Determine which device is requesting service.

Control Number 67

Parameter Block Fields

```
short     id       ->  board id
short*    addr     ->  pointer to addresslist
short*    result   ->  pointer to result
short     ibsta    <-  status variables
short     iberr    <-  returned
long      ibcnt    <-
```

`id` is a board number. `addr` contains a list of primary GPIB addresses, terminated by the value NOADDR. Starting from the beginning of the `addr`, the indicated devices are serial polled until one is found that is asserting SRQ. The status byte for this device is returned in the variable `result`. In addition, the index of the device address in `listen` is returned in the global variable `ibcnt`.

If none of the specified devices is requesting service, the error code ETAB is returned in `iberr`, and `ibcnt` contains the index of the NOADDR entry of the list.

If a device times out while responding to its serial poll, the error code EABO is returned in `iberr`, and the index of the timed-out device appears in `ibcnt`.

Example

```
paramBlk->id = id;
paramBlk->addr = listen ;
paramBlk->result = dev_stat;
osErr =   Control (refNum,FC_findrqs,&paramBlk);
```

PassControl PassControl

Purpose

Pass control to another device with Controller capability.

Control Number

58

Parameter Block Fields

```
short     id      ->  | board id
short*    addr    ->  | pointer to device address
short     ibsta   <-  | status variables
short     iberr   <-  | returned
long      ibcnt   <-  |
```

`id` is a board number. The GPIB Device Take Control message is sent to the device at the given address. The parameter `addr` points to a device address to which control should be transferred. The address contains in its low byte the primary GPIB address of the device to receive control. The high byte should be 0 if the device has no secondary address. Otherwise, it should contain the desired secondary address.

Example

```
paramBlk->id = id;
paramBlk->addr = &talkaddr;/* talkaddr contains the device address*/
osErr =   Control (refNum,FC_passcontrol,&paramBlk);
```

PPoll PPoll

Purpose

Perform a parallel poll.

Control Number

63

Parameter Block Fields

```
short     id       ->  | board id
short*    result   ->  | pointer to result
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a board number. A parallel poll is conducted, and the 8-bit result is stored into `result.` Only the lower eight bits of the result are affected.

Each bit of the poll result returns one bit of status information from each device that has been configured for parallel polls. The state of each bit (0 or 1), and the interpretation of these states are based on the latest parallel poll configuration sent to the devices and the individual status of the devices.

Example

```
paramBlk->id = id;
paramBlk->result = res_ptr;
osErr =   Control (refNum,FC_ppoll,&paramBlk);
```

PPollConfig PPollConfig

Purpose

Configure a device for parallel polls.

Control Number

61

Parameter Block Fields

```
short    id            ->  | board id
short*   addr          ->  | device address
short    limit         ->  | dataline for response
short    controlVar    ->  | condition to assert or
                       unassert
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is a board number. The GPIB device pointed to by `addr` is configured for parallel polls according to the `controlVar` and `limit` parameters. `limit` is the data line (1-8) on which the device is to respond, and `controlVar` indicates the condition under which the data line is to be asserted or unasserted. The device is expected to compare this sense value (0 or 1) to its individual status bit, and respond accordingly.

Devices have the option of configuring themselves for parallel polls, in which case they are to ignore attempts by the Controller to configure them. Determine whether the device is locally or remotely configurable before using `PPollConfig` or `PPollUnconfig`.

Example

```
paramBlk->id = id;
paramBlk->addr = value;
paramBlk->limit = dataline;
paramBlk->controlVar = sense;
osErr =   Control (refNum,FC_ppollconfig,&paramBlk);
```

PPollUnconfig PPollUnconfig

Purpose

Unconfigure devices for parallel polls.

Control Number

62

Parameter Block Fields

```
short    id      ->  | board id
short*   addr    ->  | device address list pointer
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is a board number. The GPIB devices whose addresses are contained in the address array are unconfigured for parallel polls; that is, they no longer participate in polls. The parameter `addr` is an array of address integers of any size, terminated by the value NOADDR.

If the array contains only the value NOADDR, the GPIB Parallel Poll Unconfigure (PPU) is sent, unconfiguring all devices.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;
osErr =   Control (refNum,FC_ppollunconfig,&paramBlk);
```

RcvRespMsg RcvRespMsg

Purpose

Read data bytes from addressed device.

Control Number 48

Parameter Block Fields

```
short    id           ->  | board id
char*    IOBufPtr     ->  | Pointer to buffer to receive
                          | data
long     IOCount      ->  | number of bytes to receive
short    controlVar   ->  | termination character
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is a board number. Up to `count` data bytes are read from the GPIB and placed into the pre-allocated string `data`. The parameter `controlVar` is a flag used to describe the method of signaling the end of the data. If it has a value between 0 and hex 00FF, the ASCII character with the corresponding hex value is considered the termination character, and the read is stopped when the character is detected.

`RcvRespMsg` assumes that the GPIB Talker and Listeners have already been addressed by a prior call to routines such as `ReceiveSetup`, `Receive`, or `SendCmds`. Therefore, it is used specifically to skip the addressing step of GPIB management. The `Receive` routine is normally used to accomplish the entire sequence of addressing followed by the reception of data bytes.

Example

```
paramBlk->id = id;
paramBlk->IOBufPtr = IOBufPtr;
paramBlk->IOCount = count;
paramBlk->controlVar = eotmode;
osErr =   Control (refNum,FC_rcvrespmsg,&paramBlk);
```

ReadStatusByte ReadStatusByte

Purpose

Serial poll a single device to get its status byte.

Control Number

59

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | Pointer to device address
short*    result   ->  | Pointer to result
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is a board number. The indicated device is serial polled, and its status byte is placed into the variable `result`, with the status byte zero-extended into the upper byte.

Example

```
paramBlk->id = id;
paramBlk->addr = talkaddr;
paramBlk->result = result;
osErr =   Control (refNum,FC_readstatusbyte,&paramBlk);
```

Receive Receive

Purpose

Read data bytes from a GPIB device.

Control Number

49

Parameter Block Fields

```
short    id          ->  board id
short*   addr        ->  Pointer to device address
char*    IOBufPtr    ->  Pointer to buffer to receive
                         data
long     IOCount     ->  number of bytes to receive
short    controlVar  ->  termination character
short    ibsta       <-  status variables
short    iberr       <-  returned
long     ibcnt       <-
```

`id` is a board number. The indicated GPIB device is addressed, and up to `count` data bytes are read from that device and placed into the pre-allocated string pointed to by `IOBufPtr. eotmode` is a value used to describe the method of signaling the end of the data. If it is a value between 0 and hex 00FF, the ASCII character with the corresponding hex value is considered the termination character, and the read is stopped when the character is detected. If `eotmode` is the constant STOPEND (defined in the header file `decl.h`), the read is stopped when EOI is detected.

Example

```
paramBlk->id = id;
paramBlk->IOBufPtr = IOBufPtr;
paramBlk->addr = talkaddr;
paramBlk->IOCount = count;
paramBlk->controlVar = eotmode;
osErr =   Control (refNum,FC_receive,&paramBlk);
```

ReceiveSetup

ReceiveSetup

Purpose

Prepare a specified device to send data bytes and prepare the GPIB interface board to read them.

Control Number

47

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | Pointer to device address
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is an interface board. The GPIB device at address `addr` is addressed to listen. The indicated GPIB device is addressed as a Talker, and the indicated board is addressed as a Listener. Following this routine, it is common to call a routine such as `RcvRespMsg` to actually transfer the data from the Talker.

This routine is useful to initially address devices in preparation for receiving data, followed by multiple calls of `RcvRespMsg` to receive multiple blocks of data, thus eliminating the need to re-address the devices between blocks. Alternatively, the `Receive` routine could be used to send the first data block, followed by `RcvRespMsg` for all subsequent blocks.

Example

```
paramBlk->id = brdID;          /* board number */
paramBlk->addr = talkadr;      /* Pointer to address of device */
osErr = Control (refNum,FC_receivesetup,&paramBlk);
/* Call Device Manager */
```

ResetSys ResetSys

Purpose

Initialize a GPIB system on three levels.

Control Number 66

Parameter Block Fields

