KPXI Compact GPIB Controller

Reference Manual

KPXI-488-901-01 Rev. A / January 2007

ECA 42912

WARRANTY

Keithley Instruments, Inc. warrants this product to be free from defects in material and workmanship for a period of 1 year from date of shipment.

Keithley Instruments, Inc. warrants the following items for 90 days from the date of shipment: probes, cables, rechargeable batteries, diskettes, and documentation.

During the warranty period, we will, at our option, either repair or replace any product that proves to be defective.

To exercise this warranty, write or call your local Keithley Instruments representative, or contact Keithley Instruments headquarters in Cleveland, Ohio. You will be given prompt assistance and return instructions. Send the product, transportation prepaid, to the indicated service facility. Repairs will be made and the product returned, transportation prepaid. Repaired or replaced products are warranted for the balance of the original warranty period, or at least 90 days.

LIMITATION OF WARRANTY

This warranty does not apply to defects resulting from product modification without Keithley Instruments' express written consent, or misuse of any product or part. This warranty also does not apply to fuses, software, non-rechargeable batteries, damage from battery leakage, or problems arising from normal wear or failure to follow instructions.

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES.

NEITHER KEITHLEY INSTRUMENTS, INC. NOR ANY OF ITS EMPLOYEES SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF ITS INSTRUMENTS AND SOFTWARE EVEN IF KEITHLEY INSTRUMENTS, INC., HAS BEEN ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH DAMAGES. SUCH EXCLUDED DAMAGES SHALL INCLUDE, BUT ARE NOT LIMITED TO: COSTS OF REMOVAL AND INSTALLATION, LOSSES SUSTAINED AS THE RESULT OF INJURY TO ANY PERSON, OR DAMAGE TO PROPERTY.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139
440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY (534-8453) • www.keithley.com

12/06

KPXI
Compact GPIB Controller
Reference Manual

©2007, Keithley Instruments, Inc.
All rights reserved.
Cleveland, Ohio, U.S.A.
Document Number: KPXI-488-901-01 Rev. A / January 2007

Manual Print History

The print history shown below lists the printing dates of all Revisions and Addenda created for this manual. The Revision Level letter increases alphabetically as the manual undergoes subsequent updates. Addenda, which are released between Revisions, contain important change information that the user should incorporate immediately into the manual. Addenda are numbered sequentially. When a new Revision is created, all Addenda associated with the previous Revision of the manual are incorporated into the new Revision of the manual. Each new Revision includes a revised copy of this print history page.

Revision A (Document Number KPXI-488-901-01)..... January 2007

All Keithley Instruments product names are trademarks or registered trademarks of Keithley Instruments, Inc.
Other brand names are trademarks or registered trademarks of their respective holders.

KEITHLEY

Safety Precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with non-hazardous voltages, there are situations where hazardous conditions may be present.

This product is intended for use by qualified personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product. Refer to the manual for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product may be impaired.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating properly, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the manual. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, and perform safe installations and repairs of products. Only properly trained service personnel may perform installation and service procedures.

Keithley Instruments products are designed for use with electrical signals that are rated Measurement Category I and Measurement Category II, as described in the International Electrotechnical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Measurement Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Measurement Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the Manual.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30V RMS, 42.4V peak, or 60VDC are present. A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 volts, no conductive part of the circuit may be exposed.

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, make sure the line cord is connected to a properly grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided, in close proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

The instrument and accessories must be used in accordance with its specifications and operating instructions or the safety of the equipment may be impaired.

Do not exceed the maximum signal levels of the instruments and accessories, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.

When fuses are used in a product, replace with same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as safety earth ground connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

If a screw is present, connect it to safety earth ground using the wire recommended in the user documentation.

The symbol on an instrument indicates that the user should refer to the operating instructions located in the manual.

The symbol on an instrument shows that it can source or measure 1000 volts or more, including the combined effect of normal and common mode voltages. Use standard safety precautions to avoid personal contact with these voltages.

The symbol on an instrument shows that the surface may be hot. Avoid personal contact to prevent burns.

The symbol indicates a connection terminal to the equipment frame.

