Important Information

Warranty

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

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

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

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

Copyright

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

HS488™, natinst.com™, and NI-488.2™ are trademarks of National Instruments Corporation. Product and company names mentioned herein are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

About This Manual
Using the NI-488.2 Documentation.................................................................xi
Accessing the NI-488.2 Online Help...............................................................xi
Conventions Used in This Manual.................................................................xii
Related Documentation.................................................................................xii

Chapter 1
NI-488.2 Traditional Calls
List of Traditional Calls..................................................................................1-2
Device-Level Calls.........................................................................................1-2
Board-Level Calls.........................................................................................1-3
IBASK.............................................................................................................1-5
IBBNA.............................................................................................................1-11
IBCAC.............................................................................................................1-12
IBCLR.............................................................................................................1-14
IBCMD............................................................................................................1-15
IBCMDA.........................................................................................................1-17
IBCONFIG......................................................................................................1-19
IBDEV.............................................................................................................1-25
IBDMA.............................................................................................................1-27
IBEOS.............................................................................................................1-28
IBEOT.............................................................................................................1-30
IBFINF..........................................................................................................1-31
IBGTS.............................................................................................................1-33
IBIST..............................................................................................................1-35
IBLINES........................................................................................................1-36
IBLN...............................................................................................................1-38
IBLOC.............................................................................................................1-40
IBNOTIFY...................................................................................................1-42
IBONL.............................................................................................................1-46
IBPAD.............................................................................................................1-47
IBPCT.............................................................................................................1-48
IBPPC.............................................................................................................1-49
IBRD................................................................................................................1-51
IBRDA.............................................................................................................1-53
IBRDF.............................................................................................................1-55
IBRPP.............................................................................................................1-57
IBRSC.............................................................................................................1-58
IBRSP.............................................................................................................1-59
IBRSV.............................................................................................................1-61
Contents

IBSAD ............................................................................................................. 1-62
IBSIC .............................................................................................................. 1-63
IBSRE ............................................................................................................. 1-64
IBSTOP ........................................................................................................... 1-65
IBTMO ............................................................................................................ 1-66
IBTRG ............................................................................................................. 1-68
IBWAIT .......................................................................................................... 1-69
IBWRT ............................................................................................................ 1-71
IBWRTA ......................................................................................................... 1-73
IBWRTF .......................................................................................................... 1-75

Chapter 2
NI-488.2 Multi-Device Calls

List of Multi-Device Calls .............................................................................. 2-2
AllSpoll ........................................................................................................... 2-4
DevClear ......................................................................................................... 2-5
DevClearList ................................................................................................... 2-6
EnableLocal ..................................................................................................... 2-7
EnableRemote .................................................................................................. 2-8
FindLstn ......................................................................................................... 2-9
FindRQS ......................................................................................................... 2-10
PassControl .................................................................................................... 2-11
PPoll ................................................................................................................ 2-12
PPollConfig ..................................................................................................... 2-13
PPollUnconfig .................................................................................................. 2-15
RcvRespMsg ................................................................................................... 2-16
ReadStatusByte ............................................................................................... 2-18
Receive ............................................................................................................ 2-19
ReceiveSetup ................................................................................................... 2-21
ResetSys .......................................................................................................... 2-22
Send ................................................................................................................. 2-23
SendCmds ....................................................................................................... 2-25
SendDataBytes ................................................................................................ 2-26
SendIFC .......................................................................................................... 2-28
SendList .......................................................................................................... 2-29
SendLLO ......................................................................................................... 2-31
SendSetup ....................................................................................................... 2-32
SetRWLS .................................................................................................... 2-33
TestSRQ ........................................................................................................... 2-34
TestSys .......................................................................................................... 2-35
Trigger ........................................................................................................... 2-37
TriggerList ...................................................................................................... 2-38
WaitSRQ ......................................................................................................... 2-39
Chapter 3
Supplemental Calls for Multithreaded Applications

List of Supplemental Calls

ThreadIbcnt .......................................................... 3-3
ThreadIbcntl ....................................................... 3-4
ThreadIberr .......................................................... 3-5
ThreadIbsta .......................................................... 3-6

Appendix A
Multiline Interface Messages

Appendix B
Status Word Conditions

ERR (dev, brd) .......................................................... B-2
TIMO (dev, brd) ....................................................... B-2
END (dev, brd) ......................................................... B-2
SRQI (brd) .............................................................. B-3
RQS (dev) .............................................................. B-3
CMPL (dev, brd) ...................................................... B-3
LOK (brd) .............................................................. B-3
REM (brd) .............................................................. B-4
CIC (brd) .............................................................. B-4
ATN (brd) ............................................................. B-4
TACS (brd) ........................................................... B-4
LACS (brd) ........................................................... B-5
DTAS (brd) ........................................................... B-5
DCAS (brd) ........................................................... B-5
Appendix C
Error Codes and Solutions

EDVR (0)....................................................................................................................... C-2
ECIC (1)....................................................................................................................... C-2
ENOL (2)....................................................................................................................... C-3
EADR (3)....................................................................................................................... C-4
EARG (4)....................................................................................................................... C-4
ESAC (5)....................................................................................................................... C-5
EABO (6)....................................................................................................................... C-5
ENEB (7)....................................................................................................................... C-5
EDMA (8)....................................................................................................................... C-6
EOIP (10)...................................................................................................................... C-6
ECAP (11)..................................................................................................................... C-7
EFSO (12)..................................................................................................................... C-7
EBUS (14)..................................................................................................................... C-8
ESTB (15)..................................................................................................................... C-8
ESRQ (16)..................................................................................................................... C-9
ETAB (20)..................................................................................................................... C-9

Appendix D
Technical Support Resources

Glossary

Index
Tables

Table 1-1. Traditional Calls: Device-Level ...........................................................1-2
Table 1-2. Traditional Calls: Board-Level .............................................................1-3
Table 1-3. ibask Board Configuration Parameter Options ......................................1-7
Table 1-4. ibask Device Configuration Parameter Options ....................................1-10
Table 1-5. ibconfig Board Configuration Parameter Options ..............................1-21
Table 1-6. ibconfig Device Configuration Parameter Options ..............................1-24
Table 1-7. EOS Configurations ..............................................................................1-30
Table 1-8. Notify Mask Layout .............................................................................1-44
Table 1-9. Timeout Code Values ...........................................................................1-67
Table 1-10. Wait Mask Layout ..............................................................................1-71

Table 2-1. NI-488.2 API: Multi-Device Calls .......................................................2-2

Table 3-1. Supplemental Calls for Multithreaded Applications ............................3-2

Table A-1. Multiline Interface Messages ..............................................................A-2
About This Manual

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

Using the NI-488.2 Documentation

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

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

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

Accessing the NI-488.2 Online Help

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

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

This manual uses the following conventions:

»

The » symbol leads you through nested menu items and dialog box options to a final action. The sequence File » Page Setup » Options directs you to pull down the File menu, select the Page Setup item, and select Options from the last dialog box.

This icon denotes a note, which alerts you to important information.

bold

Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows tabs, or LEDs.

IEEE 488.1 and IEEE 488.2

IEEE 488 and IEEE 488.2 refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.

italic

Italic text denotes emphasis, a cross reference, or an introduction to a key concept.

monospace

Text in this font denotes text or characters that are to be literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, constants, variables, filenames, and extensions, and for statements and comments taken from program code.

Related Documentation

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

- Microsoft Platform Software Development Kit (SDK)
NI-488.2 Traditional Calls

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

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

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

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

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

Device-Level Calls

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibask</td>
<td>Return information about software configuration parameters.</td>
</tr>
<tr>
<td>ibbna</td>
<td>Change the access interface of a device.</td>
</tr>
<tr>
<td>ibclr</td>
<td>Clear a specific device.</td>
</tr>
<tr>
<td>ibconfig</td>
<td>Change the software configuration parameters.</td>
</tr>
<tr>
<td>ibdev</td>
<td>Open and initialize a device descriptor.</td>
</tr>
<tr>
<td>ibeos</td>
<td>Configure the end-of-string (EOS) termination mode or character.</td>
</tr>
<tr>
<td>ibeot</td>
<td>Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations.</td>
</tr>
<tr>
<td>ibln</td>
<td>Check for the presence of a device on the bus.</td>
</tr>
<tr>
<td>ibloc</td>
<td>Go to Local.</td>
</tr>
<tr>
<td>ibnotify</td>
<td>Notify user of one or more GPIB events by invoking the user callback.</td>
</tr>
<tr>
<td>ibonl</td>
<td>Place the device online or offline.</td>
</tr>
<tr>
<td>ibpad</td>
<td>Change the primary address.</td>
</tr>
<tr>
<td>ibpct</td>
<td>Pass control to another GPIB device with Controller capability.</td>
</tr>
<tr>
<td>ibppc</td>
<td>Parallel poll configure.</td>
</tr>
<tr>
<td>ibrd</td>
<td>Read data from a device into a user buffer.</td>
</tr>
<tr>
<td>ibrda</td>
<td>Read data asynchronously from a device into a user buffer.</td>
</tr>
<tr>
<td>ibrdf</td>
<td>Read data from a device into a file.</td>
</tr>
</tbody>
</table>
Table 1-2. Traditional Calls: Device-Level (Continued)

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

Board-Level Calls

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

Table 1-3. Traditional Calls: Board-Level

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibask</td>
<td>Return information about software configuration parameters.</td>
</tr>
<tr>
<td>ib cac</td>
<td>Become Active Controller.</td>
</tr>
<tr>
<td>ib cmd</td>
<td>Send GPIB commands.</td>
</tr>
<tr>
<td>ib cmd a</td>
<td>Send GPIB commands asynchronously.</td>
</tr>
<tr>
<td>ib config</td>
<td>Change the software configuration parameters.</td>
</tr>
<tr>
<td>ib dma</td>
<td>Enable or disable DMA.</td>
</tr>
<tr>
<td>ib eos</td>
<td>Configure the end-of-string (EOS) termination mode or character.</td>
</tr>
<tr>
<td>ib eot</td>
<td>Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations.</td>
</tr>
<tr>
<td>ib find</td>
<td>Open and initialize an interface.</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ibgts</td>
<td>Go from Active Controller to Standby.</td>
</tr>
<tr>
<td>ibist</td>
<td>Set or clear the interface individual status bit for parallel polls.</td>
</tr>
<tr>
<td>iblines</td>
<td>Return the status of the eight GPIB control lines.</td>
</tr>
<tr>
<td>ibln</td>
<td>Check for the presence of a device on the bus.</td>
</tr>
<tr>
<td>ibloc</td>
<td>Go to Local.</td>
</tr>
<tr>
<td>ibnotify</td>
<td>Notify user of one or more GPIB events by invoking the user callback.</td>
</tr>
<tr>
<td>ibonl</td>
<td>Place the interface online or offline.</td>
</tr>
<tr>
<td>ibpad</td>
<td>Change the primary address.</td>
</tr>
<tr>
<td>ibppc</td>
<td>Parallel poll configure.</td>
</tr>
<tr>
<td>ibrd</td>
<td>Read data from a device into a user buffer.</td>
</tr>
<tr>
<td>ibrda</td>
<td>Read data asynchronously from a device into a user buffer.</td>
</tr>
<tr>
<td>ibrdf</td>
<td>Read data from a device into a file.</td>
</tr>
<tr>
<td>ibrpp</td>
<td>Conduct a parallel poll.</td>
</tr>
<tr>
<td>ibrsc</td>
<td>Request or release system control.</td>
</tr>
<tr>
<td>ibrv</td>
<td>Request service and change the serial poll status byte.</td>
</tr>
<tr>
<td>ibsad</td>
<td>Change or disable the secondary address.</td>
</tr>
<tr>
<td>ibsic</td>
<td>Assert interface clear.</td>
</tr>
<tr>
<td>ibsre</td>
<td>Set or clear the Remote Enable (REN) line.</td>
</tr>
<tr>
<td>ibstop</td>
<td>Abort asynchronous I/O operation.</td>
</tr>
<tr>
<td>ibtmo</td>
<td>Change or disable the I/O timeout period.</td>
</tr>
<tr>
<td>ibwait</td>
<td>Wait for GPIB events.</td>
</tr>
<tr>
<td>ibwrt</td>
<td>Write data to a device from a user buffer.</td>
</tr>
<tr>
<td>ibwrta</td>
<td>Write data asynchronously to a device from a user buffer.</td>
</tr>
<tr>
<td>ibwrtf</td>
<td>Write data to a device from a file.</td>
</tr>
</tbody>
</table>
**IBASK**

**Board-Level/Device-Level**

**Purpose**

Return information about software configuration parameters.

**Format**

**C**

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

**Visual Basic**

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

or

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

**Input**

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

**Output**

- **value**
  - Current value of the selected configuration item

**Function Return**

The value of *ibsta*

**Description**

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

**Possible Errors**

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

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

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Returned Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td><code>IbaAUTOPOLL</code></td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>IbaCICPROT</code></td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>IbaDMA</code></td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>IbaEndBitIsNormal</code></td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
</tr>
<tr>
<td><code>IbaEOSchar</code></td>
<td></td>
</tr>
<tr>
<td><code>IbaEOScmp</code></td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 1-4. ibask Board Configuration Parameter Options (Continued)