```
short     id       ->  board id
short*    addr     ->  Ptr to array of device
                   addresses
short     ibsta    <-  status variables
short     iberr    <-  returned
long      ibcnt    <-
```

`id` is an interface board. `addr` is an addresslist terminated by the value NOADDR. The GPIB system is initialized on the following three levels.

• Bus initialization	Remote Enable (REN) is asserted, followed by Interface Clear (IFC), causing all devices to become unaddressed and the GPIB interface board (the System Controller) to become the Controller-in-Charge.
• Message exchange initialization	The Device Clear (DCL) message is sent to all connected devices. This ensures that all 488.2-compatible devices receive the Reset (RST) message that follows.
• Device initialization	`*RST` message is sent to all devices whose addresses are contained in the `addr` argument. This causes device-specific functions within each device to be initialized.

Example

```
paramBlk->id = id;              /* board number  */
paramBlk->addr = listen;    /* Ptr to addresslist */
osErr = Control (refNum,FC_resetsys,&paramBlk);
/* Call Device Manager */
```

Send Send

Purpose

Send data bytes to a single GPIB device.

Control Number

46

Parameter Block Fields

```
short    id           ->  | board Id
short*   addr         ->  | Ptr to device address
char*    IOBufPtr     ->  | Ptr to data buffer
long     IOCount      ->  | number of bytes to send
short    controlVar   ->  | data termination mode
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is an interface board. `listen` is a pointer to a device address. `IOBufPtr` is a pointer to a data buffer. `IOCount` is the number of bytes to be sent over the GPIB. The indicated GPIB device is addressed as a Listener, the indicated board is addressed as a Talker, and `count` data bytes contained in `data` are sent. `controlVar` is a flag used to describe the method of signaling the end of the data to the Listener. It should be set to one of the following constants.

- `NLend` Send NL (linefeed) with EOI after the data bytes.
- `DABend` Send EOI with the last data byte in the string.
- `NULLend` Do nothing to mark the end of the transfer.

These constants are defined in the header file `decl.h.`

Example

```
paramBlk->id = id;
paramBlk->IOBufPtr = buffer;       /* Ptr to data buffer */
paramBlk->addr = listen;           /* Ptr to address     */
paramBlk->IOCount = cnt;           /* buffer length      */
paramBlk->controlVar = eotmode;    /* termination mode   */
osErr = Control (refNum,FC_send,&paramBlk);
/* Call Device Manager */
```

SendCmds SendCmds

Purpose

Send GPIB command bytes.

Control Number

43

Parameter Block Fields

```
short    id          ->  board Id
char*    IOBufPtr    ->  Ptr to data buffer
long     IOCount     ->  number of commands to send
short    ibsta       <-  status variables
short    iberr       <-  returned
long     ibcnt       <-
```

`id` is an interface board. `IOBufPtr` is a pointer to a data buffer. `IOCount` is the number of bytes to be sent over the GPIB. Although it is a long value in this language, an integer value can also be passed.

`SendCmds` is not normally required for GPIB operation. It is used to send special command sequences to the GPIB when other routines are not provided.

Example

```
paramBlk->id = id;
paramBlk->IOCount = cnt;   /* number of commands being sent */
paramBlk->IOBufPtr = buf;  /* Ptr to command buffer */
osErr = Control (refNum,FC_sendcmds,&paramBlk);
/* Call Device Manager */
```

SendDataBytes SendDataBytes

Purpose

Send data bytes to addressed devices.

Control Number

45

Parameter Block Fields

```
short    id            ->  | board Id
char*    IOBufPtr      ->  | Ptr to data buffer
long     IOCount       ->  | number of data bytes to send
short    controlVar    ->  | termination mode
short    ibsta         <-  | status variables
short    iberr         <-  | returned
long     ibcnt         <-  |
```

`id` is an interface board. `IOBufPtr` is a pointer to a data buffer. `IOCount` is the number of bytes to be sent over the GPIB. `controlVar` is a flag used to describe the method of signaling the end of the data to the Listeners. Set this flag to one of the following constants.

- `NLend` Send NL (linefeed) with EOI after the data bytes.
- `DABend` Send EOI with the last data byte in the string.
- `NULLend` Do nothing to mark the end of the transfer.

These constants are defined in the header file `decl.h.`

`SendDataBytes` assumes that all GPIB Listeners have already been addressed by a prior call to functions such as `SendSetup`, `Send`, or `SendCmds.` Thus, it is used specifically to skip the addressing step of GPIB management. The `Send` routine is normally used to accomplish the entire sequence of addressing followed by the transmission of data bytes.

Example

```
paramBlk->id = id;
paramBlk->IOBufPtr = buf;          /* Ptr to data buffer */
paramBlk->IOCount = cnt;           /* buffer length      */
paramBlk->controlVar = eotmode;    /* termination mode   */
osErr = Control (refNum,FC_senddatabytes,&paramBlk);
/* Call Device Manager */
```

SendIFC SendIFC

Purpose

Clear the GPIB interface functions with IFC.

Control Number

77

Parameter Block Fields

```
short    id      ->  | board Id
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is an interface board. When the GPIB Device IFC message is issued, the interface functions of all connected devices return to their cleared states.

This function is used as part of GPIB initialization. It forces the GPIB interface board to be Controller of the GPIB, and ensures that the connected devices are all unaddressed and that the interface functions of the devices are in the idle state.

Example

```
paramBlk->id = id;             /* board number */
osErr = Control (refNum,FC_sendifc,&paramBlk);
/* Call Device Manager */
```

SendList SendList

Purpose

Send data bytes to multiple GPIB devices.

Control Number

51

Parameter Block Fields

```
short    id           ->  | board Id
short*   addr         ->  | Ptr to device addresslist
char*    IOBufPtr     ->  | Ptr to data buffer
long     IOCount      ->  | number of bytes to send
short    controlVar   ->  | termination mode
short    ibsta        <-  | status variables
short    iberr        <-  | returned
long     ibcnt        <-  |
```

`id` is an interface board. `addr` is a pointer to an addresslist array terminated by the value NOADDR. `IOBufPtr` is a pointer to a data buffer. `IOCount` is the number of bytes to be sent over the GPIB. The indicated GPIB device is addressed as a Listener, and the indicated board is addressed as a Talker. `controlVar` is a flag used to describe the method of signaling the end of the data to the Listener. Set this flag to one of the following constants.

- `NLend` Send NL (linefeed) with EOI after the data bytes.
- `DABend` Send EOI with the last data byte in the string.
- `NULLend` Do nothing to mark the end of the transfer.

These constants are defined in the header file `decl.h.`

This routine is similar to `Send`, except that multiple Listeners are able to receive the data with only one transmission.

Example

```
paramBlk->id = id;
paramBlk->IOBufPtr = buf;          /* Ptr to data buffer      */
paramBlk->addr = listen;           /* Ptr to addresslist      */
paramBlk->IOCount = cnt;           /* number of bytes to send */
paramBlk->controlVar = eotmode;    /* data termination mode   */
osErr = Control (refNum,FC_sendlist,&paramBlk);
/* Call Device Manager */
```

SendLLO SendLLO

Purpose

Send the Local Lockout message to all devices.

Control Number

56

Parameter Block Fields

```
short     id        ->  | board Id
short     ibsta     <-  | status variables
short     iberr     <-  | returned
long      ibcnt     <-  |
```

`id` is an interface board. The GPIB Local Lockout message is sent to all devices, preventing them from independently choosing the local or remote states. While Local Lockout is in effect, only the Controller can alter the local or remote state of the devices by sending appropriate GPIB messages.

`SendLLO` is reserved for use in unusual local/remote situations, particularly those in which all devices are to be locked into local programming state. In the typical case of placing devices in Remote Mode With Lockout state, use the `SendRWLS` routine.

Example

```
paramBlk->id = id;
osErr = Control (refNum,FC_sendllo,&paramBlk);
```

SendSetup SendSetup

Purpose

Prepare specified devices to receive data bytes.

Control Number

44

Parameter Block Fields

```
short     id      ->  | board id
short*    addr    ->  | Ptr to array of device
                  addresses
short     ibsta   <-  | status variables
short     iberr   <-  | returned
long      ibcnt   <-  |
```

`id` is an interface board. `addr` is an addresslist terminated by the value NOADDR. The GPIB devices whose addresses are contained in the addresslist array are addressed as Listeners, and the indicated board is addressed as a Talker. Following this call, it is common to call a routine such as `SendDataBytes` to actually transfer the data to the Listeners.

This command is useful for initially addressing devices in preparation for sending data, followed by multiple calls of `SendDataBytes` to send multiple blocks of data, thus eliminating the need to re-address the devices between blocks. Alternatively, the `Send` routine can be used to send the first data block, followed by `SendDataBytes` for all subsequent blocks.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;  /* Ptr to addresslist */
osErr = Control (refNum,FC_sendsetup,&paramBlk);
/* Call Device Manager */
```

SendRWLS SendRWLS

Purpose

Place specified devices in the Remote With Lockout state.