The **WARNING** heading in a manual explains dangers that might result in personal injury or death. Always read the associated information very carefully before performing the indicated procedure.

The **CAUTION** heading in a manual explains hazards that could damage the instrument. Such damage may invalidate the warranty.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits, including the power transformer, test leads, and input jacks, must be purchased from Keithley Instruments. Standard fuses, with applicable national safety approvals, may be used if the rating and type are the same. Other components that are not safety related may be purchased from other suppliers as long as they are equivalent to the original component. (Note that selected parts should be purchased only through Keithley Instruments to maintain accuracy and functionality of the product.) If you are unsure about the applicability of a replacement component, call a Keithley Instruments office for information.

To clean an instrument, use a damp cloth or mild, water based cleaner. Clean the exterior of the instrument only. Do not apply cleaner directly to the instrument or allow liquids to enter or spill on the instrument. Products that consist of a circuit board with no case or chassis (e.g., data acquisition board for installation into a computer) should never require cleaning if handled according to instructions. If the board becomes contaminated and operation is affected, the board should be returned to the factory for proper cleaning/servicing.

Table of Contents

Section	Topic	Page
1	**Keithley Command Compatible Functions**	1-1
	Introduction	1-2
	Using Keithley Command Compatible functions	1-2
	Microsoft® Visual Basic (Version 6.0)	1-2
	Microsoft Visual C/C++	1-4
	Keithley Command Compatible function reference	1-5
	GPIBBOARDPRESENT	1-5
	BOARDSELECT	1-5
	DMACHANNEL	1-5
	ENTER	1-5
	FEATURE	1-6
	INITIALIZE	1-6
	LISTENERPRESENT	1-7
	PPOLL	1-7
	RARRAY	1-7
	RECEIVE	1-8
	SEND	1-8
	SETINPUTEOS	1-8
	SETOUTPUTEOS	1-8
	SETPORT	1-9
	SETTIMEOUT	1-9
	SPOLL	1-9
	SRQ	1-10
	TARRAY	1-10
	TRANSMIT	1-10
	WAITSRQDEVICE	1-12
2	**NI Command Compatible Functions**	2-1
	Introduction	2-3
	Using NI Command Compatible functions	2-3
	Microsoft Visual Basic (Version 6.0)	2-3
	Microsoft Visual C/C++	2-5
	Overview of NI command compatible functions	2-5
	IEEE 488 device-level functions	2-5
	IEEE 488 board-level functions	2-6
	IEEE 488.2 functions	2-7
	Data Types	2-8
	NI command compatible function reference	2-9
	ibask	2-9
	ibbna	2-11
	ibcac	2-11
	ibclr	2-12
	ibcmd	2-12
	ibcmda	2-13
	ibconfig	2-14
	ibdev	2-16
	ibdma	2-17
	ibeot	2-17
	ibeos	2-17
	ibfind	2-18
	ibgts	2-19

Section	Topic	Page

ibist 2-19
iblines 2-20
ibln 2-20
ibloc 2-21
ibonl 2-22
ibnotify 2-22
ibpad 2-24
ibsad 2-24
ibpct 2-24
ibppc 2-25
ibrd 2-25
ibrda 2-26
ibrdf 2-27
ibrpp 2-28
ibrsc 2-28
ibrsp 2-29
ibrsv 2-29
ibsic 2-30
ibsre 2-30
ibstop 2-31
ibtmo 2-31
ibtrg 2-32
ibwait 2-32
ibwrt 2-33
ibwrta 2-34
ibwrtf 2-35
Multi-device functions 2-36
AllSpoll 2-36
DevClear 2-36
DevClearList 2-36
EnableLocal 2-37
EnableRemote 2-37
FindLstn 2-38
FindRQS 2-38
PassControl 2-38
PPoll 2-39
PPollConfig 2-39
PPollUnConfig 2-40
RcvRespMsg 2-40
ReadStatusByte 2-41
Receive 2-41
ReceiveSetup 2-42
ResetSys 2-42
Send 2-43
SendCmds 2-43
SendDataBytes 2-43
SendList 2-44
SendIFC 2-45
SendLLO 2-45
SendSetup 2-45
SetRWLS 2-46
TestSRQ 2-46
TestSys 2-46
Trigger 2-47
TriggerList 2-47
WaitSRQ 2-48

Appendix	Topic	Page

A **Status/Error Codes** A-1
NI command compatible status codes A-2
NI command compatible function error codes A-3

List of Figures

Section	Figure	Title	Page
1	Figure 1-1	Open Project dialog box	1-2
2	Figure 2-1	Open Project dialog box	2-3

This page left blank intentionally.