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Returned Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IbaEOSrd</strong></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>The EOS character is ignored during read operations.</td>
</tr>
<tr>
<td>non-zero</td>
<td>Read operation is terminated by the EOS character.</td>
</tr>
<tr>
<td>Refer to ibeos.</td>
<td></td>
</tr>
<tr>
<td><strong>IbaEOSwrt</strong></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>The EOI line is not asserted when the EOS character is sent during a write operation.</td>
</tr>
<tr>
<td>non-zero</td>
<td>The EOI line is asserted when the EOS character is sent during a write operation.</td>
</tr>
<tr>
<td>Refer to ibeos.</td>
<td></td>
</tr>
<tr>
<td><strong>IbaEOT</strong></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>The GPIB EOI line is not asserted at the end of a write operation.</td>
</tr>
<tr>
<td>non-zero</td>
<td>EOI is asserted at the end of a write.</td>
</tr>
<tr>
<td>Refer to ibeot.</td>
<td></td>
</tr>
<tr>
<td><strong>IbaHSCableLength</strong></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>High-speed (HS488) data transfer is disabled.</td>
</tr>
<tr>
<td>1 to 15</td>
<td>High-speed (HS488) data transfer is enabled. The number returned represents the number of meters of GPIB cable in your system.</td>
</tr>
<tr>
<td></td>
<td>Refer to the NI-488.2 online help for information about high-speed (HS488) data transfer. For instructions on accessing the online help, refer to the <em>Using the NI-488.2 Documentation</em> section in <em>About This Manual</em>.</td>
</tr>
<tr>
<td><strong>IbaIst</strong></td>
<td>The individual status (ist) bit of the interface.</td>
</tr>
<tr>
<td><strong>IbaPAD</strong></td>
<td>The current primary address of the interface. Refer to ibpad.</td>
</tr>
<tr>
<td><strong>IbaPP2</strong></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>The interface is in PP1 mode—remote parallel poll configuration.</td>
</tr>
<tr>
<td>non-zero</td>
<td>The interface is in PP2 mode—local parallel poll configuration.</td>
</tr>
<tr>
<td>For more information about parallel polls, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the <em>Using the NI-488.2 Documentation</em> section in <em>About This Manual</em>.</td>
<td></td>
</tr>
<tr>
<td>Option (Constant)</td>
<td>Returned Information</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IbaPPC</td>
<td>The current parallel poll configuration information of the interface. Refer to ibppc.</td>
</tr>
<tr>
<td>IbaPPollTime</td>
<td>zero: The interface uses the standard duration (2 μs) when conducting a parallel poll.</td>
</tr>
<tr>
<td></td>
<td>1 to 17: The interface uses a variable length duration when conducting a parallel poll. The duration values correspond to the ibtmo timing values.</td>
</tr>
<tr>
<td>IbaReadAdjust</td>
<td>zero: Read operations do not have pairs of bytes swapped.</td>
</tr>
<tr>
<td></td>
<td>one: Read operations have each pair of bytes swapped.</td>
</tr>
<tr>
<td>IbaRsv</td>
<td>The current serial poll status byte of the interface.</td>
</tr>
<tr>
<td>IbaSAD</td>
<td>The current secondary address of the interface. Refer to ibsad.</td>
</tr>
<tr>
<td>IbaSC</td>
<td>zero: The interface is not the GPIB System Controller.</td>
</tr>
<tr>
<td></td>
<td>non-zero: The interface is the System Controller.</td>
</tr>
<tr>
<td></td>
<td>Refer to ibrsc.</td>
</tr>
<tr>
<td>IbaSendLLO</td>
<td>zero: The GPIB LLO command is not sent when a device is put online—ibfind or ibdev.</td>
</tr>
<tr>
<td></td>
<td>non-zero: The LLO command is sent.</td>
</tr>
<tr>
<td>IbaSRE</td>
<td>zero: The interface does not automatically assert the GPIB REN line when it becomes the System Controller.</td>
</tr>
<tr>
<td></td>
<td>non-zero: The interface automatically asserts REN when it becomes the System Controller.</td>
</tr>
<tr>
<td></td>
<td>Refer to ibrsc and ibsre.</td>
</tr>
<tr>
<td>IbaTIMING</td>
<td>The current bus timing of the interface.</td>
</tr>
<tr>
<td></td>
<td>1: Normal timing (T1 delay of 2 μs).</td>
</tr>
<tr>
<td></td>
<td>2: High speed timing (T1 delay of 500 ns).</td>
</tr>
<tr>
<td></td>
<td>3: Very high speed timing (T1 delay of 350 ns).</td>
</tr>
<tr>
<td>IbaTMO</td>
<td>The current timeout period of the interface. Refer to ibtmo.</td>
</tr>
</tbody>
</table>
Table 1-5 lists the options you can use with \texttt{ibask} when \texttt{ud} is a device descriptor or a device index.

**Table 1-5. \texttt{ibask} Device Configuration Parameter Options**

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Returned Information</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbaWriteAdjust</td>
<td></td>
<td>zero</td>
<td>Write operations do not have pairs of bytes swapped.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one</td>
<td>Write operations have each pair of bytes swapped.</td>
</tr>
<tr>
<td>IbaBNA</td>
<td>The index of the GPIB access interface used by the given device descriptor.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IbaEOSchar</td>
<td>The current EOS character of the device. Refer to \texttt{ibeos}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IbaEOScmp</td>
<td>zero</td>
<td>A 7-bit compare is used for all EOS comparisons.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>An 8-bit compare is used for all EOS comparisons.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refer to \texttt{ibeos}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IbaEOSrd</td>
<td>zero</td>
<td>The EOS character is ignored during read operations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Read operation will be terminated by the EOS character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refer to \texttt{ibeos}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IbaEOSwrt</td>
<td>zero</td>
<td>The EOI line is not asserted when the EOS character is sent during a write operation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>The EOI line is asserted when the EOS character is sent during a write.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refer to \texttt{ibeos}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IbaEOT</td>
<td>zero</td>
<td>The GPIB EOI line is not asserted at the end of a write operation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>EOI is asserted at the end of a write.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refer to \texttt{ibeot}.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 1-5. ibask Device Configuration Parameter Options (Continued)

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Returned Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbaPAD</td>
<td>The current primary address of the device. Refer to ibpad.</td>
</tr>
<tr>
<td>IbaReadAdjust</td>
<td>zero: Read operations do not have pairs of bytes swapped.</td>
</tr>
<tr>
<td></td>
<td>one: Read operations have each pair of bytes swapped.</td>
</tr>
<tr>
<td>IbaREADDR</td>
<td>zero: No unnecessary addressing is performed between device-level read and write operations.</td>
</tr>
<tr>
<td></td>
<td>non-zero: Addressing is always performed before a device-level read or write operation.</td>
</tr>
<tr>
<td>IbaSAD</td>
<td>The current secondary address of the device. Refer to ibsad.</td>
</tr>
<tr>
<td>IbaSPollTime</td>
<td>The length of time the driver waits for a serial poll response when polling the device. The length of time is represented by the ibtmo timing values.</td>
</tr>
<tr>
<td>IbaTMO</td>
<td>The current timeout period of the device. Refer to ibtmo.</td>
</tr>
<tr>
<td>IbaUnAddr</td>
<td>zero: The GPIB commands Untalk (UNT) and Unlisten (UNL) are not sent after each device-level read and write operation.</td>
</tr>
<tr>
<td></td>
<td>non-zero: The UNT and UNL commands are sent after each device-level read and write.</td>
</tr>
<tr>
<td>IbaWriteAdjust</td>
<td>zero: Write operations do not have pairs of bytes swapped.</td>
</tr>
<tr>
<td></td>
<td>one: Write operations have each pair of bytes swapped.</td>
</tr>
</tbody>
</table>
IBBNA

Device-Level

Purpose
Change the access interface of a device.

Format

C

int ibbna (int ud, char *bname)

Visual Basic

CALL ibbna (ud%, bname$)

or

status% = ilbna (ud%, bname$)

Input

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

Output

Function Return The value of ibsta

Description

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

Possible Errors

EARG Either ud does not refer to a device or bname does not refer to a valid interface name.

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

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

ENEB The access interface is not installed or configured properly.

EOIP Asynchronous I/O is in progress.
IBCAC
Board-Level

Purpose
Become Active Controller.

Format
C
int ibcac (int ud, int v)

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

Input
ud
An interface unit descriptor
v
Determines if control is to be taken asynchronously or synchronously

Output
Function Return
The value of ibsta

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

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

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

Most applications do not need to use ibcac. Functions that require ATN to be asserted, such as ibcmd, do so automatically.
Possible Errors

- **EARG**  \( \text{ud} \) is valid but does not refer to an interface.
- **ECIC** The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
- **EDVR** Either \( \text{ud} \) is invalid or the NI-488.2 driver is not installed.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
IBCLR

Device-Level

Purpose
Clear a specific device.

Format

C

int ibclr (int ud)

Visual Basic

CALL ibclr (ud%)

or

status% = ilclr (ud%)

Input

ud
A device unit descriptor

Output

Function Return
The value of ibsta

Description

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

Possible Errors

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

EBUS
No devices are connected to the GPIB.

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

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

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

EOIP
Asynchronous I/O is in progress.
**IBCMD**

**Board-Level**

**Purpose**

Send GPIB commands.

**Format**

**C**

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

**Visual Basic**

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

or

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

**Input**

- **ud**
  - An interface unit descriptor
- **cmdbuf**
  - Buffer of command bytes to send
- **count**
  - Number of command bytes to send

**Output**

**Function Return**

The value of **ibsta**

**Description**

`ibcmd` sends `count` bytes from `cmdbuf` over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable, `ibcntl`. Refer to Appendix A, *Multiline Interface Messages*, for a table of the defined interface messages.

Command bytes are used to configure the state of the GPIB. They are not used to send instructions to GPIB devices. Use `ibwrt` to send device-specific instructions.
Possible Errors

- **EABO**: The timeout period expired before all of the command bytes were sent.
- **EARG**: ud is valid but does not refer to an interface.
- **ECIC**: The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
- **EDVR**: Either ud is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **ENOL**: No Listeners are on the GPIB.
- **EOIP**: Asynchronous I/O is in progress.
**IBCMDA**

**Board-Level**

**Purpose**
Send GPIB commands asynchronously.

**Format**

**C**

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

**Visual Basic**

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

or

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

**Input**

- **ud**
  An interface unit descriptor
- **cmdbuf**
  Buffer of command bytes to send
- **count**
  Number of command bytes to send

**Output**

- **Function Return**
  The value of **ibsta**

**Description**

`ibcmda` sends `count` bytes from `cmdbuf` over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable, `ibcntl`. Refer to Appendix A, *Multiline Interface Messages*, for a table of the defined interface messages.

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

The asynchronous I/O calls (`ibcmda, ibrda, ibwrta`) are designed so that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O begins, further NI-488.2 calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed; the driver returns EOIP in this case.
Once the I/O is complete, the application must resynchronize with the NI-488.2 driver. Resynchronization is accomplished by using one of the following functions:

- **ibnotify**: If the `ibsta` value passed to the `ibnotify` callback contains CMPL, the driver and application are resynchronized.
- **ibwait**: If the returned `ibsta` contains CMPL, the driver and application are resynchronized.
- **ibstop**: The I/O is canceled; the driver and application are resynchronized.
- **ibonl**: The I/O is canceled and the interface is reset; the driver and application are resynchronized.

**Possible Errors**

- **EARG**: `ud` is valid but does not refer to an interface.
- **ECIC**: The interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in *About This Manual*.
- **EDVR**: Either `ud` is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **ENOL**: No Listeners are on the GPIB.
- **EOIP**: Asynchronous I/O is in progress.
**IBCONFIG**

**Board-Level/Device-Level**

### Purpose
Change the software configuration parameters.

### Format

- **C**
  
  ```c
  ibconfig (int ud, int option, int value)
  ```

- **Visual Basic**
  
  ```vbnet
  CALL ibconfig (ud%, option%, value%)
  ```
  
  or
  
  ```vbnet
  status% = ilconfig (ud%, option%, value%)
  ```

### Input

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

### Output

- **Function Return** The value of `ibsta`

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

### Possible Errors

- **EARG** Either `option` or `value` is not valid. Table 1-5 lists the valid options.
- **ECAP** The driver is not able to make the requested change.
- **EDVR** Either `ud` is invalid or the NI-488.2 driver is not installed.
- **EOIP** Asynchronous I/O is in progress.
Table 1-6 lists the options you can use with `ibconfig` when `ud` is an interface descriptor or an interface index. Default values are in **bold italics**.

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

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbcAUTOPOLL</td>
<td>zero</td>
<td>Disable automatic serial polling.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Enable automatic serial polling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility. For more information about automatic serial polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the <em>Using the NI-488.2 Documentation</em> section in <em>About This Manual</em>.</td>
</tr>
<tr>
<td>IbcCICPROT</td>
<td>zero</td>
<td>Disable the CIC protocol.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Enable the CIC protocol.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility. For more information about the CIC protocol, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the <em>Using the NI-488.2 Documentation</em> section in <em>About This Manual</em>.</td>
</tr>
<tr>
<td>IbcDMA</td>
<td></td>
<td>Identical to <code>ibdma</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEndBitIsNormal</td>
<td>zero</td>
<td>Do not set the END bit of <code>ibsta</code> when an EOS match occurs during a read.</td>
</tr>
<tr>
<td></td>
<td><strong>non-zero</strong></td>
<td>Set the END bit of <code>ibsta</code> when an EOS match occurs during a read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEOSchar</td>
<td>Any 8-bit value.</td>
<td>This byte becomes the new EOS character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEOScmp</td>
<td>zero</td>
<td>Use 7 bits for the EOS character comparison.</td>
</tr>
<tr>
<td></td>
<td><strong>non-zero</strong></td>
<td>Use 8 bits for the EOS character comparison.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEOSrd</td>
<td>zero</td>
<td>Ignore EOS character during read operations.</td>
</tr>
<tr>
<td></td>
<td><strong>non-zero</strong></td>
<td>Terminate reads when the EOS character is read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
</tbody>
</table>
Table 1-6. **ibconfig** Board Configuration Parameter Options (Continued)

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbcEOSwrt</td>
<td>zero</td>
<td>Do not assert EOI with the EOS character during write operations.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Assert EOI with the EOS character during write operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEOT</td>
<td></td>
<td>Identical to ibeot. Changes the data termination mode for write operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcHSCableLength</td>
<td>zero</td>
<td>High-speed (HS488) data transfer is disabled.</td>
</tr>
<tr>
<td></td>
<td>1 to 15</td>
<td>The number of meters of GPIB cable in your system. The NI-488.2 software uses this information to select the appropriate high-speed (HS488) data transfer mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refer to the NI-488.2 online help for information about high-speed (HS488) data transfer. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.</td>
</tr>
<tr>
<td>IbcIst</td>
<td>zero</td>
<td>Changes the individual status (ist) bit of the interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identical to ibist.</td>
</tr>
<tr>
<td>IbcPAD</td>
<td></td>
<td>Identical to ibpad. Changes the primary address of the interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcPP2</td>
<td>zero</td>
<td>PP1 mode</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>PP2 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.</td>
</tr>
<tr>
<td>IbcPPC</td>
<td>zero</td>
<td>Configures the interface for parallel polls.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identical to board-level ibppc.</td>
</tr>
</tbody>
</table>
Table 1-6. \textit{ibconfig} Board Configuration Parameter Options (Continued)

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{IbcPPollTime}</td>
<td>\texttt{zero}</td>
<td>Use the standard duration (2 $\mu$s) when conducting a parallel poll.</td>
</tr>
<tr>
<td></td>
<td>1 to 17</td>
<td>Use a variable length duration when conducting a parallel poll. The duration represented by 1 to 17 corresponds to the \texttt{ibtmo} values.</td>
</tr>
<tr>
<td>\texttt{IbcReadAdjust}</td>
<td>\texttt{zero}</td>
<td>No byte swapping.</td>
</tr>
<tr>
<td></td>
<td>\texttt{one}</td>
<td>Swap pairs of bytes during a read.</td>
</tr>
<tr>
<td>\texttt{IbcRsv}</td>
<td>\texttt{zero}</td>
<td>Changes the serial poll status byte of the interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identical to \texttt{ibrsv}.</td>
</tr>
<tr>
<td>\texttt{IbcSAD}</td>
<td></td>
<td>Identical to \texttt{ibsad}. Changes the secondary address of the interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>\texttt{IbcSC}</td>
<td></td>
<td>Identical to \texttt{ibrsc}. Request or release system control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>\texttt{IbcSendLLO}</td>
<td>\texttt{zero}</td>
<td>Do not send LLO when putting a device online—\texttt{ibfind} or \texttt{ibdev}.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Send LLO when putting a device online—\texttt{ibfind} or \texttt{ibdev}.</td>
</tr>
<tr>
<td>\texttt{IbcSRE}</td>
<td>\texttt{zero}</td>
<td>Assert the Remote Enable (REN) line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identical to \texttt{ibsre}.</td>
</tr>
<tr>
<td>\texttt{IbcTIMING}</td>
<td></td>
<td>The T1 delay is the GPIB Source Handshake timing.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Normal timing (T1 delay of 2 $\mu$s).</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>High-speed timing (T1 delay of 500 ns).</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Very high-speed timing (T1 delay of 350 ns).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>\texttt{IbcTMO}</td>
<td></td>
<td>Identical to \texttt{ibtmo}. Changes the timeout period of the interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
</tbody>
</table>
Table 1-7 lists the options you can use with `ibconfig` when `ud` is a device descriptor or a device index. Default values are in **bold italics**.