Control Number

55

Parameter Block Fields

```
short     id      ->  | board id
short*    addr    ->  | Ptr to array of device
                  addresses
short     ibsta   <-  | status variables
short     iberr   <-  | returned
long      ibcnt   <-  |
```

`id` is an interface board. `addr` is an addresslist terminated by the value NOADDR. The GPIB devices whose addresses are contained in the `addresslist` array are placed in remote mode by asserting Remote Enable (REN) and addressing the devices as Listeners. In addition, all devices are placed in Lockout State, which prevents them from independently returning to local programming mode without passing through the Controller.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;  /* Ptr to addresslist */
osErr = Control (refNum,FC_sendrwls,&paramBlk);
/* Call Device Manager */
```

TestSRQ TestSRQ

Purpose

Determine the current state of the SRQ line.

Control Number

64

Parameter Block Fields

```
short    id      ->  | board id
short*   result  ->  | Ptr to result holder
short    ibsta   <-  | status variables
short    iberr   <-  | returned
long     ibcnt   <-  |
```

`id` is an interface board. `result` is a pointer to an integer. This call places the value 1 in the variable `result` if the GPIB SRQ line is asserted. Otherwise, it places the value 0 into `result`.

This routine is similar in format to the `WaitSRQ` routine, except that `WaitSRQ` suspends itself while waiting for an occurrence of SRQ, whereas `TestSRQ` returns immediately with the current SRQ state.

Example

```
paramBlk->id = id;
paramBlk->result = result;     /* Ptr to result short */
osErr = Control (refNum,FC_testsrq,&paramBlk);
/* Call Device Manager */
```

TestSys TestSys

Purpose

Cause devices to conduct self-tests.

Control Number

70

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | Ptr to address list
short*    result   ->  | Ptr to result list
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is an interface board. The parameter `addr` is an array of address integers of any size, terminated by the value NOADDR. The GPIB devices whose addresses are contained in the `addrs` array simultaneously receive a message that instructs them to conduct their self-test procedures. Each device returns an integer code signifying the results of its tests, and these codes are placed into the corresponding elements of the `result` array. The IEEE 488.2 standard specifies that a result code of 0 indicates that the device passed its tests, and any other value indicates that the tests resulted in an error. The variable `ibcnt` contains the number of devices that failed their tests.

Example

```
paramBlk->id = id;
paramBlk->addr = addrs;       /* Ptr to addresslist array */
paramBlk->result = result;    /* Ptr to result array      */
osErr = Control (refNum,FC_testsys,&paramBlk);
/* Call Device Manager */
```

Trigger Trigger

Purpose

Trigger a single device.

Control Number

57

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | Ptr to address
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is an interface board. The GPIB Group Execute Trigger message is sent to the device at the given address. The parameter `addr` is a pointer to an short integer that contains in its low byte the primary GPIB address of the device to be triggered. The high byte should be 0 if the device has no secondary address. Otherwise, it should contain the desired secondary address. If the address is NOADDR, the Group Execute Trigger message is sent with no addressing, thereby triggering all previously addressed Listeners.

The `Trigger` routine triggers only one GPIB device. To send a single message that triggers several GPIB devices, use the `TriggerList` routine.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;  /* Ptr to address */
osErr = Control (refNum,FC_trigger,&paramBlk);
/* Call Device Manager */
```

TriggerList TriggerList

Purpose

Trigger multiple devices.

Control Number

60

Parameter Block Fields

```
short     id       ->  | board id
short*    addr     ->  | Ptr to address array
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is an interface board. The GPIB devices whose addresses are contained in the address array are triggered simultaneously. If the array contains only the value NOADDR, the Group Execute Trigger message is sent without addressing, thereby triggering all previously addressed Listeners.

Although the `TriggerList` routine can trigger any number of GPIB devices, use the `Trigger` function to trigger only one GPIB device.

Example

```
paramBlk->id = id;
paramBlk->addr = listen;  /* Ptr to address array */
osErr = Control (refNum,FC_triggerlist,&paramBlk);
/* Call Device Manager */
```

WaitSRQ WaitSRQ

Purpose

Wait until a device asserts Service Request.

Control Number

65

Parameter Block Fields

```
short     id       ->  | board id
short*    result   ->  | Ptr to result holder
short     ibsta    <-  | status variables
short     iberr    <-  | returned
long      ibcnt    <-  |
```

`id` is an interface board. This routine suspends execution of the program until a GPIB device connected to the indicated board asserts the Service Request (SRQ) line. If the SRQ occurs within the timeout period, the variable `result` is set to the value 1. If no SRQ is detected before the timeout period expires, `result` is set to 0.

Notice that this call is similar in format to the `TestSRQ` routine, except that `TestSRQ` returns immediately with SRQ status, whereas `WaitSRQ` suspends the program during the timeout period until an SRQ occurs.

Example