List of Tables

Section	Table	Title	Page
1	Table 1-1	FEATURE parameters	1-6
	Table 1-2	TRANSMIT command string parameters	1-11
2	Table 2-1	IEEE 488 device-level functions	2-5
	Table 2-2	IEEE 488 board-level functions	2-6
	Table 2-3	IEEE 488.2 functions	2-7
	Table 2-4	Data types	2-8
	Table 2-5	ibask board configuration parameter options	2-9
	Table 2-6	ibask device configuration parameter options	2-10
	Table 2-7	Board configuration parameter options	2-14
	Table 2-8	Device configuration parameter options	2-15
	Table 2-9	EOS mode V value	2-18
	Table 2-10	iblines	2-20
	Table 2-11	GPIB event codes for mask	2-23
	Table 2-12	Callback description (for ibnotify)	2-23
	Table 2-13	ibtmo timeout	2-31
	Table 2-14	ibwait valid mask codes	2-33

Appendix	Table	Title	Page
A	Table A-1	NI command compatible status codes	A-2
	Table A-2	NI command compatible function error codes	A-3

This page left blank intentionally.

Section 1

Keithley Command Compatible Functions

In this section:

Topic	Page
Introduction	1-2
Using Keithley Command Compatible functions	1-2
Microsoft® Visual Basic (Version 6.0)	1-2
Microsoft Visual C/C++	1-4
Keithley Command Compatible function reference	1-5
GPIBBOARDPRESENT	1-5
BOARDSELECT	1-5
DMACHANNEL	1-5
ENTER	1-5
FEATURE	1-6
INITIALIZE	1-6
LISTENERPRESENT	1-7
PPOLL	1-7
RARRAY	1-7
RECEIVE	1-8
SEND	1-8
SETINPUTEOS	1-8
SETOUTPUTEOS	1-8
SETPORT	1-9
SETTIMEOUT	1-9
SPOLL	1-9
SRQ	1-10
TARRAY	1-10
TRANSMIT	1-10
WAITSRQDEVICE	1-12

Introduction

This section contains information about Keithley Command Compatible Functions. Refer to Section 2 for information on the National Instruments™ (NI)[1] command compatible functions.

NOTE *Refer to Section 2 for NI Command Compatible Functions.*

If you have any questions after reviewing this information, please contact your local Keithley Instruments representative or call one of our Applications Engineers at 1-800-KEITHLEY (US only) or visit our website at www.keithley.com.

Using Keithley Command Compatible functions

Microsoft® Visual Basic (Version 6.0)

To create a Windows® XP/2000 Keithley Command Compatible application using the API and Microsoft Visual Basic, follow these steps:

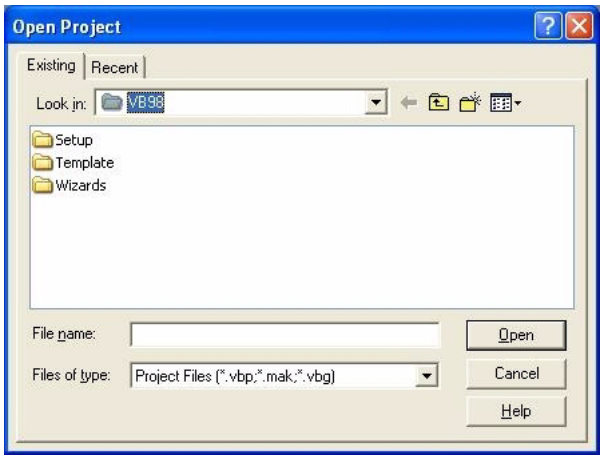
Step 1: Enter Visual Basic and open or create a project to use the Keithley Command Compatible functions

To create a new project, select **New Project** from the **File** menu.