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

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbcEosChar</td>
<td>Any 8-bit value. This byte becomes the new EOS character. Default determined by the NI-488.2 Configuration utility.</td>
<td></td>
</tr>
<tr>
<td>IbcEosCmp</td>
<td>zero</td>
<td>Use seven bits for the EOS character comparison. Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Use eight bits for the EOS character comparison. Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEosRd</td>
<td>zero</td>
<td>Ignore EOS character during read operations. Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Terminate reads when the EOS character is read. Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEosWrt</td>
<td>zero</td>
<td>Do not send EOI with the EOS character during write operations. Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Send EOI with the EOS character during writes. Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcEot</td>
<td>Identical to <code>ibeot</code>. Changes the data termination method for writes. Default determined by the NI-488.2 Configuration utility.</td>
<td></td>
</tr>
<tr>
<td>IbcPad</td>
<td>Identical to <code>ibpad</code>. Changes the primary address of the device. Default determined by the NI-488.2 Configuration utility.</td>
<td></td>
</tr>
<tr>
<td>IbcReadAdjust</td>
<td>zero</td>
<td>No byte swapping.</td>
</tr>
<tr>
<td></td>
<td>one</td>
<td>Swap pairs of bytes during a read.</td>
</tr>
</tbody>
</table>
Table 1-7. ibconfig Device Configuration Parameter Options (Continued)

<table>
<thead>
<tr>
<th>Option (Constant)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbcREADDR</td>
<td>zero</td>
<td>No unnecessary readdressing is performed between device-level reads and writes.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Addressing is always performed before a device-level read or write.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcSAD</td>
<td></td>
<td>Identical to ibsad. Changes the secondary address of the device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcSPollTime</td>
<td>0 to 17</td>
<td>Sets the length of time the driver waits for a serial poll response byte when polling the given device. The length of time represented by 0 to 17 corresponds to the ibtmo values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default is 11.</td>
</tr>
<tr>
<td>IbcTMO</td>
<td></td>
<td>Identical to ibtmo. Changes the device timeout period.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default determined by the NI-488.2 Configuration utility.</td>
</tr>
<tr>
<td>IbcUnAddr</td>
<td>zero</td>
<td>Do not send Untalk (UNT) and Unlisten (UNL) at the end of device-level reads and writes.</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
<td>Send UNT and UNL at the end of device-level reads and writes.</td>
</tr>
<tr>
<td>IbcWriteAdjust</td>
<td>zero</td>
<td>No byte swapping.</td>
</tr>
<tr>
<td></td>
<td>one</td>
<td>Swap pairs of bytes during a write.</td>
</tr>
</tbody>
</table>
IBDEV

Device-Level

Purpose
Open and initialize a device descriptor.

Format
C

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

Visual Basic

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

or

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

Input

<table>
<thead>
<tr>
<th>BdIndx</th>
<th>pad</th>
<th>sad</th>
<th>tmo</th>
<th>eot</th>
<th>eos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index of the access interface for the device</td>
<td>The primary GPIB address of the device</td>
<td>The secondary GPIB address of the device</td>
<td>The I/O timeout value</td>
<td>EOI mode of the device</td>
<td>EOS character and modes</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Function Return</th>
<th>The device descriptor or a –1</th>
</tr>
</thead>
</table>

Description

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

For more details on the meaning and effect of each input parameter, see the corresponding NI-488.2 calls for `ibbna`, `ibpad`, `ibsad`, `ibtmo`, `ibeot`, and `ibeos`. 
If `ibdev` is unable to get a valid device descriptor, a –1 is returned; the ERR bit is set in `ibsta` and `iberr` contains EDVR.

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

**Possible Errors**

- **EARG**  `pad, sad, tmo, eot, or eos` is invalid. Refer to `ibpad, ibsad, ibtmo, ibeot,` and `ibeos` for details on setting these parameters.
- **EDVR**  Either no device descriptors are available or `BdIndx` refers to a GPIB interface that is not installed.
- **ENEB**  The interface is not installed or is not properly configured.
IBDMA
Board-Level

Purpose
Enable or disable DMA.

Format
C
int ibdma (int ud, int v)

Visual Basic
CALL ibdma (ud%, v%)
or
status% = ibdma (ud%, v%)

Input
ud Interface descriptor
v Enable or disable the use of DMA

Output
Function Return The value of ibsta

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

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

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

**Format**

**C**

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

**Visual Basic**

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

or

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

**Input**

- **ud**: Interface or device descriptor
- **v**: EOS mode and character information

**Output**

**Function Return**  
The value of `ibsta`

**Description**

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

**Note**  
Defining an EOS byte does not cause the driver to automatically send that byte at the end of write I/O operations. Your application is responsible for placing the EOS byte at the end of the data strings that it defines.
Table 1-8 describes the different EOS configurations and the corresponding values of v. If no error occurs during the call, the value of the previous EOS setting is returned in iberr.

**Table 1-8. EOS Configurations**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Configuration</th>
<th>Value of v</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>High Byte</td>
</tr>
<tr>
<td>A</td>
<td>Terminate read when EOS is detected</td>
<td>00000100</td>
</tr>
<tr>
<td>B</td>
<td>Set EOI with EOS on write function</td>
<td>00001000</td>
</tr>
<tr>
<td>C</td>
<td>Compare all 8 bits of EOS byte rather than low 7 bits (all read and write functions)</td>
<td>00010000</td>
</tr>
</tbody>
</table>

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

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

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

**Examples**

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

**Possible Errors**

- **EARG** The high byte of v contains invalid bits.
- **EDVR** Either ud is invalid or the NI-488.2 driver is not installed.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
IBEOT

Board-Level/Device-Level

Purpose

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

Format

C

int ibeot (int ud, int v)

Visual Basic

CALL ibeot (ud%, v%)

or

status% = ileot (ud%, v%)

Input

ud

Interface or device descriptor

v

Enables or disables the end of transmission assertion of EOI

Output

Function Return

The value of ibsta

Description

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

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

Possible Errors

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

Purpose
Open and initialize an interface or a user-configured device descriptor.

Format
C
int ibfind (char *udname)

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

Input
udname        A user-configured device or interface name

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

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

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

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

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

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

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

Purpose
Go from Active Controller to Standby.

Format
C

int ibgts (int ud, int v)

Visual Basic

CALL ibgts (ud%, v%)

or

status% = ilgts (ud%, v%)

Input
ud
Interface descriptor

v
 Determines whether to perform acceptor handshaking

Output
Function Return
The value of ibsta

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

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


© National Instruments Corporation 1-33 NI-488.2 Function Reference Manual for Windows
Possible Errors

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

Format
C
int ibist (int ud, int v)

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

Input
ud                Interface descriptor
v                Indicates whether to set or clear the ist bit

Output
Function Return  The value of ibsta

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

For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.

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

Board-Level

Purpose
Return the status of the eight GPIB control lines.

Format
C
int iblines (int ud, short *clines)

Visual Basic
CALL iblines (ud%, clines%)

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

Input
ud          Interface descriptor

Output
clines      Returns GPIB control line state information
Function Return The value of ibsta

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOI</td>
<td>ATN</td>
<td>SRQ</td>
<td>REN</td>
<td>IFC</td>
<td>NRFD</td>
<td>NDAC</td>
<td>DAV</td>
</tr>
</tbody>
</table>

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

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

Possible Errors

- **EARG**  
  ud is valid but does not refer to an interface.

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

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

- **EOIP**  
  Asynchronous I/O is in progress.
IBLN
Board-Level/Device-Level

Purpose
Check for the presence of a device on the bus.

Format
C
int ibln (int ud, int pad, int sad, short *listen)

Visual Basic
CALL ibln (ud%, pad%, sad%, listen%)
or
status% = ibln (ud%, pad%, sad%, listen%)

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

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

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

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

EARG  Either the \texttt{pad} or \texttt{sad} argument is invalid.

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

EDVR  Either \texttt{ud} is invalid or the NI-488.2 driver is not installed.

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

EOIP  Asynchronous I/O is in progress.
IBLOC
Board-Level/Device-Level

Purpose
Go to Local.

Format
C
int ibloc (int ud)

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

Input
ud Interface or device descriptor

Output
Function Return The value of ibsta

Description
Board-Level
ibloc places the interface in local mode if it is not in a lockout state. The interface is in a
lockout state if LOK does not appear in the status word ibsta. If the interface is in a lockout
state, the call has no effect.

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

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

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

**Board-Level/Device-Level**

**Purpose**

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

**Format**

**C**

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

**Visual Basic**

Not supported

**Input**

- `ud` Interface or device descriptor
- `mask` Bit mask of GPIB events to notice
- `Callback` Pointer to the callback function (see following prototype)
- `RefData` User-defined reference data for the callback

**Output**

- **Function Return** The value of `ibsta`

**Description**

If `mask` is non-zero, `ibnotify` monitors the events specified by `mask`, and when one or more of the events is true, your `Callback` is invoked. The `ibnotify` mask bits are identical to the `ibsta` bits, and are defined in Table 1-9. For a board-level `ibnotify` call, all mask bits are valid except for ERR and RQS. For a device-level `ibnotify` call, the only valid mask bits are CMPL, TIMO, END, and RQS.

If TIMO is set in the notify mask, `ibnotify` calls the callback function when the timeout period has elapsed, if one or more of the other specified events have not already occurred. If TIMO is not set in the notify mask, the callback is not called until one or more of the specified events occur.
Table 1-9. Notify Mask Layout

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Bit Position</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMO</td>
<td>14</td>
<td>4000</td>
<td>Use the timeout period (see ibtmo) to limit the notify period</td>
</tr>
<tr>
<td>END</td>
<td>13</td>
<td>2000</td>
<td>END or EOS is detected</td>
</tr>
<tr>
<td>SRQI</td>
<td>12</td>
<td>1000</td>
<td>SRQ is asserted (board-level only)</td>
</tr>
<tr>
<td>RQS</td>
<td>11</td>
<td>800</td>
<td>Device requested service (device-level only)</td>
</tr>
<tr>
<td>CMPL</td>
<td>8</td>
<td>100</td>
<td>I/O completed</td>
</tr>
<tr>
<td>LOK</td>
<td>7</td>
<td>80</td>
<td>GPIB interface is in Lockout State (board-level only)</td>
</tr>
<tr>
<td>REM</td>
<td>6</td>
<td>40</td>
<td>GPIB interface is in Remote State (board-level only)</td>
</tr>
<tr>
<td>CIC</td>
<td>5</td>
<td>20</td>
<td>GPIB interface is Controller-In-Charge (board-level only)</td>
</tr>
<tr>
<td>ATN</td>
<td>4</td>
<td>10</td>
<td>Attention is asserted (board-level only)</td>
</tr>
<tr>
<td>TACS</td>
<td>3</td>
<td>8</td>
<td>GPIB interface is Talker (board-level only)</td>
</tr>
<tr>
<td>LACS</td>
<td>2</td>
<td>4</td>
<td>GPIB interface is Listener (board-level only)</td>
</tr>
<tr>
<td>DTAS</td>
<td>1</td>
<td>2</td>
<td>GPIB interface is in Device Trigger State (board-level only)</td>
</tr>
<tr>
<td>DCAS</td>
<td>0</td>
<td>1</td>
<td>GPIB interface is in Device Clear State (board-level only)</td>
</tr>
</tbody>
</table>

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

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

A given ud can have only one outstanding ibnotify call at any one time. If a current ibnotify is in effect for ud, it is replaced by a subsequent ibnotify call. An outstanding ibnotify call for ud can be canceled by a subsequent ibnotify call for ud that has a mask of 0.

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

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

Callback Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ud</td>
<td>Interface or device descriptor</td>
</tr>
<tr>
<td>ibsta</td>
<td>Value of ibsta</td>
</tr>
<tr>
<td>iberr</td>
<td>Value of iberr</td>
</tr>
<tr>
<td>ibcntl</td>
<td>Value of ibcntl</td>
</tr>
<tr>
<td>RefData</td>
<td>User-defined reference data for the callback</td>
</tr>
</tbody>
</table>

Callback Return Value

Bit mask of the GPIB events to notice next.

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

The return value of the Callback is interpreted as a mask value, which is used to automatically rearm the asynchronous event notification mechanism. If the return value is zero, it is not rearmed. If the return value is non-zero, the asynchronous event notification mechanism is rearmed with the return mask value. If the Callback rearm fails due to an error, the Callback is invoked with `ibsta` set to `ERR`, `iberr` set to `EDVR`, and `ibcntl` set to `IBNOTIFY_REARM_FAILED`, which is defined in `decl-32.h`.

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

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

EARG  A bit set in mask is invalid.

ECAP  ibnotify has been invoked from within an ibnotify Callback function, or the handler cannot perform notification on one or more of the specified mask bits.

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

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

Possible Error for the Callback

EDVR  The Callback return failed to rearm the Callback.
Chapter 1   NI-488.2 Traditional Calls — IBONL

IBONL
Board-Level/Device-Level

Purpose
Place the device or interface online or offline.

Format

C
int ibonl (int ud, int v)

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

Input

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

Output

   Function Return The value of ibsta

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

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

Possible Errors

  EDVR Either ud is invalid or the NI-488.2 driver is not installed.
  ENEB The interface is not installed or is not properly configured.
IBPAD
Board-Level/Device-Level

Purpose
Change the primary address.

Format
C
int ibpad (int ud, int v)

Visual Basic
CALL ibpad (ud%, v%)

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

Input
ud Interface or device descriptor
v GPIB primary address

Output
Function Return The value of ibsta

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

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

Purpose
Pass control to another GPIB device with Controller capability.

Format

C

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

Visual Basic

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

or

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

Input

**ud**
Device descriptor

Output

Function Return
The value of ibsta

Description

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

Possible Errors

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

**EBUS**
No devices are connected to the GPIB.

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

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

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

**EOIP**
Asynchronous I/O is in progress.
**IBPPC**

**Board-Level/Device-Level**

**Purpose**
Parallel poll configure.

**Format**

**C**

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

**Visual Basic**

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

or

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

**Input**

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

**Output**

- Function Return The value of `ibsta`

**Description**

**Device-Level**

If `ud` is a device descriptor, `ibppc` enables or disables the device from responding to parallel polls. The device is addressed and sent the appropriate parallel poll message—Parallel Poll Enable (PPE) or Disable (PPD). Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero to send PPD.

**Board-Level**

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

For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the *Using the NI-488.2 Documentation* section in About This Manual.
Possible Errors

EARG \( v \) does not contain a valid PPE or PPD message.

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

ECAP Board-level: The interface is not configured to perform local parallel poll configuration. Refer to ibconfig, option IbcPP2.

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

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

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

EOIP Asynchronous I/O is in progress.
IBRD
Board-Level/Device-Level

Purpose
Read data from a device into a user buffer.

Format
C
int ibrd (int ud, void *rdbuf, long count)

Visual Basic
CALL ibrd (ud%, rdbuf$)
or
status% = ilrd (ud%, rdbuf$, count&)

Input
ud
    Interface or device descriptor
count
    Number of bytes to be read from the GPIB

Output
rdbuf
    Address of buffer into which data is read
Function Return
    The value of ibsta

Description
Device-Level
If ud is a device descriptor, ibrd addresses the GPIB, reads up to count bytes of data, and places the data into the buffer specified by rdbuf. The operation terminates normally when count bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period. The actual number of bytes transferred is returned in the global variable ibcntl.

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

- **EABO**: Either `count` bytes or END was not received within the timeout period or a Device Clear message was received after the read operation began.

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

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

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

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

- **ENEB**: The interface is not installed or is not properly configured.