```
paramBlk->id = id;
paramBlk->result = result;     /* Ptr to result short */
osErr = Control (refNum,FC_waitsrq,&paramBlk);
/* Call Device Manager */
```

 © National Instruments Corp.

Device Manager GPIB Programming Example

You can take full advantage of the ANSI/IEEE Standard 488.2-1987 by using the NI-488.2 routines. These routines are completely compatible with the Controller commands and protocols defined in the IEEE 488.2 standard.

The NI-488.2 routines are easy to learn and use. Only a few routines are needed for most application programs.

This example illustrates the programming steps you can use to program a representative IEEE 488.2 instrument from your Macintosh computer using the NI-488.2 routines for the Device Manager.

The NI-488.2 driver supports up to eight interfaces. These interfaces are referenced by number from your application program. The reference number is zero (0) for the first board, one (1) for the second board, and so on. If you installed multiple interfaces in your computer, and you do not know which number references which board, run the `NI-488 Config` control panel configuration utility. The control panel shows you the relationship between the board number and the physical slot or port in the computer. Refer to Chapter 6, *GPIB Configuration Utility*, in the *NI-488.2 User Manual for Macintosh* for additional information about running and using `NI-488 Config`.

Device Manager Example Program—NI-488.2 Routines

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*     Example High Level Control Calls for Device Manager      */
/*                                                               */
/*                           488.2                               */
/*                                                               */
/*                                                               */
/*****************************************************************/


#include <stdio.h>

/* Include MacTraps in the Project for OpenDriver call */
/* Include ANSI in the Project for any printf calls    */
/* Function Control Numbers (in DrInterface.h) */

typedef          short  short;
typedef unsigned short ushort;
typedef          long   long;
typedef unsigned long  ulong;

typedef struct StatusBlk
{
```

```
    short ibsta;        /* The GPIB status variables */
    short iberr;
    short ibret;
    long  ibcnt;
} StatusBlk;

typedef gpibBlock *gpibBlockPtr;



main()
{
    short refNum;
    short osErr, brdID, i, NumDevices;
    gpibBlock paramBlk, *pb;   /* parameter block for the driver */
    StatusBlk statusBlk;
    char  buffer[1000];
    ushort   addrs[30];
    ushort results[30];


    osErr = OpenDriver("\p.GPIB Driver",&refNum);
    if(osErr)
    {
       printf("\n\nOpenDriver error: %d",osErr);
       return -1;
    }

    pb = &paramBlk;
    pb->statusBlk = &statusBlk;
    /* Tell Driver where to put our status vars */


/* SendIFC */

    pb->id = 0;
    Control(refNum,FC_sendifc,&pb);   /* Open up Board */


    brdID = pb->id;  /* Save if changing between devices and
boards */


    printf("\n ibsta: %x",statusBlk.ibsta);
    if(statusBlk.ibsta & 0x8000)
      printf("\n iberr: %d",statusBlk.iberr);
    else
      printf("\n");
```

```
/* FindLstn */

    for(i=0;i!=31;i++)
       addrs[i] = (ushort)i;

    addrs[i] = 0xFFFF;  /* NOADDR (0xffff) terminates address list */

    pb->addr = addrs;
    pb->result = results;
    pb->limit = 5;

    Control(refNum,FC_findlstn,&pb);


    printf("\n ibsta: %x",statusBlk.ibsta);
    if(statusBlk.ibsta & 0x8000)
      printf("\n iberr: %d",statusBlk.iberr);
    else
      printf("\n");


    printf("\n %ld Listener(s) on the Bus. ",statusBlk.ibcnt);

    NumDevices = statusBlk.ibcnt;


/* DevClearList */


    results[NumDevices] = 0xFFFF;
   /* NOADDR (0xffff) terminates address list */

    pb->addr = results;
    Control(refNum,FC_devclearlist,&pb);

    printf("\n ibsta: %x",statusBlk.ibsta);
    if(statusBlk.ibsta & 0x8000)
      printf("\n iberr: %d",statusBlk.iberr);
    else
      printf("\n");

/* TriggerList */


    pb->addr = results;
    Control(refNum,FC_triggerlist,&pb);

    printf("\n ibsta: %x",statusBlk.ibsta);
    if(statusBlk.ibsta & 0x8000)
      printf("\n iberr: %d",statusBlk.iberr);
    else
      printf("\n");
```

```
/* Receive */


    for(i=0;i!=NumDevices;i++)  /* Receive 5 Bytes from Each Device */
    {
      pb->IOBufPtr = buffer;
      pb->addr = &results[i];
      pb->IOCount = 5L;
      pb->controlVar = 0x0001;    /* EOI is End (eotmode) */
      Control(refNum, FC_receive, &pb);

      printf("\n ibsta: %x",statusBlk.ibsta);
      if(statusBlk.ibsta & 0x8000)
        printf("\n iberr: %d",statusBlk.iberr);
      else
        printf("\n");
    }


/* Send */


    for(i=0;i!=NumDevices;i++)    /* Send 5 Bytes to Each Device  */
    {
      pb->IOBufPtr = buffer;
      pb->addr = &results[i];
      pb->IOCount = 5L;
      pb->controlVar = 0x0002;      /* EOI is End (eotmode) */
      Control(refNum, FC_send, &pb);

      printf("\n ibsta: %x",statusBlk.ibsta);
      if(statusBlk.ibsta & 0x8000)
        printf("\n iberr: %d",statusBlk.iberr);
      else
        printf("\n");
    }


}
```

Appendix A
Multiline Interface Messages

This appendix contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions. These multiline interface messages are sent and received with ATN TRUE.

For more information on these messages, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*.

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US	CFE	3F	077	63	?	UNL

Message Definitions

CFE†	Configuration Enable
CFG†	Configure
DCL	Device Clear
GET	Group Execute Trigger
GTL	Go To Local
LLO	Local Lockout
MLA	My Listen Address
MSA	My Secondary Address
MTA	My Talk Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable

†This multiline interface message is a proposed extension to the IEEE 488.1 specification to support the HS488 high-speed protocol.

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE,CFG1
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE, CFG2
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE,CFG3
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE,CFG4
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE,CFG5
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE,CFG6
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE,CFG7
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE,CFG8
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE,CFG9
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE,CFG10
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE,CFG11
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE,CFG12
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE,CFG13
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE,CFG14
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE,CFG15
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

PPE	Parallel Poll Enable	SPE	Serial Poll Enable
PPU	Parallel Poll Unconfigure	TCT	Take Control
SDC	Selected Device Clear	UNL	Unlisten
SPD	Serial Poll Disable	UNT	Untalk

Appendix B
Status Word Conditions

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

For information about how to use `ibsta` in your application program, refer to Chapter 2, *Developing Your Application*, in the *NI-488.2 User Manual for Macintosh*.

If a function call returns an ENEB or EDVR error, all status word bits except the ERR bit are cleared, indicating that it is not possible to obtain the status of the GPIB board.

Each bit in `ibsta` can be set for NI-488 device calls (dev), NI-488 board calls and NI-488.2 calls (brd), or both (dev, brd).

The following table lists the status word bits.

Table B-1. Status Word Bits

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

ERR (dev, brd)

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

TIMO (dev, brd)

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

END (dev, brd)

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

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

SRQI (brd)

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

RQS (dev)

RQS appears in the status word only after a device-level call and indicates that the device is requesting service. RQS is set whenever bit 6 is asserted in the serial poll status byte of the device. The serial poll that obtains the status byte can be the result of a call to `ibrsp`, or the poll might be automatic if automatic serial polling is enabled. Do not

issue an `ibwait` on RQS for a device that does not respond to serial polls. RQS is cleared when an `ibrsp` reads the serial poll status byte that caused the RQS.

CMPL (dev, brd)

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

LOK (brd)

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

REM (brd)

REM indicates whether or not the board is in the remote state. REM is set whenever the Remote Enable (REN) GPIB line is asserted and the GPIB board detects that its listen address has been sent either by the GPIB board or by another Controller. REM is cleared in the following situations:

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

CIC (brd)

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

ATN (brd)

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

TACS (brd)

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

LACS (brd)

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

DTAS (brd)

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

DCAS (brd)

DCAS indicates whether the GPIB board has detected a device clear command. DCAS is set whenever the GPIB board detects that the Device Clear (DCL) command has been sent by another Controller, or whenever the GPIB board as a Listener detects that the Selected Device Clear (SDC) command has been sent by another Controller. DCAS is cleared on any call immediately following an `ibwait` call, if the DCAS bit was set in the `ibwait` mask parameter. It also clears on any call immediately following a read or write.

Appendix C
Error Codes and Solutions

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

The following table lists the GPIB error codes.

Table C-1. GPIB Error Codes

Error Mnemonic	`iberr` Value	Meaning
EDVR	0	System error
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	No Listeners on the GPIB
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
EDMA	8	No DMA channel available
EOIP	10	Asynchronous I/O in progress
ECAP	11	No capability for operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial poll status byte queue overflow
ESRQ	16	SRQ stuck in ON position
ETAB	20	Table problem
ELCK	21	Board or device is locked

EDVR (0)

EDVR is returned when the board or device name passed to `ibfind` is not configured in the software.

EDVR is also returned when an invalid unit descriptor is passed to any function call.

EDVR is also returned when the driver is not installed. In this case, `ibcnt` contains a system level error code.

Solutions

- Use `ibdev` to open a device without specifying its symbolic name.
- Use only device or board names that are configured in the utility program `NI-488 Config` as parameters in the `ibfind` function.
- Use the unit descriptor returned from the `ibfind` function as the first parameter in subsequent NI-488 functions. Examine the variable after the `ibfind` and before the failing function to make sure it was not corrupted.
- Make sure the NI-488.2 driver is installed by checking to see if `NI-488 INIT` is in the `Extensions` folder in the `System Folder`.

ECIC (1)

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

- Any device-level NI-488 functions that affect the GPIB
- Any board-level NI-488 functions that issue GPIB command bytes such as `ibcmd`, `ibcmda`, `ibln`, `ibrpp`
- `ibcac`, `ibgts`
- Any of the NI-488.2 routines that issue GPIB command bytes such as `SendCmds`, `PPoll`, `Send`, `Receive`

Solutions

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

ENOL (2)

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

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

Solutions

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

EADR (3)

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

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

Solutions

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

EARG (4)

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

- `ibtmo` called with a value not in the range 0 through 17
- `ibpad` or `ibsad` called with invalid addresses
- `ibppc` called with invalid parallel poll configurations
- A board-level NI-488 call made with a valid device descriptor or a device-level NI-488 call made with a board descriptor
- An NI-488.2 routine called with an invalid address
- `PPollConfig` called with an invalid data line or sense bit

Solutions

- Make sure that the parameters passed to the NI-488 function or NI-488.2 routine are valid.
- Do not use a device descriptor in a board function or vice-versa.