- **EOIP**: Asynchronous I/O is in progress.
IBRDA

Board-Level/Device-Level

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

Format
C

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

Visual Basic

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

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

Input
- `ud`  
  Interface or device descriptor
- `count`  
  Number of bytes to be read from the GPIB

Output
- `rdbuf`  
  Address of buffer into which data is read
- Function Return  
  The value of `ibsta`

Description

Device-Level
If `ud` is a device descriptor, `ibrda` addresses the GPIB, begins an asynchronous read of up to `count` bytes of data from a GPIB device, and places the data into the buffer specified by `rdbuf`. The operation terminates normally when `count` bytes have been received or END is received. The actual number of bytes transferred is returned in the global variable `ibcntl`.

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

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

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

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

Possible Errors

- **EABO** Board-level: a Device Clear message was received from the CIC.
- **EADR** Board-level: The GPIB is not correctly addressed; use `ibcmd` to address the GPIB.
  Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access interface. Use `ibpad` and `ibsad`.
- **EBUS** Device-level: No devices are connected to the GPIB.
- **ECIC** Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.
- **EDVR** Either `ud` is invalid or the NI-488.2 driver is not installed.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
IBRDF
Board-Level/Device-Level

Purpose
Read data from a device into a file.

Format
C
int ibrdf (int ud, char *flname)

Visual Basic
CALL ibrdf (ud%, flname$)

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

Input
ud           Interface or device descriptor
flname       Name of file into which data is read

Output
Function Return       The value of ibsta

Description

Device-Level
If ud is a device descriptor, ibrdf addresses the GPIB, reads data from a GPIB device, and
places the data into the file specified by flname. The operation terminates normally when
END is received. The operation terminates with an error if the transfer could not complete
within the timeout period. The actual number of bytes transferred is returned in the global
variable ibcntl.

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

**EABO** END was not received within the timeout period, or \texttt{ud} is an interface descriptor and Device Clear was received after the read operation began.

**EADR** Board-level: The GPIB is not correctly addressed; use \texttt{ibcmd} to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access interface. Use \texttt{ibpad} and \texttt{ibsad}.

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

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

**EDVR** Either \texttt{ud} is invalid or the NI-488.2 driver is not installed.

**EFSO** \texttt{ibrdf} could not access \texttt{fname}.

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

**EOIP** Asynchronous I/O is in progress.
**IBRPP**

**Board-Level/Device-Level**

**Purpose**

Conduct a parallel poll.

**Format**

**C**

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

**Visual Basic**

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

or

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

**Input**

| ud         | Interface or device descriptor |

**Output**

| ppr       | Parallel poll response byte |

**Function Return**

The value of ibsta

**Description**

`ibrpp` parallel poll all the devices on the GPIB. The result of this poll is returned in ppr.

For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.

**Possible Errors**

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

Purpose
Request or release system control.

Format

C

int ibrsc (int ud, int v)

Visual Basic
CALL ibrsc (ud%, v%)

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

Input
ud  Interface descriptor
v  Determines if system control is to be requested or released

Output
Function Return  The value of ibsta

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

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

Purpose
Conduct a serial poll.

Format
C
int ibrsp (int ud, char *spr)

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

Input
ud    Device descriptor

Output
spr    Serial poll response byte
Function Return The value of ibsta

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

For more information about serial polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.
Possible Errors

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

Board-Level

Purpose
Request service and change the serial poll status byte.

Format

C

int ibrsv (int ud, int v)

Visual Basic

CALL ibrsv (ud%, v%)

or

status% = ilrsv (ud%, v%)

Input

ud  Interface descriptor
v  Serial poll status byte

Output

Function Return  The value of ibsta

Description

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

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

Possible Errors

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

Purpose
Change or disable the secondary address.

Format
C
int ibsad (int ud, int v)

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

Input
ud  Interface or device descriptor
v  GPIB secondary address

Output
Function Return  The value of ibsta

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

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

Purpose
Assert interface clear.

Format
C
int ibsic (int ud)

Visual Basic
CALL ibsic (ud%)

CIC
status% = ilsic (ud%)

Input
ud              Interface descriptor

Output
Function Return  The value of ibsta

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

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

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

Board-Level

Purpose
Set or clear the Remote Enable (REN) line.

Format

C
int ibsre (int ud, int v)

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

Input
ud  Interface descriptor
v   Indicates whether to set or clear the REN line

Output
Function Return The value of ibsta

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

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

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

Purpose
Abort asynchronous I/O operation.

Format
C

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

Visual Basic

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

or

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

Input
ud

Interface or device descriptor

Output
Function Return

The value of `ibsta`

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

Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EABO</td>
<td>Asynchronous I/O was successfully stopped.</td>
</tr>
<tr>
<td>EBUS</td>
<td>Device-level: No devices are connected to the GPIB.</td>
</tr>
<tr>
<td>EDVR</td>
<td>Either <code>ud</code> is invalid or the NI-488.2 driver is not installed.</td>
</tr>
<tr>
<td>ENEB</td>
<td>The interface is not installed or is not properly configured.</td>
</tr>
</tbody>
</table>
IBTMO
Board-Level/Device-Level

Purpose
Change or disable the I/O timeout period.

Format
C
int ibtmo (int ud, int v)

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

Input
ud  Interface or device descriptor  
v  Timeout duration code

Output
Function Return  The value of ibsta

Description
ibtmo sets the timeout period of the interface or device to v. The timeout period is used to select the maximum duration allowed for a synchronous I/O operation (for example, ibrd and ibwrt) or for an ibwait or ibnotify operation with TIMO in the wait mask. If the operation does not complete before the timeout period elapses, the operation is aborted and TIMO is returned in ibsta. Refer to Table 1-10 for a list of valid timeout values. These timeout values represent the minimum timeout period. The actual period could be longer.

Table 1-10. Timeout Code Values

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value of v</th>
<th>Minimum Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNONE</td>
<td>0</td>
<td>disable (no timeout)</td>
</tr>
<tr>
<td>T10us</td>
<td>1</td>
<td>10 µs</td>
</tr>
<tr>
<td>T30us</td>
<td>2</td>
<td>30 µs</td>
</tr>
<tr>
<td>T100us</td>
<td>3</td>
<td>100 µs</td>
</tr>
</tbody>
</table>
### Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EARG</td>
<td>v is invalid.</td>
</tr>
<tr>
<td>EDVR</td>
<td>Either ud is invalid or the NI-488.2 driver is not installed.</td>
</tr>
<tr>
<td>ENEB</td>
<td>The interface is not installed or is not properly configured.</td>
</tr>
<tr>
<td>EOIP</td>
<td>Asynchronous I/O is in progress.</td>
</tr>
</tbody>
</table>

### Table 1-10. Timeout Code Values (Continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value of v</th>
<th>Minimum Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>T300us</td>
<td>4</td>
<td>300 µs</td>
</tr>
<tr>
<td>7ms</td>
<td>5</td>
<td>1 ms</td>
</tr>
<tr>
<td>3ms</td>
<td>6</td>
<td>3 ms</td>
</tr>
<tr>
<td>10ms</td>
<td>7</td>
<td>10 ms</td>
</tr>
<tr>
<td>30ms</td>
<td>8</td>
<td>30 ms</td>
</tr>
<tr>
<td>100ms</td>
<td>9</td>
<td>100 ms</td>
</tr>
<tr>
<td>300ms</td>
<td>10</td>
<td>300 ms</td>
</tr>
<tr>
<td>1s</td>
<td>11</td>
<td>1 s</td>
</tr>
<tr>
<td>3s</td>
<td>12</td>
<td>3 s</td>
</tr>
<tr>
<td>10s</td>
<td>13</td>
<td>10 s</td>
</tr>
<tr>
<td>30s</td>
<td>14</td>
<td>30 s</td>
</tr>
<tr>
<td>100s</td>
<td>15</td>
<td>100 s</td>
</tr>
<tr>
<td>300s</td>
<td>16</td>
<td>300 s</td>
</tr>
<tr>
<td>1000s</td>
<td>17</td>
<td>1000 s</td>
</tr>
</tbody>
</table>
IBTRG

Device-Level

Purpose

Trigger selected device.

Format

C

int ibtrg (int ud)

Visual Basic

CALL ibtrg (ud%)

or

status% = iltrg (ud%)

Input

ud

Device descriptor

Output

Function Return

The value of ibsta

Description

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

Possible Errors

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

EBUS No devices are connected to the GPIB.

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

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

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

EOIP Asynchronous I/O is in progress.
### IBWAIT

**Board-Level/Device-Level**

#### Purpose
Wait for GPIB events.

#### Format

**C**

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

**Visual Basic**

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

or

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

#### Input

- **ud**: Interface or device descriptor
- **mask**: Bit mask of GPIB events to wait for

#### Output

**Function Return**

The value of `ibsta`

#### Description

`ibwait` monitors the events specified by `mask` and delays processing until one or more of the events occur. If the wait mask is zero, `ibwait` returns immediately with the updated `ibsta` status word. If TIMO is set in the wait mask, `ibwait` returns when the timeout period has elapsed, if one or more of the other specified events have not already occurred. If TIMO is not set in the wait mask, the function waits indefinitely for one or more of the specified events to occur. The existing `ibwait` mask bits are identical to the `ibsta` bits; they are described in Table 1-11. If `ud` is a device descriptor, the only valid wait mask bits are TIMO, END, RQS, and CMPL. If `ud` is an interface descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period using the `ibtmo` function.
Possible Errors

- **EARG**: The bit set in mask is invalid.
- **EBUS**: Device-level: No devices are connected to the GPIB.
- **ECIC**: Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.
- **EDVR**: Either ud is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **ESRQ**: Device-level: If RQS is set in the wait mask, ESRQ indicates that the Stuck SRQ condition exists. For more information about serial polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.

### Table 1-11. Wait Mask Layout

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Bit Position</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMO</td>
<td>14</td>
<td>4000</td>
<td>Use the timeout period (see ibtmo) to limit the notify period</td>
</tr>
<tr>
<td>END</td>
<td>13</td>
<td>2000</td>
<td>END or EOS is detected</td>
</tr>
<tr>
<td>SRQI</td>
<td>12</td>
<td>1000</td>
<td>SRQ is asserted (board-level only)</td>
</tr>
<tr>
<td>RQS</td>
<td>11</td>
<td>800</td>
<td>Device requested service (device-level only)</td>
</tr>
<tr>
<td>CMPL</td>
<td>8</td>
<td>100</td>
<td>I/O completed</td>
</tr>
<tr>
<td>LOK</td>
<td>7</td>
<td>80</td>
<td>GPIB interface is in Lockout State (board-level only)</td>
</tr>
<tr>
<td>REM</td>
<td>6</td>
<td>40</td>
<td>GPIB interface is in Remote State (board-level only)</td>
</tr>
<tr>
<td>CIC</td>
<td>5</td>
<td>20</td>
<td>GPIB interface is Controller-In-Charge (board-level only)</td>
</tr>
<tr>
<td>ATN</td>
<td>4</td>
<td>10</td>
<td>Attention is asserted (board-level only)</td>
</tr>
<tr>
<td>TACS</td>
<td>3</td>
<td>8</td>
<td>GPIB interface is Talker (board-level only)</td>
</tr>
<tr>
<td>LACS</td>
<td>2</td>
<td>4</td>
<td>GPIB interface is Listener (board-level only)</td>
</tr>
<tr>
<td>DTAS</td>
<td>1</td>
<td>2</td>
<td>GPIB interface is in Device Trigger State (board-level only)</td>
</tr>
<tr>
<td>DCAS</td>
<td>0</td>
<td>1</td>
<td>GPIB interface is in Device Clear State (board-level only)</td>
</tr>
</tbody>
</table>
IBWRT
Board-Level/Device-Level

Purpose
Write data to a device from a user buffer.

Format
C

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

Visual Basic

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

or

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

Input

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ud</td>
<td>Interface or device descriptor</td>
</tr>
<tr>
<td>wrtbuf</td>
<td>Address of the buffer containing the bytes to write</td>
</tr>
<tr>
<td>count</td>
<td>Number of bytes to be written</td>
</tr>
</tbody>
</table>

Output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Return</td>
<td>The value of ibsta</td>
</tr>
</tbody>
</table>

Description

Device-Level

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

Board-Level

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

EABO    Either \texttt{count} bytes were not sent within the timeout period, or a Device Clear message was received after the write operation began.

EADR    Board-level: The GPIB is not correctly addressed; use \texttt{ibcmd} to address the GPIB.

Device-level: A conflict exists between the device GPIB address and the GPIB address of the device access interface. Use \texttt{ibpad} and \texttt{ibsad}.

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

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

EDVR    Either \texttt{ud} is invalid or the NI-488.2 driver is not installed.

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

ENOL    No Listeners were detected on the bus.

EOIP    Asynchronous I/O is in progress.
IBWRTA

Board-Level/Device-Level

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

Format
C

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

Visual Basic

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

or

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

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ud</td>
<td>Interface or device descriptor</td>
</tr>
<tr>
<td>wrtbuf</td>
<td>Address of the buffer containing the bytes to write</td>
</tr>
<tr>
<td>count</td>
<td>Number of bytes to be written</td>
</tr>
</tbody>
</table>

Output

Function Return The value of ibsta

Description

Device-Level

If `ud` is a device descriptor, `ibwrta` addresses the GPIB properly and writes `count` bytes from `wrtbuf` to a GPIB device. The operation terminates normally when `count` bytes have been sent. The actual number of bytes transferred is returned in the global variable `ibcntl`.

Board-Level

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

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

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

- **ibnotify**: If the ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- **ibwait**: If the returned ibsta contains CMPL, the driver and application are resynchronized.
- **ibstop**: The I/O is canceled; the driver and application are resynchronized.
- **ibonl**: The I/O is canceled and the interface is reset; the driver and application are resynchronized.

Possible Errors

- **EABO**: Board-level: A Device Clear message was received from the CIC.
- **EADR**: Board-level: The NI-488.2 is not correctly addressed; use ibcmd to address the GPIB.
- **EBUS**: Device-level: No devices are connected to the GPIB.
- **ECIC**: Device-level: The access interface is not Controller-In-Charge. For more information, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.
- **EDVR**: Either ud is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **ENOL**: No Listeners were detected on the bus.
- **EOIP**: Asynchronous I/O is in progress.
### IBWRTF

**Board-Level/Device-Level**

#### Purpose
Write data to a device from a file.

#### Format

**C**

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

**Visual Basic**

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

or

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

#### Input
- **ud**    Interface or device descriptor
- **flname** Name of file containing the data to be written

#### Output
- **Function Return** The value of `ibsta`

#### Description

##### Device-Level
If `ud` is a device descriptor, `ibwrtf` addresses the GPIB and writes all of the bytes from the file `flname` to a GPIB device. The operation terminates normally when all of the bytes have been sent. The operation terminates with an error if all of the bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the global variable `ibcntl`.

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

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

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

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

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

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

EFSO  ibwrtf could not access flname.

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

EOIP  Asynchronous I/O is in progress.
NI-488.2 Multi-Device Calls

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

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

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

**Table 2-1. Sections of Function Descriptions**

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

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

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

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

**Purpose**
Serial poll all devices.

**Format**

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

**Visual Basic**
```vbnet
CALL AllSpoll (boardID%, addrlist%(), resultlist%())
```

**Input**

<table>
<thead>
<tr>
<th>boardID</th>
<th>The interface number</th>
</tr>
</thead>
<tbody>
<tr>
<td>addrlist</td>
<td>A list of device addresses that is terminated by NOADDR</td>
</tr>
</tbody>
</table>

**Output**

| resultlist | A list of serial poll response bytes corresponding to device addresses in addrlist |

**Description**
AllSpoll serial polls all of the devices described by addrlist. It stores the poll responses in resultlist and the number of responses in ibcntl.