ESAC (5)

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

Solutions

Give the GPIB board System Controller capability by calling `ibrsc 1` or by using `NI-488 Config` to configure that capability into the software.

EABO (6)

EABO indicates that an I/O operation has been canceled, usually due to a timeout condition. Other causes for this error are calling `ibstop` or receiving the Device Clear message from the CIC while performing an I/O operation.

Frequently, the I/O is not progressing (the Listener is not continuing to handshake or the Talker has stopped talking), or the byte count in the call which timed out was more than the other device was expecting.

Solutions

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

ENEB (7)

ENEB occurs when there is no GPIB board present. This happens when the board is not physically plugged into the system, or there is a conflict in the system.

Solutions

Verify that all GPIB interfaces and external controller boxes are plugged in securely, powered on, and configured properly in the GPIB configuration.

EDMA (8)

EDMA occurs when the driver is unable to allocate a DMA channel.

Solutions

Verify that other boards are not using all seven available DMA channels. Disconnect the RTSI connector from the other DMA boards temporarily.

EOIP (10)

EOIP occurs when an asynchronous I/O operation has not finished before some other call is made. During asynchronous I/O, you can only use `ibstop`, `ibwait`, and `ibonl`, or perform other non-GPIB operations. Once the asynchronous I/O has begun, further GPIB calls other than `ibstop`, `ibwait`, or `ibonl` are strictly limited. If a call might interfere with the I/O operation in progress, the driver returns EOIP.

Solutions

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

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

ECAP (11)

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

Solutions

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

EFSO (12)

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

Solutions

- Make sure the file is in the same folder as your application.
- Make sure there is enough room on the disk to hold the file.

EBUS (14)

EBUS results when certain GPIB bus errors occur during device functions. All device functions send command bytes to perform addressing and other bus management. Devices are expected to accept these command bytes within the time limit specified by

the default configuration or the `ibtmo` function. EBUS results if a timeout occurred while sending these command bytes.

Solutions

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

ESTB (15)

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

Solutions

- Call `ibrsp` more frequently to empty the queue.
- Disable autopolling with the `ibconfig` function or the `NI-488 Config` utility.

ESRQ (16)

ESRQ occurs only during the `ibwait` function or the `WaitSRQ` routine. ESRQ indicates that a wait for RQS is not possible because the GPIB SRQ line is stuck on. This situation can be caused by the following events:

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

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

Solutions

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

ETAB (20)

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

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

Solutions

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

ELCK (21)

ELCK occurs if the requested GPIB-ENET board or device is being used through another connection.

Solutions

Wait for the lock on the board or device to be released, or try using `ibunlock` if you previously used `iblock` to lock access to the connection.

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 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	519 622 9311
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 71 11
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 48301892	02 48301915
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Use additional pages if necessary.

Name ____

Company ____

Address ____

____

Fax (___)________________ Phone (___)______________________________

Computer brand ____

Model ___________________________ RAM _____________________MB

Processor ________________________ Speed

_________MHz

Operating system ____

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-488.2™ Function Reference Manual for Macintosh**

Edition Date: **October 1995**

Part Number: **320898A-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 — Listeners use this GPIB interface function to receive data, and all devices use it to receive commands. See *source handshake* and *handshake*.

access board — The GPIB board that controls and communicates with the devices on the bus that are attached to it.

ANSI — American National Standards Institute.

ASCII — American Standard Code for Information Interchange.

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

automatic serial polling (autopolling) — A feature of the NI-488.2 software in which serial polls are executed automatically by the driver whenever a device asserts the GPIB SRQ line.

B

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

C

CFE — Configuration Enable is the GPIB command which precedes CFGn and is used to place devices into their configuration mode.

CFGn	These GPIB commands (CFG1 through CFG15) follow CFE and are used to configure all devices for the number of meters of cable in the system so that HS488 transfers occur without errors.
CIC	See *Controller-In-Charge*.
Controller-In-Charge (CIC)	The device that manages the GPIB by sending interface messages to other devices.
CPU	Central processing unit.

D

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

E

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

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

G

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

H

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

hex	Hexadecimal; a number represented in base 16, for example decimal 16 = hex 10.
high-level function	See *device-level function*.
Hz	Hertz.

I

`ibcnt`	After each NI-488.2 I/O function, this global variable contains the actual number of bytes transmitted.
`iberr`	A global variable that contains the specific error code associated with a function call that failed.
`IBIC 488.2`	`IBIC 488.2`, the Interface Bus Interactive Control utility, is used to communicate with GPIB devices, troubleshoot problems, and develop your application.
`ibsta`	At the end of each function call, this global variable (status word) contains status information.
IEEE	Institute of Electrical and Electronic Engineers.
interface message	A broadcast message sent from the Controller to all devices and used to manage the GPIB.
I/O (Input/Output)	In the context of this manual, the transmission of commands or messages between the computer via the GPIB board and other devices on the GPIB.
I/O address	The address of the GPIB board from the point of view of the CPU, as opposed to the GPIB address of the GPIB board. Also called port address or board address.
`ist`	An Individual Status bit of the status byte used in the Parallel Poll Configure function.

K

KB	Kilobytes.

L

LAD (Listen Address)	See *MLA*.
language interface	Code that enables an application program that uses NI-488 functions or NI-488.2 routines to access the driver.
listen address	See *MLA*.
Listener	A GPIB device that receives data messages from a Talker.
low-level function	See *board-level function*.

M

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

N

NDAC (Not Data Accepted)	One of the three GPIB handshake lines. See *handshake*.

`NI-488 Config`	The NI-488.2 driver configuration control panel utility.
NRFD (Not Ready For Data)	One of the three GPIB handshake lines. See *handshake*.

P

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

R

RAM	Random-access memory.
resynchronize	The NI-488.2 software and the user application must resynchronize after asynchronous I/O operations have completed.
RQS	Request Service.

S

s	Seconds.

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

T

TAD (Talk Address)	See *MTA*.
Talker	A GPIB device that sends data messages to Listeners.
TCT	Take Control is the GPIB command used to pass control of the bus from the current Controller to an addressed Talker.

timeout	A feature of the NI-488.2 driver that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.
TLC	An integrated circuit that implements most of the GPIB Talker, Listener, and Controller functions in hardware.

U

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

Index

A

access board of device, changing. *See* IBBNA function.
active controller. *See* IBCAC function; IBGTS function.
address configuration functions. *See* IBPAD function; IBSAD function.
AllSpoll routine
    Device Manager, 3-101
    NI-488.2, 2-4 to 2-5
asynchronous operation, aborting. *See* IBSTOP function.
ATN status word condition, B-3

B

become active controller. *See* IBCAC function.
board configuration parameter options
    IBASK function (table), 1-7 to 1-10
    IBCONFIG function (table), 1-25 to 1-28

C

CIC status word condition, B-3
clear functions/routines
    DevClear routine
        Device Manager, 3-102
        NI-488.2, 2-6
    DevClearList routine
        Device Manager, 3-103
        NI-488.2, 2-7 to 2-8
    IBCLR function
        Device Manager, 3-9
        NI-488, 1-17 to 1-18
    IBSIC
        Device Manager, 3-66
        NI-488, 1-80 to 1-81
    IBSRE
        Device Manager, 3-67 to 3-68
        NI-488, 1-82 to 1-83
    SendIFC
        Device Manager, 3-121
        NI-488.2, 2-40
CMPL status word condition, B-3

command functions/routines
IBCMD function
Device Manager, 3-10 to 3-13
NI-488, 1-19 to 1-20
IBCMDA function
Device Manager, 3-14 to 3-15
NI-488, 1-21 to 1-22
SendCmds routine
Device Manager, 3-119
NI-488.2, 2-36 to 2-37
common errors and solutions. *See* error codes and solutions.
configuration functions. *See* IBASK function; IBCONFIG function.
configuration options. *See* board configuration parameter options; device configuration parameter options.
control line function. *See* IBLINES function.
controller functions/routines
IBCAC function
Device Manager, 3-7 to 3-8
NI-488, 1-15 to 1-16
IBGTS function
Device Manager, 3-28 to 3-29
NI-488, 1-42 to 1-43
IBPCT function
Device Manager, 3-46
NI-488, 1-59 to 1-60
IBRSC function
Device Manager, 3-59
NI-488, 1-72 to 1-73
PassControl routine
Device Manager, 3-109
NI-488.2, 2-17
customer communication, *xiii,* D-1

D

DCAS status word condition, B-4
DevClear routine
Device Manager, 3-102
NI-488.2, 2-6
DevClearList routine
Device Manager, 3-103
NI-488.2, 2-7 to 2-8
device configuration parameter options
IBASK function (table), 1-11 to 1-12
IBCONFIG function (table), 1-29 to 1-30
device descriptor, opening and initializing. *See* IBDEV function.

Device Manager functions. *See also* NI-488 functions.
- IBASK, 3-4 to 3-5
- IBBNA, 3-6
- IBCAC, 3-7 to 3-8
- IBCLR, 3-9
- IBCMD, 3-10 to 3-13
- IBCMDA, 3-14 to 3-15
- IBCONFIG, 3-16
- IBDEV, 3-17 to 3-18
- IBDMA, 3-19
- IBEOS, 3-20 to 3-23
- IBEOT, 3-24 to 3-25
- IBFIND, 3-26 to 3-27
- IBGTS, 3-28 to 3-29
- IBIST, 3-30 to 3-31
- IBLINES, 3-32 to 3-33
- IBLLO, 3-34
- IBLN, 3-35 to 3-36
- IBLOC, 3-37 to 3-38
- IBLOCK, 3-39 to 3-40
- IBONL, 3-41 to 3-43
- IBPAD, 3-44 to 3-45
- IBPCT, 3-46
- IBPPC, 3-47 to 3-49
- IBRD, 3-50 to 3-52
- IBRDA, 3-53 to 3-56
- IBRPP, 3-57 to 3-58
- IBRSC, 3-59
- IBRSP, 3-60 to 3-61
- IBRSV, 3-62 to 3-63
- IBSAD, 3-64 to 3-65
- IBSIC, 3-66
- IBSRE, 3-67 to 3-68
- IBSTOP, 3-69 to 3-70
- IBTMO, 3-71 to 3-74
- IBTRG, 3-75
- IBUNLOCK, 3-76 to 3-77
- IBWAIT, 3-78 to 3-81
- IBWRT, 3-82 to 3-84
- IBWRTA, 3-85 to 3-87
- list of control calls (table), 3-2 to 3-3

Device Manager programming examples
- accessing GPIB driver from THINK Pascal, 3-93 to 3-98
- high-level device manager calls, 3-88 to 3-90
- low-level device manager calls, 3-90 to 3-92
- NI-488.2 routines, 3-131 to 3-134

Device Manager routines. *See also* NI-488.2 routines.
AllSpoll, 3-101
DevClear, 3-102
DevClearList, 3-103
EnableLocal, 3-104
EnableRemote, 3-105
FindLstn, 3-106 to 3-107
FindRQS, 3-108
list of routines (table), 3-99 to 3-100
PassControl, 3-109
PPoll, 3-110
PPollConfig, 3-111
PPollUnconfig, 3-112
RcvRespMsg, 3-113
ReadStatusByte, 3-114
Receive, 3-115
ReceiveSetup, 3-116
ResetSys, 3-117
Send, 3-118
SendCmds, 3-119
SendDataBytes, 3-120
SendIFC, 3-121
SendList, 3-122
SendLLO, 3-123
SendRWLS, 3-125
SendSetup, 3-124
TestSRQ, 3-126
TestSys, 3-127
Trigger, 3-128
TriggerList, 3-129
WaitSRQ, 3-130
DMA function. *See* IBDMA function.
documentation
conventions used, *xii-xiii*
how to use manual set, *xi*
organization of manual, *xii*
related documentation, *xiii*
DTAS status word condition, B-4

E

EABO error code, C-4 to C-5
EADR error code, C-3
EARG error code, C-4
EBUS error code, C-6 to C-7
ECAP error code, C-6
ECIC error code, C-2

EDMA error code, C-5
EDVR error code, C-1 to C-2
EFSO error code, C-6
ELCK error code, C-8
EnableLocal routine
    Device Manager, 3-104
    NI-488.2, 2-9 to 2-10
EnableRemote routine
    Device Manager, 3-105
    NI-488.2, 2-11 to 2-12
END message. *See* IBEOT function.
END status word condition, B-2
ENEB error code, C-5
ENOL error code, C-3
EOI line, enabling or disabling. *See* IBEOT function.
EOIP error code, C-5 to C-6
EOS modes, configuring. *See* IBEOS function.
ERR status word condition, B-2
error codes and solutions
    EABO, C-4 to C-5
    EADR, C-3
    EARG, C-4
    EBUS, C-6 to C-7
    ECAP, C-6
    ECIC, C-2
    EDMA, C-5
    EDVR, C-1 to C-2
    EFSO, C-6
    ELCK, C-8
    ENEB, C-5
    ENOL, C-3
    EOIP, C-5 to C-6
    ESAC, C-4
    ESRQ, C-7 to C-8
    ESTB, C-7
    ETAB, C-8
    list of error codes (table), C-1
ESAC error code, C-4
ESRQ error code, C-7 to C-8
ESTB error code, C-7
ETAB error code, C-8

F

finding GPIB board or device. *See* IBFIND function.
FindLstn routine
    Device Manager, 3-106 to 3-107
    NI-488.2, 2-13 to 2-14
FindRQS routine
    Device Manager, 3-108
    NI-488.2, 2-15 to 2-16
functions. *See* Device Manager functions; NI-488 functions.

G

GPIB address, configuring. *See* IBPAD function; IBSAD function.

I

IbaAUTOPOLL configuration option parameter (table), 1-7
IbaBaud configuration option parameter (table), 1-10
IbaBNA configuration option parameter (table), 1-12
IbaCICPROT configuration option parameter (table), 1-7
IbaComPort configuration option parameter (table), 1-10
IbaDataBits configuration option parameter (table), 1-10
IbaDMA configuration option parameter (table), 1-9
IbaEndBitIsNormal configuration option parameter (table)
    boards, 1-9
    devices, 1-12
IbaEOSchar configuration option parameter (table)
    boards, 1-8
    devices, 1-12
IbaEOScmp configuration option parameter (table)
    boards, 1-8
    devices, 1-12
IbaEOSrd configuration option parameter (table)
    boards, 1-8
    devices, 1-11
IbaEOSwrt configuration option parameter (table)
    boards, 1-8
    devices, 1-12
IbaEOT configuration option parameter (table)
    boards, 1-7
    devices, 1-11
IbaHSCableLength configuration option parameter (table), 1-9
IbaIst configuration option parameter (table), 1-9

IbaPAD configuration option parameter (table)
 boards, 1-7
 devices, 1-11
IbaParity configuration option parameter (table), 1-10
IbaPP2 configuration option parameter (table), 1-8
IbaPPC configuration option parameter (table), 1-7
IbaReadAdjust configuration option parameter (table)
 boards, 1-9
 devices, 1-12
IbaRsv configuration option parameter (table), 1-10
IbaSAD configuration option parameter (table)
 boards, 1-7
 devices, 1-11
IbaSC configuration option parameter (table), 1-7
IBASK function
 Device Manager, 3-4 to 3-5
 NI-488
  board configuration parameter options (table), 1-7 to 1-10
  description, 1-5
  device configuration parameter options (table), 1-11 to 1-12
  possible errors, 1-6
IbaSRE configuration option parameter (table), 1-8
IbaStopBits configuration option parameter (table), 1-10
IbaTIMING configuration option parameter (table), 1-9
IbaTMO configuration option parameter (table)
 boards, 1-7
 devices, 1-11
IbaWriteAdjust configuration option parameter (table)
 boards, 1-9
 devices, 1-12
IBBNA function
 Device Manager, 3-6
 NI-488, 1-13 to 1-14
IBCAC function
 Device Manager, 3-7 to 3-8
 NI-488, 1-15 to 1-16
IbcAUTOPOLL configuration parameter option (table), 1-25
IbcCICPROT configuration parameter option (table), 1-26
IbcDMA configuration parameter option (table), 1-27
IbcEndBitIsNormal configuration parameter option (table)
 boards, 1-28
 devices, 1-30
IbcEOSchar configuration parameter option (table)
 boards, 1-26
 devices, 1-30
IbcEOScmp configuration parameter option (table)
 boards, 1-26
 devices, 1-30