**Possible Errors**

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

Purpose
Clear a single device.

Format
**C**
```c
void DevClear (int boardID, Addr4882_t address)
```

**Visual Basic**
```vbnet
CALL DevClear (boardID%, address%)
```

Input
- **boardID** The interface number
- **address** Address of the device you want to clear

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

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

**Purpose**
Clear multiple devices.

**Format**

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

**Visual Basic**
CALL DevClearList (boardID%, addrlist%())

**Input**
- `boardID`: The interface number
- `addrlist`: A list of device addresses terminated by NOADDR that you want to clear

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

**Possible Errors**
- **EARG**: An invalid address appears in `addrlist`; `ibcnt1` is the index of the invalid address in the `addrlist` array.
- **EBUS**: No devices are connected to the GPIB.
- **ECIC**: The interface is not the Controller-In-Charge; see `SendIFC`.
- **EDVR**: Either `boardID` is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **EOIP**: Asynchronous I/O is in progress.
EnableLocal

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

Format

C
void EnableLocal (int boardID, Addr4882_t *addrlist)

Visual Basic
CALL EnableLocal (boardID%, addrlist%())

Input
boardID The interface number
addrlist A list of device addresses that is terminated by NOADDR

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

Possible Errors
EARG An invalid address appears in addrlist; ibcntl is the index of the invalid address in the addrlist array.
EBUS No devices are connected to the GPIB.
ECIC The interface is not the Controller-In-Charge; see SendIFC.
EDVR Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB The interface is not installed or is not properly configured.
EOIP Asynchronous I/O is in progress.
ESAC The interface is not configured as System Controller.
EnableRemote

Purpose
Enable remote GPIB programming for devices.

Format

C

void EnableRemote (int boardID, Addr4882_t *addrlist)

Visual Basic

CALL EnableRemote (boardID%, addrlist%())

Input

boardID  The interface number
addrlist  A list of device addresses that is terminated by NOADDR

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

Possible Errors

EARG  An invalid address appears in addrlist; ibcnt1 is the index of the invalid address in the addrlist array.
EBUS  No devices are connected to the GPIB.
ECIC  The interface is not the Controller-In-Charge; see SendIFC.
EDVR  Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB  The interface is not installed or is not properly configured.
EOIP  Asynchronous I/O is in progress.
ESAC  The interface is not configured as System Controller.
FindLstn

Purpose
Find listening devices on the GPIB.

Format

C

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

Visual Basic

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

Input

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

Output

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

Description
FindLstn tests all of the primary addresses in padlist as follows: If a device is present at a primary address given in padlist, the primary address is stored in resultlist. Otherwise, all secondary addresses of the primary address are tested, and the addresses of any devices found are stored in resultlist. No more than limit addresses are stored in resultlist. ibcntl contains the actual number of addresses stored in resultlist.

Possible Errors

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

Purpose
Determine which device is requesting service.

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

Visual Basic
CALL FindRQS (boardID%, addrlist%(), result%)

Input
boardID The interface number
addrlist List of device addresses that is terminated by NOADDR

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

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

Possible Errors
EARG An invalid address appears in addrlist; ibcntl is the index of the invalid address in the addrlist array.
EBUS No devices are connected to the GPIB.
ECIC The interface is not the Controller-In-Charge; see SendIFC.
EDVR Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB The interface is not installed or is not properly configured.
EOIP Asynchronous I/O is in progress.
ETAB None of the devices in addrlist are requesting service or addrlist contains only NOADDR. ibcntl contains the index of NOADDR in addrlist.
PassControl

Purpose
Pass control to another device with Controller capability.

Format
C
void PassControl (int boardID, Addr4882_t address)

Visual Basic
CALL PassControl (boardID%, address%)

Input
boardID The interface number
address Address of the device to which you want to pass control

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

Possible Errors
EARG The address parameter is invalid. It must be a valid primary/secondary address pair. It cannot be the constant NOADDR.
EBUS No devices are connected to the GPIB.
ECIC The interface is not the Controller-In-Charge; see SendIFC.
EDVR Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB The interface is not installed or is not properly configured.
EOIP Asynchronous I/O is in progress.
PPoll

Purpose
Perform a parallel poll on the GPIB.

Format
C
void PPoll (int boardID, short *result)

Visual Basic
CALL PPoll (boardID%, result%)

Input
boardID The interface number

Output
result The parallel poll result

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

For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.

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

Purpose
Configure a device to respond to parallel polls.

Format

**C**

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

**Visual Basic**

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

Input

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boardID</td>
<td>The interface number</td>
</tr>
<tr>
<td>address</td>
<td>Address of the device to be configured</td>
</tr>
<tr>
<td>dataline</td>
<td>Data line (a value in the range of 1 to 8) on which the device responds to parallel polls</td>
</tr>
<tr>
<td>lineSense</td>
<td>Sense (either 0 or 1) of the parallel poll response</td>
</tr>
</tbody>
</table>

Description

PPollConfig configures the device described by address to respond to parallel polls by asserting or not asserting the GPIB data line, dataline. If lineSense equals the individual status (ist) bit of the device, the assigned GPIB data line is asserted during a parallel poll. Otherwise, the data line is not asserted during a parallel poll. The Controller can use parallel polling to obtain 1-bit, device-dependent status messages from up to eight devices simultaneously.

For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.
Possible Errors

EARG    Either the address parameter is invalid, dataline is not in the range 1 to 8, or lineSense is not 0 or 1. The address must be a valid primary/secondary address pair. It cannot be the constant NOADDR.

EBUS    No devices are connected to the GPIB.

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

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

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

EOIP    Asynchronous I/O is in progress.
**PPollUnconfig**

**Purpose**

Unconfigure devices for parallel polls.

**Format**

**C**

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

**Visual Basic**

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

**Input**

- **boardID**: The interface number
- **addrlist**: A list of device addresses that is terminated by NOADDR

**Description**

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

For more information about parallel polling, refer to the NI-488.2 online help. For instructions on accessing the online help, refer to the Using the NI-488.2 Documentation section in About This Manual.

**Possible Errors**

- **EARG**: An invalid address appears in `addrlist`; `ibcntl` is the index of the invalid address in the `addrlist` array.
- **EBUS**: No devices are connected to the GPIB.
- **ECIC**: The interface is not the Controller-In-Charge; see `SendIFC`.
- **EDVR**: Either `interfaceID` is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **EOIP**: Asynchronous I/O is in progress.
**RcvRespMsg**

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

**Format**

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

**Visual Basic**

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

**Input**
- `boardID` The interface number
- `count` Number of bytes read
- `termination` Description of the data termination mode
  - `STOPend` or an 8-bit EOS character

**Output**
- `buffer` Stores the received data bytes

**Description**

`RcvRespMsg` reads up to `count` bytes from the GPIB and places these bytes into `buffer`. Data bytes are read until either `count` data bytes have been read or the termination condition is detected. If the termination condition is `STOPend`, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when the 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable, `ibcnt1`.

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

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

Purpose
Serial poll a single device.

Format
C

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

Visual Basic
CALL ReadStatusByte (boardID%, address%, result%)

Input
- boardID: The interface number
- address: A device address

Output
- result: Serial poll response byte

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

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

Purpose
Read data bytes from a device.

Format

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

Visual Basic
CALL Receive (boardID%, address%, buffer$, termination%)

Input

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

Output

buffer      Stores the received data bytes

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

EABO  The I/O timeout period elapsed before all the bytes were received.
EARG  The address or termination parameter is invalid. The address must be a valid primary/secondary address pair. It cannot be the constant NOADDR.
EBUS  No devices are connected to the GPIB.
ECIC  The interface is not the Controller-In-Charge; see SendIFC.
EDVR  Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB  The interface is not installed or is not properly configured.
EOIP  Asynchronous I/O is in progress.
ReceiveSetup

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

Format
C

void ReceiveSetup (int boardID, Addr4882_t address)

Visual Basic
CALL ReceiveSetup (boardID%, address%)

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

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

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

Purpose
Reset and initialize IEEE 488.2-compliant devices.

Format
C

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

Visual Basic

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

Input

- **boardID**: The interface number
- **addrlist**: A list of device addresses that is terminated by `NOADDR`

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

Possible Errors

- **EABO**: I/O operation is aborted.
- **EARG**: Either an invalid address appears in `addrlist` or `addrlist` is empty;
  `ibcntl` is the index of the invalid address in the `addrlist` array.
- **EBUS**: No devices are connected to the GPIB.
- **ECIC**: The interface is not the Controller-In-Charge; see `SendIFC`.
- **EDVR**: Either `boardID` is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **ENOL**: No Listeners are on the GPIB.
- **EOIP**: Asynchronous I/O is in progress.
- **ESAC**: The interface is not System Controller.
Send

Purpose
Send data bytes to a device.

Format
**C**

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

**Visual Basic**

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

Input
- **boardID**: The interface number
- **address**: Address of a device to which data is sent
- **buffer**: The data bytes to be sent
- **count**: Number of bytes to be sent
- **eotmode**: The data termination mode: **DABend**, **NULLend**, or **NLend**

Description
Send addresses the device described by **address** to listen and the interface to talk. Then, **count** bytes from **buffer** are sent to the device. The last byte is sent with the EOI line asserted if **eotmode** is **DABend**. The last byte is sent without the EOI line asserted if **eotmode** is **NULLend**. If **eotmode** is **NLend**, a new line character ("\n") is sent with the EOI line asserted after the last byte of **buffer**. The actual number of bytes transferred is returned in the global variable, **ibcnt1**.
Possible Errors

- **EABO**: The I/O timeout period has expired before all of the bytes were sent.
- **EARG**: Either the `address` parameter or `eotmode` parameter is invalid, or the `buffer` is empty and `eotmode` is `DABend`. The `address` must be a valid primary/secondary address pair; it cannot be the constant `NOADDR`. The `eotmode` parameter can only be `DABend`, `NULLend`, or `NLend`.
- **EBUS**: No devices are connected to the GPIB.
- **ECIC**: The interface is not the Controller-In-Charge; see `SendIFC`.
- **EDVR**: Either `boardID` is invalid or the NI-488.2 driver is not installed.
- **ENEB**: The interface is not installed or is not properly configured.
- **ENOL**: No Listeners are on the GPIB to accept the data bytes.
- **EOIP**: Asynchronous I/O is in progress.
SendCmds

Purpose
Send GPIB command bytes.

Format

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

Visual Basic
CALL SendCmds (boardID%, buffer$)

Input
boardID The interface number
buffer Command bytes to be sent
count Number of bytes to be sent

Description
SendCmds sends count command bytes from buffer over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable ibcntl. Refer to Appendix A, Multiline Interface Messages, for a listing of the defined interface messages.

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

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

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

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

Visual Basic
CALL SendDataBytes (boardID%, buffer$, eotmode%)

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

Description
SendDataBytes sends count number of bytes from the buffer to devices which are already addressed to listen. The last byte is sent with the EOI line asserted if eotmode is DABend; the last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend then a new line character ('\n') is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global variable, ibcnt1.

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

- **EABO**  The I/O timeout period expired before all of the bytes were sent.
- **EADR**  The interface is not talk-active; use SendSetup to address the GPIB properly.
- **EARG**  Either the eotmode parameter is invalid (it can only be DABend, NULLend, or NLend), or the buffer is empty and the eotmode is DABend.
- **ECIC**  The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR**  Either boardID is invalid or the NI-488.2 driver is not installed.
- **ENEB**  The interface is not installed or is not properly configured.
- **ENOL**  No Listeners are on the GPIB to accept the data bytes; use SendSetup to address the GPIB properly.
- **EOIP**  Asynchronous I/O is in progress.
SendIFC

Purpose
Reset the GPIB by sending interface clear.

Format
C
void SendIFC (int boardID)

Visual Basic
CALL SendIFC (boardID%)

Input
boardID The interface number

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

Possible Errors
EDVR Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB The interface is not installed or is not properly configured.
EOIP Asynchronous I/O is in progress.
ESAC The interface is not configured as the System Controller; see ibrsc.
SendList

Purpose
Send data bytes to multiple GPIB devices.

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

Visual Basic
CALL SendList (boardID%, addrlist%(), buffer$, eotmode%)

Input
boardID The interface number
addrlist A list of device addresses to send data
buffer The data bytes to be sent
count Number of bytes transmitted
eotmode The data termination mode: DABend, NULLend, or NLend

Description
SendList addresses the devices described by addrlist to listen and the interface to talk. Then, count bytes from buffer are sent to the devices. The last byte is sent with the EOI line asserted if eotmode is DABend. The last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend, a new line character ('\n') is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global variable, ibcntl.
Possible Errors

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

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

Format
C

void SendLLO (int boardID)

Visual Basic

CALL SendLLO (boardID%)

Input
boardID The interface number

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

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

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

Format
C

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

Visual Basic

```vbnet
CALL SendSetup (boardID%, addrlist%())
```

Input

- `boardID` The interface number
- `addrlist` A list of device addresses that is terminated by NOADDR

Description

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

Possible Errors

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

Purpose
Place devices in Remote With Lockout State.

Format
C
void SetRWLS (int boardID, Addr4882_t *addrlist)

Visual Basic
CALL SetRWLS (boardID%, addrlist%())

Input
boardID The interface number
addrlist A list of device addresses that is terminated by NOADDR

Description
SetRWLS places the devices described by addrlist in remote mode by asserting the Remote Enable (REN) GPIB line. Then, those devices are placed in lockout state by the Local Lockout (LLO) GPIB message. You cannot program those devices locally until the Controller-In-Charge releases the Local Lockout by way of the EnableLocal call.

Possible Errors
- EARG: Either an invalid address appears in addrlist or the addrlist is empty; ibcntl is the index of the invalid address in the addrlist array.
- EBUS: No devices are connected to the GPIB.
- ECIC: The interface is not the Controller-In-Charge; see SendIFC.
- EDVR: Either boardID is invalid or the NI-488.2 driver is not installed.
- ENEB: The interface is not installed or is not properly configured.
- EOIP: Asynchronous I/O is in progress.
- ESAC: The interface is not configured as System Controller.
TestSRQ

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

Format

C

void TestSRQ (int boardID, short *result)

Visual Basic

CALL TestSRQ (boardID%, result%)

Input
boardID The interface number

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

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

Possible Errors

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

**Purpose**
Cause IEEE 488.2-compliant devices to conduct self tests.

**Format**

**C**

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

**Visual Basic**

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

**Input**

- `boardID`: The interface number
- `addrlist`: A list of device addresses terminated by NOADDR

**Output**

- `resultlist`: A list of test results; each entry corresponds to an address in `addrlist`

**Description**

TestSys sends the "*TST?" message to the IEEE 488.2-compliant devices described by `addrlist`. The "*TST?" message instructs them to conduct their self-test procedures. A 16-bit test result code is read from each device and stored in `resultlist`. A test result of 0 indicates that the device passed its self test. Refer to the documentation that came with the device to determine the meaning of the failure code. Any other value indicates that the device failed its self test. If the function returns without an error (that is, the ERR bit is not set in `ibsta`), `ibcntl` contains the number of devices that failed. Otherwise, the meaning of `ibcntl` depends on the error returned. If a device fails to send a response before the timeout period expires, a test result of -1 is reported for it, and the error EABO is returned.
Possible Errors

EABO  The interface timed out before receiving a result from a device; ibcntl contains the index of the timed-out device. -1 is stored as the test result for the timed-out device.