IbcEOSrd configuration parameter option (table)
 boards, 1-26
 devices, 1-29
IbcEOSwrt configuration parameter option (table)
 boards, 1-26
 devices, 1-30
IbcEOT configuration parameter option (table)
 boards, 1-25
 devices, 1-29
IbcHSCableLength configuration parameter option (table), 1-28
IBCLR function
 Device Manager, 3-9
 NI-488, 1-17 to 1-18
IBCMD function
 Device Manager, 3-10 to 3-13
 NI-488, 1-19 to 1-20
IBCMDA function
 Device Manager, 3-14 to 3-15
 NI-488, 1-21 to 1-22
IBCONFIG function
 Device Manager, 3-16
 NI-488
  board configuration parameter options (table), 1-25 to 1-28
  description, 1-23
  device configuration parameter options (table), 1-29 to 1-30
  possible errors, 1-24
IbcPAD configuration parameter option (table)
 boards, 1-25
 devices, 1-29
IbcPP2 configuration parameter option (table), 1-27
IbcPPC configuration parameter option (table), 1-25
IbcPPollTime configuration parameter option (table), 1-28
IbcReadAdjust configuration parameter option (table), 1-27
IbcREADDR configuration parameter option (table), 1-25
IbcSAD configuration parameter option (table)
 boards, 1-25
 devices, 1-29
IbcSendLLO configuration parameter option (table), 1-27
IbcSRE configuration parameter option (table), 1-26
IbcTIMING configuration parameter option (table), 1-27
IbcTMO configuration parameter option (table)
 boards, 1-25
 devices, 1-29
IbcUnAddr configuration parameter option (table), 1-28
IbcWriteAdjust configuration parameter option (table)
 boards, 1-27
 devices, 1-30

IBDEV function
  Device Manager, 3-17 to 3-18
  NI-488, 1-31 to 1-32
IBDMA function
  Device Manager, 3-19
  NI-488, 1-33 to 1-34
IBEOS function
  Device Manager, 3-20 to 3-23
  NI-488, 1-35 to 1-37
IBEOT function
  Device Manager, 3-24 to 3-25
  NI-488, 1-38 to 1-39
IBFIND function
  Device Manager, 3-26 to 3-27
  NI-488, 1-40 to 1-41
IBGTS function
  Device Manager, 3-28 to 3-29
  NI-488, 1-42 to 1-43
IBIST function
  Device Manager, 3-30 to 3-31
  NI-488, 1-44 to 1-45
IBLINES function
  Device Manager, 3-32 to 3-33
  NI-488, 1-46 to 1-47
IBLLO function
  Device Manager, 3-34
  NI-488, 1-48
IBLN function
  Device Manager, 3-35 to 3-36
  NI-488, 1-49 to 1-50
IBLOC function
  Device Manager, 3-37 to 3-38
  NI-488, 1-51 to 1-52
IBLOCK function
  Device Manager, 3-39 to 3-40
  NI-488, 1-53 to 1-54
IBONL function
  Device Manager, 3-41 to 3-43
  NI-488, 1-55 to 1-56
IBPAD function
  Device Manager, 3-44 to 3-45
  NI-488, 1-57 to 1-58
IBPCT function
  Device Manager, 3-46
  NI-488, 1-59 to 1-60
IBPPC function
  Device Manager, 3-47 to 3-49
  NI-488, 1-61 to 1-62

IBRD function
    Device Manager, 3-50 to 3-52
    NI-488, 1-63 to 1-64
IBRDA function
    Device Manager, 3-53 to 3-56
    NI-488, 1-65 to 1-67
IBRDF function, NI-488, 1-68 to 1-69
IBRPP function
    Device Manager, 3-57 to 3-58
    NI-488, 1-70 to 1-71
IBRSC function
    Device Manager, 3-59
    NI-488, 1-72 to 1-73
IBRSP function
    Device Manager, 3-60 to 3-61
    NI-488, 1-74 to 1-75
IBRSV function
    Device Manager, 3-62 to 3-63
    NI-488, 1-76 to 1-77
IBSAD function
    Device Manager, 3-64 to 3-65
    NI-488, 1-78 to 1-79
IBSIC function
    Device Manager, 3-66
    NI-488, 1-80 to 1-81
IBSRE function
    Device Manager, 3-67 to 3-68
    NI-488, 1-82 to 1-83
IBSRQ function, NI-488, 1-84
IBSTOP function
    Device Manager, 3-69 to 3-70
    NI-488, 1-85
IBTMO function
    Device Manager, 3-71 to 3-74
    NI-488, 1-86 to 1-87
IBTRG function
    Device Manager, 3-75
    NI-488, 1-88 to 1-89
IBUNLOCK function
    Device Manager, 3-76 to 3-77
    NI-488, 1-90 to 1-91
IBWAIT function
    Device Manager, 3-78 to 3-81
    NI-488, 1-92 to 1-93
IBWRT function
    Device Manager, 3-82 to 3-84
    NI-488, 1-94 to 1-95

IBWRTA function
  Device Manager, 3-85 to 3-87
  NI-488, 1-96 to 1-98
IBWRTF function, NI-488, 1-99 to 1-100
individual status bit, setting or clearing. *See* IBIST function.
interface clear functions/routines
  IBSIC function
    Device Manager, 3-66
    NI-488, 1-80 to 1-81
  SendIFC routine
    Device Manager, 3-121
    NI-488.2, 2-40
interface messages, multiline, A-1 to A-3

L

LACS status word condition, B-4
Listener functions/routines
  FindLstn routine
    Device Manager, 3-106 to 3-107
    NI-488.2, 2-13 to 2-14
  IBLN function
    Device Manager, 3-35 to 3-36
    NI-488, 1-49 to 1-50
  ReceiveSetup routine
    Device Manager, 3-116
    NI-488.2, 2-30 to 2-31
local functions/routines
  EnableLocal
    Device Manager, 3-104
    NI-488.2, 2-9 to 2-10
  IBLOC function
    Device Manager, 3-37 to 3-38
    NI-488, 1-51 to 1-52
locking access to GPIB-ENET board or device. *See* IBLOCK function.
lockout functions/routines
  IBLLO function
    Device Manager, 3-34
    NI-488, 1-48
  SendLLO routine
    Device Manager, 3-123
    NI-488.2, 2-43 to 2-44
  SendRWLS routine, 3-125
  SetRWLS routine, 2-47 to 2-48
LOK status word condition, B-3

M

manual. *See* documentation.
messages, multiline interface, A-1 to A-3

N

NI-488 functions. *See also* Device Manager functions.
IBASK, 1-5 to 1-12
IBBNA, 1-13 to 1-14
IBCAC, 1-15 to 1-16
IBCLR, 1-17 to 1-18
IBCMD, 1-19 to 1-20
IBCMDA, 1-21 to 1-22
IBCONFIG, 1-23 to 1-30
IBDEV, 1-31 to 1-32
IBDMA, 1-33 to 1-34
IBEOS, 1-35 to 1-37
IBEOT, 1-38 to 1-39
IBFIND, 1-40 to 1-41
IBGTS, 1-42 to 1-43
IBIST, 1-44 to 1-45
IBLINES, 1-46 to 1-47
IBLLO, 1-48
IBLN, 1-49 to 1-50
IBLOC, 1-51 to 1-52
IBLOCK, 1-53 to 1-54
IBONL, 1-55 to 1-56
IBPAD, 1-57 to 1-58
IBPCT, 1-59 to 1-60
IBPPC, 1-61 to 1-62
IBRD, 1-63 to 1-64
IBRDA, 1-65 to 1-67
IBRDF, 1-68 to 1-69
IBRPP, 1-70 to 1-71
IBRSC, 1-72 to 1-73
IBRSP, 1-74 to 1-75
IBRSV, 1-76 to 1-77
IBSAD, 1-78 to 1-79
IBSIC, 1-80 to 1-81
IBSRE, 1-82 to 1-83
IBSRQ, 1-84
IBSTOP, 1-85
IBTMO, 1-86 to 1-87
IBTRG, 1-88 to 1-89
IBUNLOCK, 1-90 to 1-91
IBWAIT, 1-92 to 1-93

(continued)
IBWRT, 1-94 to 1-95
IBWRTA, 1-96 to 1-98
IBWRTF, 1-99 to 1-100
list of functions
board-level functions (table), 1-3 to 1-4
device-level functions (table), 1-2 to 1-3
NI-488.2 routines. *See also* Device Manager routines.
AllSpoll, 2-4 to 2-5
DevClear, 2-6
DevClearList, 2-7 to 2-8
EnableLocal, 2-9 to 2-10
EnableRemote, 2-11 to 2-12
FindLstn, 2-13 to 2-14
FindRQS, 2-15 to 2-16
list of routines (table), 2-2 to 2-3
PassControl, 2-17
PPoll, 2-18 to 2-19
PPollConfig, 2-20 to 2-21
PPollUnconfig, 2-22 to 2-23
RcvRespMsg, 2-24 to 2-25
ReadStatusByte, 2-26 to 2-27
Receive, 2-28 to 2-29
ReceiveSetup, 2-30 to 2-31
ResetSys, 2-32 to 2-33
Send, 2-34 to 2-35
SendCmds, 2-36 to 2-37
SendDataBytes, 2-38 to 2-39
SendIFC, 2-40
SendList, 2-41 to 2-42
SendLLO, 2-43 to 2-44
SendSetup, 2-45 to 2-46
SetRWLS, 2-47 to 2-48
TestSRQ, 2-49
TestSys, 2-50 to 2-51
Trigger, 2-52
TriggerList, 2-53 to 2-54
WaitSRQ, 2-55

O

online or offline device function. *See* IBONL function.

P

parallel polling functions/routines
- IBIST function
  - Device Manager, 3-30 to 3-31
  - NI-488, 1-44 to 1-45
- IBPPC function
  - Device Manager, 3-47 to 3-49
  - NI-488, 1-61 to 1-62
- IBRPP function
  - Device Manager, 3-57 to 3-58
  - NI-488, 1-70 to 1-71
- PPoll
  - Device Manager, 3-110
  - NI-488.2, 2-18 to 2-19
- PPollConfig
  - Device Manager, 3-111
  - NI-488.2, 2-20 to 2-21
- PPollUnconfig
  - Device Manager, 3-112
  - NI-488.2, 2-22 to 2-23

pass control functions/routines
- IBPCT function
  - Device Manager, 3-46
  - NI-488, 1-59 to 1-60
- PassControl routine
  - Device Manager, 3-109
  - NI-488.2, 2-17

PPoll routine
- Device Manager, 3-110
- NI-488.2, 2-18 to 2-19

PPollConfig routine
- Device Manager, 3-111
- NI-488.2, 2-20 to 2-21

PPollUnconfig routine
- Device Manager, 3-112
- NI-488.2, 2-22 to 2-23

primary GPIB address, configuring. *See* IBPAD function.

programming examples. *See* Device Manager programming examples.

R

RcvRespMsg routine
- Device Manager, 3-113
- NI-488.2, 2-24 to 2-25

read and write termination. *See* IBEOS function; IBEOT function.

read functions/routines
IBRD
Device Manager, 3-50 to 3-52
NI-488, 1-63 to 1-64
IBRDA
Device Manager, 3-53 to 3-56
NI-488, 1-65 to 1-67
IBRDF, 1-68 to 1-69
RcvRespMsg routine
Device Manager, 3-113
NI-488.2, 2-24 to 2-25
ReadStatusByte routine
Device Manager, 3-114
NI-488.2, 2-26 to 2-27
Receive routine
Device Manager, 3-115
NI-488.2, 2-28 to 2-29
ReadStatusByte routine
Device Manager, 3-114
NI-488.2, 2-26 to 2-27
Receive routine
Device Manager, 3-115
NI-488.2, 2-28 to 2-29
ReceiveSetup routine
Device Manager, 3-116
NI-488.2, 2-30 to 2-31
REM status word condition, B-3
remote functions/routines
EnableRemote routine
Device Manager, 3-105
NI-488.2, 2-11 to 2-12
IBSRE function
Device Manager, 3-67 to 3-68
NI-488, 1-82 to 1-83
SendRWLS routine, 3-125
SetRWLS routine, 2-47 to 2-48
request for service. *See* SRQ functions/routines.
ResetSys routine
Device Manager, 3-117
NI-488.2, 2-32 to 2-33
routines. *See* Device Manager routines; NI-488.2 routines.
RQS status word condition, B-2 to B-3

S

secondary GPIB address, configuring. *See* IBSAD function.
Send routine
    Device Manager, 3-118
    NI-488.2, 2-34 to 2-35
SendCmds routine
    Device Manager, 3-119
    NI-488.2, 2-36 to 2-37
SendDataBytes routine
    Device Manager, 3-120
    NI-488.2, 2-38 to 2-39
SendIFC routine
    Device Manager, 3-121
    NI-488.2, 2-40
SendList routine
    Device Manager, 3-122
    NI-488.2, 2-41 to 2-42
SendLLO routine
    Device Manager, 3-123
    NI-488.2, 2-43 to 2-44
SendRWLS routine, Device Manager, 3-125
SendSetup routine
    Device Manager, 3-124
    NI-488.2, 2-45 to 2-46
serial polling functions/routines
    AllSpoll routine
        Device Manager, 3-101
        NI-488.2, 2-4 to 2-5
    IBRSP function
        Device Manager, 3-60 to 3-61
        NI-488, 1-74 to 1-75
    IBRSV function
        Device Manager, 3-62 to 3-63
        NI-488, 1-76 to 1-77
    ReadStatusByte routine
        Device Manager, 3-114
        NI-488.2, 2-26 to 2-27
service request functions. *See* SRQ functions/routines.
SetRWLS routine, NI-488.2, 2-47 to 2-48
SRQ functions/routines
    FindRQS routine
        Device Manager, 3-108
        NI-488.2, 2-15 to 2-16
    IBSRQ function, 1-84
    TestSRQ routine
        Device Manager, 3-126
        NI-488.2, 2-49

(continued)
WaitSRQ routine
Device Manager, 3-130
NI-488.2, 2-55
SRQI status word condition, B-2
status word conditions
ATN, B-3
CIC, B-3
CMPL, B-3
DCAS, B-4
DTAS, B-4
END, B-2
ERR, B-2
LACS, B-4
list of status word bits (table), B-1
LOK, B-3
REM, B-3
RQS, B-2 to B-3
SRQI, B-2
TACS, B-4
TIMO, B-2
system control function. *See* IBRSC function.

T

TACS status word condition, B-4
Talker routines. *See* ReceiveSetup routine.
technical support, D-1
TestSRQ routine
Device Manager, 3-126
NI-488.2, 2-49
TestSys routine
Device Manager, 3-127
NI-488.2, 2-50 to 2-51
timeouts. *See* IBTMO function.
TIMO status word condition, B-2
trigger functions/routines
IBTRG function
Device Manager, 3-75
NI-488, 1-88 to 1-89
Trigger routine
Device Manager, 3-128
NI-488.2, 2-52
TriggerList routine
Device Manager, 3-129
NI-488.2, 2-53 to 2-54

U

unlocking access to GPIB-ENET board or device. *See* IBUNLOCK function.

W

wait functions/routines
  IBWAIT function
    Device Manager, 3-78 to 3-81
    NI-488, 1-92 to 1-93
  WaitSRQ routine
    Device Manager, 3-130
    NI-488.2, 2-55
write functions/routines
  IBWRT function
    Device Manager, 3-82 to 3-84
    NI-488, 1-94 to 1-95
  IBWRTA function
    Device Manager, 3-85 to 3-87
    NI-488, 1-96 to 1-98
  IBWRTF function, 1-99 to 1-100