EARG  Either an invalid address appears in addrlist or the addrlist is empty; ibcntl is the index of the invalid address in the addrlist array.

EBUS  No devices are connected to the GPIB.

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

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

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

ENOL  No Listeners are on the GPIB.

EOIP  Asynchronous I/O is in progress.
Trigger

Purpose
Trigger a device.

Format
C
void Trigger (int boardID, Addr4882_t address)

Visual Basic
CALL Trigger (boardID%, address%)

Input
boardID The interface number
address Address of a device to be triggered

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

Possible Errors
EARG The address parameter is invalid.
EBUS No devices are connected to the GPIB.
ECIC The interface is not the Controller-In-Charge; see SendIFC.
EDVR Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB The interface is not installed or is not properly configured.
EOIP Asynchronous I/O is in progress.
Chapter 2   NI-488.2 Multi-Device Calls — TriggerList

**TriggerList**

**Purpose**
Trigger multiple devices.

**Format**

**C**

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

**Visual Basic**

```vbnet
CALL TriggerList (boardID%, addrlist%())
```

**Input**

- **boardID** The interface number
- **addrlist** A list of device addresses terminated by NOADDR

**Description**

TriggerList sends the Group Execute Trigger (GET) GPIB message to the devices described by addrlist. If the only address in addrlist is the constant NOADDR, no addressing is performed and the GET message is sent to all devices that are currently listen-active on the GPIB.

**Possible Errors**

- **EARG** An invalid address appears in addrlist; ibcnt1 is the index of the invalid address in the addrlist array.
- **EBUS** No devices are connected to the GPIB.
- **ECIC** The interface is not the Controller-In-Charge; see SendIFC.
- **EDVR** Either boardID is invalid or the NI-488.2 driver is not installed.
- **ENEB** The interface is not installed or is not properly configured.
- **EOIP** Asynchronous I/O is in progress.
WaitSRQ

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

Format
C

    void WaitSRQ (int boardID, short *result)

Visual Basic

    CALL WaitSRQ (boardID%, result%)

Input
boardID
The interface number

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

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

Possible Errors
EDVR Either boardID is invalid or the NI-488.2 driver is not installed.
ENEB The interface is not installed or is not properly configured.
EOIP Asynchronous I/O is in progress.
Supplemental Calls for Multithreaded Applications

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

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

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

Table 3-1. Sections of Function Descriptions

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreadIbcnt</td>
<td>Return the value of the thread-specific ibcnt.</td>
</tr>
<tr>
<td>ThreadIbcntl</td>
<td>Return the value of the thread-specific ibcntl.</td>
</tr>
<tr>
<td>ThreadIberr</td>
<td>Return the value of the thread-specific iberr.</td>
</tr>
<tr>
<td>ThreadIbsta</td>
<td>Return the value of the thread-specific ibsta.</td>
</tr>
</tbody>
</table>

Table 3-2. Supplemental Calls for Multithreaded Applications
ThreadIbcnt

Purpose
Return the value of the thread-specific ibcnt.

Format

C
int ThreadIbcnt ()

Visual Basic
rc% = ThreadIbcnt ()

Input
none No input parameters

Output
Function Return Value of ibcnt for the calling thread

Description
ThreadIbcnt returns the current value of ibcnt for a particular thread of execution. The global NI-488.2 status variables (ibsta, iberr, ibcnt, ibcntl) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status variables are maintained on a per thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status variables using ThreadIbsta, ThreadIberr, ThreadIbcnt, and ThreadIbcntl instead of the global NI-488.2 status variables.
ThreadIbcntl

Purpose
Return the value of the thread-specific ibcntl.

Format

C

long ThreadIbcntl ()

Visual Basic

rc& = ThreadIbcntl ()

Input
none No input parameters

Output
Function Return Value of ibcntl for the calling thread

Description
ThreadIbcntl returns the current value of ibcntl for a particular thread of execution. The global NI-488.2 status variables (ibsta, iberr, ibcnt, ibcntl) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status variables are maintained on a per thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status variables using ThreadIbsta, ThreadIberr, ThreadIbcnt, and ThreadIbcntl instead of the global NI-488.2 status variables.
ThreadIberr

Purpose
Return the value of the thread-specific iberr.

Format

C

int ThreadIberr ()

Visual Basic

rc% = ThreadIberr ()

Input
none No input parameters

Output
Function Return Value of iberr for the calling thread

Description
ThreadIberr returns the current value of iberr for a particular thread of execution. The global NI-488.2 status variables (ibsta, iberr, ibcnt, ibcntl) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status variables are maintained on a per thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status variables using ThreadIbsta, ThreadIberr, ThreadIbcnt, and ThreadIbcntl instead of the global NI-488.2 status variables.
ThreadIbsta

Purpose
Return the value of the thread-specific ibsta.

Format

C
int ThreadIbsta ()

Visual Basic
rc% = ThreadIbsta ()

Input
none No input parameters

Output
Function Return Value of ibsta for the calling thread

Description
ThreadIbsta returns the current value of ibsta for a particular thread of execution. The global NI-488.2 status variables (ibsta, iberr, ibcnt, ibcntl) are maintained on a per process basis, which means that their values are updated whenever any thread in that process makes NI-488.2 calls. The thread NI-488.2 status variables are maintained on a per thread basis, which means that their values are updated whenever that particular thread makes NI-488.2 calls. If your application performs NI-488.2 operations in multiple threads, your application should examine the thread NI-488.2 status variables using ThreadIbsta, ThreadIberr, ThreadIbcnt, and ThreadIbcntl instead of the global NI-488.2 status variables.
Multiline Interface Messages

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

The multiline interface messages are commands defined by the IEEE 488 standard. The messages are sent and received with ATN asserted. The interface functions include initializing the bus, addressing and unaddressing devices, and setting device modes for local or remote programming. For more information about these messages, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation.*
### Table A-1. Multiline Interface Messages

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>ASCII</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>NUL</td>
<td>—</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>SOH</td>
<td>GTL</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>STX</td>
<td>—</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>ETX</td>
<td>—</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>EOT</td>
<td>SDC</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>ENQ</td>
<td>PPC</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>ACK</td>
<td>—</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>BEL</td>
<td>—</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>BS</td>
<td>GET</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>HT</td>
<td>TCT</td>
</tr>
<tr>
<td>0A</td>
<td>10</td>
<td>LF</td>
<td>—</td>
</tr>
<tr>
<td>0B</td>
<td>11</td>
<td>VT</td>
<td>—</td>
</tr>
<tr>
<td>0C</td>
<td>12</td>
<td>FF</td>
<td>—</td>
</tr>
<tr>
<td>0D</td>
<td>13</td>
<td>CR</td>
<td>—</td>
</tr>
<tr>
<td>0E</td>
<td>14</td>
<td>SO</td>
<td>—</td>
</tr>
<tr>
<td>0F</td>
<td>15</td>
<td>SI</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>DLE</td>
<td>—</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>DC1</td>
<td>LLO</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>DC2</td>
<td>—</td>
</tr>
<tr>
<td>13</td>
<td>19</td>
<td>DC3</td>
<td>—</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>DC4</td>
<td>DCL</td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>NAK</td>
<td>PPU</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
<td>SYN</td>
<td>—</td>
</tr>
<tr>
<td>17</td>
<td>23</td>
<td>ETB</td>
<td>—</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>CAN</td>
<td>SPE</td>
</tr>
<tr>
<td>19</td>
<td>25</td>
<td>EM</td>
<td>SPD</td>
</tr>
<tr>
<td>1A</td>
<td>26</td>
<td>SUB</td>
<td>—</td>
</tr>
<tr>
<td>1B</td>
<td>27</td>
<td>ESC</td>
<td>—</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>FS</td>
<td>—</td>
</tr>
<tr>
<td>1D</td>
<td>29</td>
<td>GS</td>
<td>—</td>
</tr>
<tr>
<td>1E</td>
<td>30</td>
<td>RS</td>
<td>—</td>
</tr>
<tr>
<td>1F</td>
<td>31</td>
<td>US</td>
<td>CFE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>ASCII</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>32</td>
<td>SP</td>
<td>MLA0</td>
</tr>
<tr>
<td>21</td>
<td>33</td>
<td>!</td>
<td>MLA1</td>
</tr>
<tr>
<td>22</td>
<td>34</td>
<td>&quot;</td>
<td>MLA2</td>
</tr>
<tr>
<td>23</td>
<td>35</td>
<td>#</td>
<td>MLA3</td>
</tr>
<tr>
<td>24</td>
<td>36</td>
<td>$</td>
<td>MLA4</td>
</tr>
<tr>
<td>25</td>
<td>37</td>
<td>%</td>
<td>MLA5</td>
</tr>
<tr>
<td>26</td>
<td>38</td>
<td>&amp;</td>
<td>MLA6</td>
</tr>
<tr>
<td>27</td>
<td>39</td>
<td>'</td>
<td>MLA7</td>
</tr>
<tr>
<td>28</td>
<td>40</td>
<td>(</td>
<td>MLA8</td>
</tr>
<tr>
<td>29</td>
<td>41</td>
<td>)</td>
<td>MLA9</td>
</tr>
<tr>
<td>2A</td>
<td>42</td>
<td>&quot;</td>
<td>MLA10</td>
</tr>
<tr>
<td>2B</td>
<td>43</td>
<td>+</td>
<td>MLA11</td>
</tr>
<tr>
<td>2C</td>
<td>44</td>
<td>.</td>
<td>MLA12</td>
</tr>
<tr>
<td>2D</td>
<td>45</td>
<td>-</td>
<td>MLA13</td>
</tr>
<tr>
<td>2E</td>
<td>46</td>
<td>.</td>
<td>MLA14</td>
</tr>
<tr>
<td>2F</td>
<td>47</td>
<td>/</td>
<td>MLA15</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>0</td>
<td>MLA16</td>
</tr>
<tr>
<td>31</td>
<td>49</td>
<td>1</td>
<td>MLA17</td>
</tr>
<tr>
<td>32</td>
<td>50</td>
<td>2</td>
<td>MLA18</td>
</tr>
<tr>
<td>33</td>
<td>51</td>
<td>3</td>
<td>MLA19</td>
</tr>
<tr>
<td>34</td>
<td>52</td>
<td>4</td>
<td>MLA20</td>
</tr>
<tr>
<td>35</td>
<td>53</td>
<td>5</td>
<td>MLA21</td>
</tr>
<tr>
<td>36</td>
<td>54</td>
<td>6</td>
<td>MLA22</td>
</tr>
<tr>
<td>37</td>
<td>55</td>
<td>7</td>
<td>MLA23</td>
</tr>
<tr>
<td>38</td>
<td>56</td>
<td>8</td>
<td>MLA24</td>
</tr>
<tr>
<td>39</td>
<td>57</td>
<td>9</td>
<td>MLA25</td>
</tr>
<tr>
<td>3A</td>
<td>58</td>
<td>:</td>
<td>MLA26</td>
</tr>
<tr>
<td>3B</td>
<td>59</td>
<td>:</td>
<td>MLA27</td>
</tr>
<tr>
<td>3C</td>
<td>60</td>
<td>&lt;</td>
<td>MLA28</td>
</tr>
<tr>
<td>3D</td>
<td>61</td>
<td>=</td>
<td>MLA29</td>
</tr>
<tr>
<td>3E</td>
<td>62</td>
<td>&gt;</td>
<td>MLA30</td>
</tr>
<tr>
<td>3F</td>
<td>63</td>
<td>?</td>
<td>UNL</td>
</tr>
</tbody>
</table>
### Table A-1. Multiline Interface Messages (Continued)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>ASCII</th>
<th>Message</th>
<th>Hex</th>
<th>Dec</th>
<th>ASCII</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>64</td>
<td>@</td>
<td>MTA0</td>
<td>60</td>
<td>96</td>
<td>`</td>
<td>MSA0, PPE</td>
</tr>
<tr>
<td>41</td>
<td>65</td>
<td>A</td>
<td>MTA1</td>
<td>61</td>
<td>97</td>
<td>a</td>
<td>MSA1, PPE, CFG1</td>
</tr>
<tr>
<td>42</td>
<td>66</td>
<td>B</td>
<td>MTA2</td>
<td>62</td>
<td>98</td>
<td>b</td>
<td>MSA2, PPE, CFG2</td>
</tr>
<tr>
<td>43</td>
<td>67</td>
<td>C</td>
<td>MTA3</td>
<td>63</td>
<td>99</td>
<td>c</td>
<td>MSA3, PPE, CFG3</td>
</tr>
<tr>
<td>44</td>
<td>68</td>
<td>D</td>
<td>MTA4</td>
<td>64</td>
<td>100</td>
<td>d</td>
<td>MSA4, PPE, CFG4</td>
</tr>
<tr>
<td>45</td>
<td>69</td>
<td>E</td>
<td>MTA5</td>
<td>65</td>
<td>101</td>
<td>e</td>
<td>MSA5, PPE, CFG5</td>
</tr>
<tr>
<td>46</td>
<td>70</td>
<td>F</td>
<td>MTA6</td>
<td>66</td>
<td>102</td>
<td>f</td>
<td>MSA6, PPE, CFG6</td>
</tr>
<tr>
<td>47</td>
<td>71</td>
<td>G</td>
<td>MTA7</td>
<td>67</td>
<td>103</td>
<td>g</td>
<td>MSA7, PPE, CFG7</td>
</tr>
<tr>
<td>48</td>
<td>72</td>
<td>H</td>
<td>MTA8</td>
<td>68</td>
<td>104</td>
<td>h</td>
<td>MSA8, PPE, CFG8</td>
</tr>
<tr>
<td>49</td>
<td>73</td>
<td>I</td>
<td>MTA9</td>
<td>69</td>
<td>105</td>
<td>i</td>
<td>MSA9, PPE, CFG9</td>
</tr>
<tr>
<td>4A</td>
<td>74</td>
<td>J</td>
<td>MTA10</td>
<td>6A</td>
<td>106</td>
<td>j</td>
<td>MSA10, PPE, CFG10</td>
</tr>
<tr>
<td>4B</td>
<td>75</td>
<td>K</td>
<td>MTA11</td>
<td>6B</td>
<td>107</td>
<td>k</td>
<td>MSA11, PPE, CFG11</td>
</tr>
<tr>
<td>4C</td>
<td>76</td>
<td>L</td>
<td>MTA12</td>
<td>6C</td>
<td>108</td>
<td>l</td>
<td>MSA12, PPE, CFG12</td>
</tr>
<tr>
<td>4D</td>
<td>77</td>
<td>M</td>
<td>MTA13</td>
<td>6D</td>
<td>109</td>
<td>m</td>
<td>MSA13, PPE, CFG13</td>
</tr>
<tr>
<td>4E</td>
<td>78</td>
<td>N</td>
<td>MTA14</td>
<td>6E</td>
<td>110</td>
<td>n</td>
<td>MSA14, PPE, CFG14</td>
</tr>
<tr>
<td>4F</td>
<td>79</td>
<td>O</td>
<td>MTA15</td>
<td>6F</td>
<td>111</td>
<td>o</td>
<td>MSA15, PPE, CFG15</td>
</tr>
<tr>
<td>50</td>
<td>80</td>
<td>P</td>
<td>MTA16</td>
<td>70</td>
<td>112</td>
<td>p</td>
<td>MSA16, PPD</td>
</tr>
<tr>
<td>51</td>
<td>81</td>
<td>Q</td>
<td>MTA17</td>
<td>71</td>
<td>113</td>
<td>q</td>
<td>MSA17, PPD</td>
</tr>
<tr>
<td>52</td>
<td>82</td>
<td>R</td>
<td>MTA18</td>
<td>72</td>
<td>114</td>
<td>r</td>
<td>MSA18, PPD</td>
</tr>
<tr>
<td>53</td>
<td>83</td>
<td>S</td>
<td>MTA19</td>
<td>73</td>
<td>115</td>
<td>s</td>
<td>MSA19, PPD</td>
</tr>
<tr>
<td>54</td>
<td>84</td>
<td>T</td>
<td>MTA20</td>
<td>74</td>
<td>116</td>
<td>t</td>
<td>MSA20, PPD</td>
</tr>
<tr>
<td>55</td>
<td>85</td>
<td>U</td>
<td>MTA21</td>
<td>75</td>
<td>117</td>
<td>u</td>
<td>MSA21, PPD</td>
</tr>
<tr>
<td>56</td>
<td>86</td>
<td>V</td>
<td>MTA22</td>
<td>76</td>
<td>118</td>
<td>v</td>
<td>MSA22, PPD</td>
</tr>
<tr>
<td>57</td>
<td>87</td>
<td>W</td>
<td>MTA23</td>
<td>77</td>
<td>119</td>
<td>w</td>
<td>MSA23, PPD</td>
</tr>
<tr>
<td>58</td>
<td>88</td>
<td>X</td>
<td>MTA24</td>
<td>78</td>
<td>120</td>
<td>x</td>
<td>MSA24, PPD</td>
</tr>
<tr>
<td>59</td>
<td>89</td>
<td>Y</td>
<td>MTA25</td>
<td>79</td>
<td>121</td>
<td>y</td>
<td>MSA25, PPD</td>
</tr>
<tr>
<td>5A</td>
<td>90</td>
<td>Z</td>
<td>MTA26</td>
<td>7A</td>
<td>122</td>
<td>z</td>
<td>MSA26, PPD</td>
</tr>
<tr>
<td>5B</td>
<td>91</td>
<td>[</td>
<td>MTA27</td>
<td>7B</td>
<td>123</td>
<td>{</td>
<td>MSA27, PPD</td>
</tr>
<tr>
<td>5C</td>
<td>92</td>
<td>\</td>
<td>MTA28</td>
<td>7C</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5D</td>
<td>93</td>
<td>]</td>
<td>MTA29</td>
<td>7D</td>
<td>125</td>
<td>}</td>
<td>MSA29, PPD</td>
</tr>
<tr>
<td>5E</td>
<td>94</td>
<td>^</td>
<td>MTA30</td>
<td>7E</td>
<td>126</td>
<td>~</td>
<td>MSA30, PPD</td>
</tr>
<tr>
<td>5F</td>
<td>95</td>
<td>_</td>
<td>UNT</td>
<td>7F</td>
<td>127</td>
<td>DEL</td>
<td>—</td>
</tr>
</tbody>
</table>
### Multiline Interface Message Definitions

<table>
<thead>
<tr>
<th>Multiline Message</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFE †</td>
<td>Configuration Enable</td>
</tr>
<tr>
<td>CFG †</td>
<td>Configure</td>
</tr>
<tr>
<td>DCL</td>
<td>Device Clear</td>
</tr>
<tr>
<td>GET</td>
<td>Group Execute Trigger</td>
</tr>
<tr>
<td>GTL</td>
<td>Go To Local</td>
</tr>
<tr>
<td>LLO</td>
<td>Local Lockout</td>
</tr>
<tr>
<td>MLA</td>
<td>My Listen Address</td>
</tr>
<tr>
<td>MSA</td>
<td>My Secondary Address</td>
</tr>
<tr>
<td>MTA</td>
<td>My Talk Address</td>
</tr>
<tr>
<td>PPC</td>
<td>Parallel Poll Configure</td>
</tr>
<tr>
<td>PPD</td>
<td>Parallel Poll Disable</td>
</tr>
<tr>
<td>PPE</td>
<td>Parallel Poll Enable</td>
</tr>
<tr>
<td>PPU</td>
<td>Parallel Poll Unconfigure</td>
</tr>
<tr>
<td>SDC</td>
<td>Selected Device Clear</td>
</tr>
<tr>
<td>SPD</td>
<td>Serial Poll Disable</td>
</tr>
<tr>
<td>SPE</td>
<td>Serial Poll Enable</td>
</tr>
<tr>
<td>TCT</td>
<td>Take Control</td>
</tr>
<tr>
<td>UNL</td>
<td>Unlisten</td>
</tr>
<tr>
<td>UNT</td>
<td>Untalk</td>
</tr>
</tbody>
</table>

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

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

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

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

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

ERR (dev, brd)

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

TIMO (dev, brd)

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

END (dev, brd)

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

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

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

RQS (dev)

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

CMPL (dev, brd)

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

LOK (brd)

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

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

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

CIC (brd)

CIC indicates whether the GPIB interface is the Controller-In-Charge. CIC is set when the SendIFC or ibsic call is executed either while the GPIB interface is System Controller or when another Controller passes control to the GPIB interface. CIC is cleared either when the GPIB interface detects Interface Clear (IFC) from the System Controller or when the GPIB interface passes control to another device.

ATN (brd)

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

TACS (brd)

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

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

DTAS (brd)

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

DCAS (brd)

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

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

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

Table C-1 lists the GPIB error codes.

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

EDVR (0)

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

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

Solutions

Possible solutions for this error are as follows:

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

ECIC (1)

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

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

Possible solutions for this error are as follows:

- Use `ibsic` or `SendIFC` to make the GPIB interface become CIC on the GPIB.
- Use `ibrsc1` to make sure your GPIB interface is configured as System Controller.
- In multiple CIC situations, always be certain that the CIC bit appears in the status word `ibsta` before attempting these calls. If it does not appear, you can perform an `ibwait` (for CIC) call to delay further processing until control is passed to the interface.

ENOL (2)

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

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

Solutions

Possible solutions for this error are as follows:

- Make sure that the GPIB address of your device matches the GPIB address of the device to which you want to write data.
- Use the appropriate hex code in `ibcmd` to address your device.
- Check your cable connections and make sure at least two-thirds of your devices are powered on.
- Call `ibpad` (or `ibsad`, if necessary) to match the configured address to the device switch settings.
EADR (3)

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

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

Solutions

Possible solutions for this error are as follows:

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

EARG (4)

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

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

Solutions

Possible solutions for this error are as follows:

- Make sure that the parameters passed to the NI-488.2 call are valid.
- Do not use a device descriptor in an interface function or vice-versa.
ESAC (5)

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

Solutions

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

EABO (6)

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

Solutions

Possible solutions for this error are as follows:

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

ENEB (7)

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

Solutions

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

EDMA (8)

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

Solutions

Possible solutions for this error are as follows:

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

EOIP (10)

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

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

- **ibnotify**: If the `ibsta` value passed to the `ibnotify` callback contains CMPL, the driver and application are resynchronized.
- **ibwait**: If the returned `ibsta` contains CMPL, the driver and application are resynchronized.
- **ibstop**: The I/O is canceled; the driver and application are resynchronized.
- **ibonl**: The I/O is canceled and the interface is reset; the driver and application are resynchronized.

**ECAP (11)**

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

**Solutions**

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

**EFSO (12)**

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

Solutions

Possible solutions for this error are as follows:

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

EBUS (14)

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

Solutions

Possible solutions for this error are as follows:

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

ESTB (15)

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

Solutions

Possible solutions for this error are as follows:

- Call ibrsp more frequently to empty the queue.
- Disable autopoling with the ibconfig function (option IbcAUTOPOLL) or the NI-488.2 Configuration utility. To start the NI-488.2 Configuration utility, select Start→Programs→National Instruments NI-488.2→Explore GPIB. Then, select the GPIB interface under Devices and Interfaces in the left window frame, right-click, and choose Properties.
ESRQ (16)

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

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

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

Solutions

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

ETAB (20)

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

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

Solutions

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

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

NI Web Support

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

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.

- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.

- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.

- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.

- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.
Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.

- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.

- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

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

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

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

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

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-</td>
<td>nano-</td>
<td>10⁻⁹</td>
</tr>
<tr>
<td>µ-</td>
<td>micro-</td>
<td>10⁻⁶</td>
</tr>
<tr>
<td>m-</td>
<td>milli-</td>
<td>10⁻³</td>
</tr>
</tbody>
</table>

A

acceptor handshake

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

access interface

The GPIB interface that controls and communicates with the devices on the bus that are attached to it.

ANSI

American National Standards Institute.

API

Application Programmer Interface

ASCII

American Standard Code for Information Interchange.

asynchronous

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

automatic serial polling

Autopolling. A feature of the NI-488.2 software in which serial polls are executed automatically by the driver whenever a device asserts the GPIB SRQ line.

B

base I/O address

See I/O address.

BIOS

Basic Input/Output System.

board-level function

A rudimentary function that performs a single operation.
C

**CFE**  Configuration Enable. The GPIB command which precedes CFGn and is used to place devices into their configuration mode.

**CFGn**  These GPIB commands (CFG1 through CFG15) follow CFE and are used to configure all devices for the number of meters of cable in the system so that HS488 transfers occur without errors.

**CIC**  Controller-In-Charge. The device that manages the GPIB by sending interface messages to other devices.

**CPU**  Central processing unit.

D

**DAV**  Data Valid. One of the three GPIB handshake lines. See handshake.

**DCL**  Device Clear. The GPIB command used to reset the device or internal functions of all devices. See SDC.

**device-level function**  A function that combines several rudimentary board operations into one function so that the user does not have to be concerned with bus management or other GPIB protocol matters.

**DIO1 through DIO8**  The GPIB lines that are used to transmit command or data bytes from one device to another.

**DLL**  Dynamic link library.

**DMA**  Direct memory access. High-speed data transfer between the GPIB interface and memory that is not handled directly by the CPU. Not available on some systems. See programmed I/O.

**driver**  Device driver software installed within the operating system.

E

**END or END Message**  A message that signals the end of a data string. END is sent by asserting the GPIB End or Identify (EOI) line with the last data byte.

**EOI**  A GPIB line that is used to signal either the last byte of a data message (END) or the parallel poll Identify (IDY) message.
### Glossary

**EOS or EOS Byte**
A 7- or 8-bit end-of-string character that is sent as the last byte of a data message.

**EOT**
End of transmission

**ESB**
The Event Status bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.

**G**

**GET**
Group Execute Trigger. The GPIB command used to trigger a device or internal function of an addressed Listener.

**GPIB**

**GPIB address**
The address of a device on the GPIB, composed of a primary address (MLA and MTA) and perhaps a secondary address (MSA). The GPIB interface has both a GPIB address and an I/O address.

**GPIB interface**
Refers to the National Instruments family of GPIB interfaces.

**GTL**
Go To Local. The GPIB command used to place an addressed Listener in local (front panel) control mode.

**H**

**handshake**
The mechanism used to transfer bytes from the Source Handshake function of one device to the Acceptor Handshake function of another device. The three GPIB lines DA V, NRFD, and NDAC are used in an interlocked fashion to signal the phases of the transfer so that bytes can be sent asynchronously (for example, without a clock) at the speed of the slowest device. For more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*.

**hex**
Hexadecimal; a number represented in base 16. For example, decimal 16 = hex 10.

**high-level function**
See device-level function.

**Hz**
Hertz.
Glossary

I

ibcnt
After each NI-488.2 I/O call, this global variable contains the actual number of bytes transmitted. On systems with a 16-bit integer, such as MS-DOS, ibcnt is a 16-bit integer, and ibcntl is a 32-bit integer. For cross-platform compatibility, use ibcntl.

ibcntl
After each NI-488.2 I/O call, this global variable contains the actual number of bytes transmitted. On systems with a 16-bit integer, such as MS-DOS, ibcnt is a 16-bit integer, and ibcntl is a 32-bit integer. For cross-platform compatibility, use ibcntl.

iberr
A global variable that contains the specific error code associated with a function call that failed.

ibsta
At the end of each function call, this global variable (status word) contains status information.

IEEE
Institute of Electrical and Electronic Engineers.

interface message
A broadcast message sent from the Controller to all devices and used to manage the GPIB.

I/O
Input/output. In this manual, it is the transmission of commands or messages between the system via the GPIB board and other devices on the GPIB.

I/O address
The address of the GPIB interface from the point of view of the CPU, as opposed to the GPIB address of the GPIB interface. Also called port address or interface address.

ist
An Individual Status bit of the status byte used in the Parallel Poll Configure function.

L

LAD
Listen Address. See MLA.

language interface
Code that enables an application program that uses NI-488.2 calls to access the driver.

Listener
A GPIB device that receives data messages from a Talker.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLO</td>
<td>Local Lockout. The GPIB command used to tell all devices that they may or should ignore remote (GPIB) data messages or local (front panel) controls, depending on whether the device is in local or remote program mode.</td>
</tr>
<tr>
<td>low-level function</td>
<td>A rudimentary Controller-In-Charge or device function that performs a single operation.</td>
</tr>
<tr>
<td>M</td>
<td>Meters.</td>
</tr>
<tr>
<td>m</td>
<td>Meters.</td>
</tr>
<tr>
<td>MAV</td>
<td>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.</td>
</tr>
<tr>
<td>MLA</td>
<td>My Listen Address. The GPIB command used to address a device to be a Listener. It can be any one of the 31 primary addresses.</td>
</tr>
<tr>
<td>MSA</td>
<td>My Secondary Address. The GPIB command used to address a device to be a Listener or a Talker when extended (two byte) addressing is used. The complete address is a MLA or MTA address followed by an MSA address. There are 31 secondary addresses for a total of 961 distinct listen or talk addresses for devices.</td>
</tr>
<tr>
<td>MTA</td>
<td>My Talk Address. The GPIB command used to address a device to be a Talker. It can be any one of the 31 primary addresses.</td>
</tr>
<tr>
<td>multitasking</td>
<td>The concurrent processing of more than one program or task.</td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>NDAC</td>
<td>Not Data Accepted. One of the three GPIB handshake lines. See handshake.</td>
</tr>
<tr>
<td>NRFD</td>
<td>Not Ready For Data. One of the three GPIB handshake lines. See handshake.</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>parallel poll</td>
<td>The process of polling all configured devices at once and reading a composite poll response. See serial poll.</td>
</tr>
<tr>
<td>PIO</td>
<td>See programmed I/O.</td>
</tr>
</tbody>
</table>
Glossary

PPC  Parallel Poll Configure. The GPIB command used to configure an addressed Listener to participate in polls.

PPD  Parallel Poll Disable. The GPIB command used to disable a configured device from participating in polls. There are 16 PPD commands.

PPE  Parallel Poll Enable. The GPIB command used to enable a configured device to participate in polls and to assign a DIO response line. There are 16 PPE commands.

PPU  Parallel Poll Unconfigure. The GPIB command used to disable used to disable any device from participating in polls.

programmed I/O  Low-speed data transfer between the GPIB interface and memory in which the CPU moves each data byte according to program instructions. See DMA.

R  
resynchronize  The NI-488.2 software and the user application must resynchronize after asynchronous I/O operations have completed.

RQS  Request Service.

S  
s  Seconds.

SDC  Selected Device Clear. The GPIB command used to reset internal or device functions of an addressed Listener. See DCL.

serial poll  The process of polling and reading the status byte of one device at a time. See parallel poll.

service request  See SRQ.

source handshake  The GPIB interface function that transmits data and commands. Talkers use this function to send data, and the Controller uses it to send commands. See acceptor handshake and handshake.

SPD  Serial Poll Disable. The GPIB command used to cancel an SPE command.

SPE  Serial Poll Enable. The GPIB command used to enable a specific device to be polled. That device must also be addressed to talk. See SPD.
Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRQ</td>
<td>Service Request. The GPIB line that a device asserts to notify the CIC that the device needs servicing.</td>
</tr>
<tr>
<td>status byte</td>
<td>The IEEE 488.2-defined data byte sent by a device when it is serially polled.</td>
</tr>
<tr>
<td>status word</td>
<td>See \textit{ibsta}.</td>
</tr>
<tr>
<td>synchronous</td>
<td>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.</td>
</tr>
<tr>
<td>System Controller</td>
<td>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.</td>
</tr>
<tr>
<td>T</td>
<td>Talk Address. See \textit{MTA}.</td>
</tr>
<tr>
<td>Talker</td>
<td>A GPIB device that sends data messages to Listeners.</td>
</tr>
<tr>
<td>TCT</td>
<td>Take Control. The GPIB command used to pass control of the bus from the current Controller to an addressed Talker.</td>
</tr>
<tr>
<td>timeout</td>
<td>A feature of the NI-488.2 driver that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.</td>
</tr>
<tr>
<td>TLC</td>
<td>An integrated circuit that implements most of the GPIB Talker, Listener, and Controller functions in hardware.</td>
</tr>
<tr>
<td>U</td>
<td>Unit descriptor. A variable name and first argument of each function call that contains the unit descriptor of the GPIB interface or other GPIB device that is the object of the function.</td>
</tr>
<tr>
<td>UNL</td>
<td>Unlisten. The GPIB command used to unaddress any active Listeners.</td>
</tr>
<tr>
<td>UNT</td>
<td>Untalk. The GPIB command used to unaddress an active Talker.</td>
</tr>
</tbody>
</table>
Index

A
address calls
  IBPAD, 1-47
  IBSD, 1-62
AllSpoll call, 2-4
asynchronous operations, halting, 1-65
ATN status word condition, B-4

B
board configuration parameter options.
  See configuration options.
board-level calls (table), 1-3 to 1-4

C
callback, 1-42 to 1-45
calls. See NI-488.2 calls and calls for multithreaded applications.
calls for multithreaded applications
  alphabetical list of calls (table), 3-2
  Threadlbcnt, 3-3
  Threadlbcntl, 3-4
  Threadlberr, 3-5
  Threadlbst, 3-6
CIC status word condition, B-4
clear calls
  DevClear, 2-5
  DevClearList, 2-6
  IBCLR, 1-14
  IBIST, 1-35
  IBSIC, 1-63
  IBSRE, 1-64
  SendIFC, 2-28
CMPL status word condition, B-3
command calls
  IBCMD, 1-15 to 1-16
  IBCMDA, 1-17 to 1-18
  SendCmds, 2-25
configuration calls
  IBASK, 1-5 to 1-10
  IBCONFIG, 1-19 to 1-24
configuration options
  IBASK call
    board configuration parameter options, 1-6 to 1-9
    device configuration parameter options, 1-9 to 1-10
  IBCONFIG call
    board-level configuration options, 1-20 to 1-23
    device-level configuration options, 1-23 to 1-24
control line status, 1-36 to 1-37
caller calls
  IBCAC, 1-12 to 1-13
  IBGTS, 1-33 to 1-34
  IBPCT, 1-48
  IBRSIC, 1-58
  PassControl, 2-11

D
DCAS status word condition, B-5
DevClear call, 2-5
DevClearList call, 2-6
device configuration parameter options.
  See configuration options.
device-level calls (table), 1-2 to 1-3
DMA call, 1-27
Index

documentation
  accessing, xi
  conventions used in manual, xii
  related documentation, xii
DTAS status word condition, B-5

E
  EABO error code, C-5
  EADR error code, C-4
  EARG error code, C-4
  EBUS error code, C-8
  ECAP error code, C-7
  ECIC error code, C-2 to C-3
  EDMA error code, C-6
  EDVR error code, C-2
  EPSO error code, C-7 to C-8
  EnableLocal call, 2-7
  EnableRemote call, 2-8
  END status word condition, B-2
  ENEB error code, C-5 to C-6
  ENOL error code, C-3
  EOI line, enabling or disabling, 1-30
  EOIP error code, C-6 to C-7
  EOS byte, defining, 1-28
  EOS configurations, 1-29
  ERR status word condition, B-2
  error codes, C-1 to C-9
  ESAC error code, C-5
  ESRQ error code, C-9
  ESTB error code, C-8
  ETAB error code, C-9

F
  FindLstn call, 2-9
  FindRQS call, 2-10
  functions. See NI-488.2 calls and calls for multithreaded applications.

G
  GPIB error codes (table), C-1

I
  IbaAUTOPOLL configuration option, 1-6
  IbaBNA configuration option, 1-9
  IbaCICPROT configuration option, 1-6
  IbaDMA configuration option, 1-6
  IbaEndBitIsNormal configuration option, 1-6
  IbaEOSchar configuration option
    boards, 1-6
    devices, 1-9
  IbaEOScmp configuration option
    boards, 1-6
    devices, 1-9
  IbaEOSrd configuration option
    boards, 1-7
    devices, 1-9
  IbaEOSwrt configuration option
    boards, 1-7
    devices, 1-9
  IbaEOT configuration option
    boards, 1-7
    devices, 1-9
  IbaHSCableLength configuration option, 1-7
  IbaIst configuration option, 1-7
  IbaPAD configuration option
    boards, 1-7
    devices, 1-10
  IbaREADDR configuration option
    boards, 1-8
    devices, 1-10
  IbaRsv configuration option, 1-8
  IbaREADDR configuration option, 1-10
  IbaPP2 configuration option, 1-7
  IbaPPC configuration option, 1-8
  IbaPollTime configuration option, 1-8

IbaSAD configuration option
  boards, 1-8
  devices, 1-10
IbaSC configuration option, 1-8
IbaSendLLO configuration option, 1-8
IBASK call, 1-5 to 1-10
  board configuration parameter
  options, 1-6 to 1-9
  description, 1-5
  device configuration parameter
  options, 1-9 to 1-10
IbaSPollTime configuration option, 1-10
IbaSRE configuration option, 1-8
IbaTIMING configuration option, 1-8
IbaTMO configuration option
  boards, 1-8
  devices, 1-10
IbaUnAddr configuration option, 1-10
IbaWriteAdjust configuration option
  boards, 1-9
  devices, 1-10
IBBNA call, 1-11
IBCAC call, 1-12 to 1-13
IbcAUTOPOLL configuration option, 1-20
IbcCICPROT configuration option, 1-20
IbcDMA configuration option, 1-20
IbcEndBitIsNormal configuration
  option, 1-20
IbcEOSchar configuration option
  board-level, 1-20
  device-level, 1-23
IbcEOScmp configuration option
  board-level, 1-20
  device-level, 1-23
IbcEOSrd configuration option
  board-level, 1-20
  device-level, 1-23
IbcEOSwrt configuration option
  board-level, 1-21
  device-level, 1-23
IbcEOT configuration option
  board-level, 1-21
  device-level, 1-23
IbcHSCableLength configuration option, 1-21
IbcIst configuration option, 1-21
IBCLR call, 1-14
IBCMD call, 1-15 to 1-16
IBCMDA call, 1-17 to 1-18
IBCONFIG call, 1-19 to 1-24
  board-level configuration options,
  1-20 to 1-23
  description, 1-19
  device-level configuration options,
  1-23 to 1-24
IbcPAD configuration option
  board-level, 1-21
  device-level, 1-23
IbcPP2 configuration option, 1-21
IbcPPC configuration option, 1-21
IbcSPollTime configuration option, 1-22
IbcReadAdjust configuration option
  board-level, 1-22
  device-level, 1-23
IbcREADDR configuration option, 1-24
IbcRsv configuration option, 1-22
IbcSAD configuration option
  board-level, 1-22
  device-level, 1-24
IbcSC configuration option, 1-22
IbcSendLLO configuration option, 1-22
IbcSPollTime configuration option, 1-24
IbcSRE configuration option, 1-22
IbcTIMING configuration option, 1-22
IbcTMO configuration option
  board-level, 1-22
  device-level, 1-24
IbcUnAddr configuration option, 1-24
IbcWriteAdjust configuration option
  board-level, 1-23
  device-level, 1-24
Index

IBDEV call, 1-25 to 1-26
IBDMA call, 1-27
IBEOS call, 1-28 to 1-29
IBEOT call, 1-30
IBFIND call, 1-31 to 1-32
IBGTS call, 1-33 to 1-34
IBIST call, 1-35
IBLINES call, 1-36 to 1-37
IBLN call, 1-38 to 1-39
IBLOC call, 1-40 to 1-41
IBNOTIFY call, 1-42 to 1-45
IBONL call, 1-46
IBPAD call, 1-47
IBPCT call, 1-48
IBPPC call, 1-49 to 1-50
IBRD call, 1-51 to 1-52
IBRDA call, 1-53 to 1-54
IBRDF call, 1-55 to 1-56
IBRPP call, 1-57
IBRSC call, 1-58
IBRSP call, 1-59 to 1-60
IBRSV call, 1-61
IBSAD call, 1-62
IBSIC call, 1-63
IBSRE call, 1-64
ibsta (status word). See status word conditions.
IBSTOP call, 1-65
IBTMO call, 1-66 to 1-67
IBTRG call, 1-68
IBWAIT call, 1-69 to 1-70
IBWRT call, 1-71 to 1-72
IBWRTA call, 1-73 to 1-74
IBWRTF call, 1-75 to 1-76
interface clear calls
   IBSIC, 1-63
   SendIFC, 2-28

L
LACS status word condition, B-5
listeners, finding
   FindLstn call, 2-9
   IBLN call, 1-38 to 1-39
local calls
   EnableLocal, 2-7
   IBLOC, 1-40 to 1-41
   SendLLO, 2-31
lockout calls
   SendLLO, 2-31
   SetRWLS, 2-33
LOK status word condition, B-3

M
manual. See documentation.
multiline interface messages, A-1 to A-4

N
NI-488.2 calls
   multi-device calls
      AllSpoll, 2-4
      alphabetical list of calls
         (table), 2-2 to 2-3
      DevClear, 2-5
      DevClearList, 2-6
      EnableLocal, 2-7
      EnableRemote, 2-8
      FindLstn, 2-9
      FindRQS, 2-10
      PassControl, 2-11
      PPoll, 2-12
      PPollConfig, 2-13 to 2-14
      PPollUnconfig, 2-15
      RcvRespMsg, 2-16 to 2-17
      ReadStatusByte, 2-18
      Receive, 2-19 to 2-20
      ReceiveSetup, 2-21

ibsta (status word). See status word conditions.
Index

ResetSys, 2-22
Send, 2-23 to 2-24
SendCmds, 2-25
SendDataBytes, 2-26 to 2-27
SendIFC, 2-28
SendList, 2-29 to 2-30
SendLLO, 2-31
SendSetup, 2-32
SetRWLS, 2-33
TestSRQ, 2-34
TestSys, 2-35 to 2-36
Trigger, 2-37
TriggerList, 2-38
WaitSRQ, 2-39

traditional calls
alphabetical list of calls (table), 1-2 to 1-4
board-level calls (table), 1-3 to 1-4
device-level calls (table), 1-2 to 1-3
IBASK, 1-5 to 1-10
IBBNA, 1-11
IBCAC, 1-12 to 1-13
IBCLR, 1-14
IBCMD, 1-15 to 1-16
IBCMDA, 1-17 to 1-18
IBCONFIG, 1-19 to 1-24
IBDEV, 1-25 to 1-26
IBDMA, 1-27
IBEOS, 1-28 to 1-29
IBEOT, 1-30
IBFIND, 1-31 to 1-32
IBGTS, 1-33 to 1-34
IBIST, 1-35
IBLINES, 1-36 to 1-37
IBLN, 1-38 to 1-39
IBLOC, 1-40 to 1-41
IBNOTIFY, 1-42 to 1-45
IBONL, 1-46
IBPAD, 1-47
IBPCT, 1-48
IBPPC, 1-49 to 1-50
IBRD, 1-51 to 1-52
IBRDA, 1-53 to 1-54
IBRF, 1-55 to 1-56
IBRPP, 1-57
IBRSC, 1-58
IBRSV, 1-61
IBSAD, 1-62
IBSIC, 1-63
IBSRE, 1-64
IBSTOP, 1-65
IBTMO, 1-66 to 1-67
IBTRG, 1-68
IBWAIT, 1-69 to 1-70
IBWRT, 1-71 to 1-72
IBWRTA, 1-73 to 1-74
IBWRTF, 1-75 to 1-76
notify call, 1-42 to 1-45
notify mask layout (table), 1-43

O

online/offline call, 1-46

P

parallel polling calls
IBIST, 1-35
IBPPC, 1-49 to 1-50
IBRPP, 1-57
PPoll, 2-12
PPollConfig, 2-13 to 2-14
PPollUnconfig, 2-15
PassControl call, 2-11
PPoll call, 2-12
PPollConfig call, 2-13 to 2-14
PPollUnconfig call, 2-15
primary address, changing, 1-47
Index

R
RcvRespMsg call, 2-16 to 2-17
read calls
   IBRD, 1-51 to 1-52
   IBRDA, 1-53 to 1-54
   IBRDF, 1-55 to 1-56
ReadStatusByte call, 2-18
Receive call, 2-19 to 2-20
ReceiveSetup call, 2-21
REM status word condition, B-4
remote calls
   EnableRemote, 2-8
   IBSRE, 1-64
   SetRWLS, 2-33
ResetSys call, 2-22
RQS status word condition, B-3

S
Send call, 2-23 to 2-24
SendCmds call, 2-25
SendDataBytes call, 2-26 to 2-27
SendIFC call, 2-28
SendList call, 2-29 to 2-30
SendLLO call, 2-31
SendSetup call, 2-32
serial polling calls
   AllSpoll, 2-4
   IBRSP, 1-59 to 1-60
   IBRSV, 1-61
   ReadStatusByte, 2-18
service request calls
   FindRQS, 2-10
   TestSRQ, 2-34
SetRWLS call, 2-33
SRQ calls
   TestSRQ, 2-34
   WaitSRQ, 2-39
SRQI status word condition, B-3

status word conditions
   ATN, B-4
   CIC, B-4
   CMPL, B-3
   DCAS, B-5
   DTAS, B-5
   END, B-2
   ERR, B-2
   ibsta (status word) layout (table), B-1
   LACS, B-5
   LOK, B-3
   REM, B-4
   RQS, B-3
   SRQI, B-3
   TACS, B-4
   TIMO, B-2

T
TACS status word condition, B-4
technical support, D-1 to D-2
TestSRQ call, 2-34
TestSys call, 2-35 to 2-36
ThreadIbcnt call, 3-3
ThreadIbcntl call, 3-4
ThreadIberr call, 3-5
ThreadIbsta call, 3-6
timeout code values (table), 1-66 to 1-67
TIMO status word condition, B-2
trigger calls
   IBTRG, 1-68
   Trigger, 2-37
   TriggerList, 2-38
Trigger call, 2-37
TriggerList call, 2-38
Index

W

wait calls

  IBWAIT, 1-69 to 1-70
  WaitSRQ, 2-39
wait mask layout (table), 1-70
WaitSRQ call, 2-39
write calls

  IBWRT, 1-71 to 1-72
  IBWRTA, 1-73 to 1-74
  IBWRTF, 1-75 to 1-76