To use an existing project:

1. Open the file by selecting **Open Project** from the **File** menu. The **Open Project** dialog box appears (Figure 1-1).

Figure 1-1
Open Project dialog box



2. Load the project by finding and double-clicking the project file name in the applicable directory.

Step 2: Include function declarations and constants file (IEEEVB.BAS)

If it is not already included in the project, add the **IEEEVB.BAS** file as a module to your project. All Keithley Command Compatible function declarations and constants are contained in this file. These function declarations and constants are used to develop user self-measurement applications.

1. National Instruments™ and NI are trademarks of the National Instruments Corporation.

Step 3: Design the application interface

Add elements, such as a command button, list box, or text box, etc., on the Visual Basic form used to design the interface. These elements are standard controls from the Visual Basic Toolbox. To place a needed control on the form:

1. Select the needed control from the **Toolbox**.
2. Draw the control on the form. Alternatively, to place the default-sized control on the form, click the form. Use the **Select Objects** tool to reposition or resize controls.

Step 4: Set control properties

Set control properties from the properties list. To view the properties list, select the desired control and do one of the following:

- Press **F4**
- Select the **Properties** command in the **View** menu

  or

- Click the **Properties** button on the Toolbar.

Step 5: Write the event codes

The event codes define the action desired when an event occurs. To write the event codes:

1. Double-click the control or form needing event code (the code module will appear).
2. Add new code as needed. All functions that are declared in **IEEEVB.BAS** can be called to perform data acquisition operations (refer to Keithley Command Compatible function reference).

Step 6: Run your application

To run the application, either:

- Press **F5**
- Select **Start** from the **Run** menu

  or

- Click the **Start** icon on the Toolbar

Microsoft Visual C/C++

To create a Windows XP/2000 Keithley command compatible library application using the Keithley Command Compatible function library (which is CEC command-compatible) and Microsoft Visual C/C++, follow these steps:

Step 1: Enter Visual C/C++ and open or create a project in which you wish to use Keithley Command Compatible functions

NOTE *The project can be a new or existing one.*

Step 2: Include function declarations and constants file (IEEE-C.H)

Include **IEEE-C.H** in the C/C++ source files that call Keithley Command Compatible functions by adding the following statement in the source file:

```
#include "IEEE-C.H"
```

NOTE *Keithley Command Compatible function declarations and constants are contained in **IEEE-C.H**. Use the functions and constants to develop user self data-acquisition applications.*

Step 3: Build your application

1. Set suitable compile and link options.
2. Select **Build** from the **Build** menu (Visual C/C++ 4.0 and higher).
3. Remember to link the Keithley Command Compatible library **ieee_32m.lib**.

Keithley Command Compatible function reference

This section contains a detailed description of Keithley Command Compatible library functions, including the compatible library data types and function reference. The following functions are arranged alphabetically:

GPIBBOARDPRESENT

Description This function checks if a GPIB board is present in the GPIB system.

Syntax **Microsoft C/C++ and Borland C++**

```
char gpib_board_present(void)
```

Visual Basic

```
GpibBoardPresent( ) As Long
```

Return Value 0: if no GPIB is installed

1: if a GPIB board is installed

BOARDSELECT

Description This function selects a board to be the active board.

Syntax **Microsoft C/C++ and Borland C++**

```
void boardselect (long int bd)
```

Visual Basic

```
call boardselect (ByVal board As Long)
```

Parameters **board**: the board number. The valid value is from 0 to 3

DMACHANNEL

Description This function sets the DMA channel. This function is ignored for the Model KPXI-488.

Syntax **Microsoft C/C++ and Borland C++**

```
void dmachannel (long int c)
```

Visual Basic

```
call dmachannel (ByVal chan As Long)
```

Parameters **chan**: DMA channel number

ENTER

Description This function reads data from a specified device.

Syntax **Microsoft C/C++ and Borland C++**

```
long int enter (char *buf, unsigned long maxlen,
  unsigned long *len, long int addr,
  long int  *status)
```

Visual Basic

```
call enter(buf As String, maxlen As Integer,
  len As Integer, addr As Integer, status As Integer)
```

Parameters **buf**: the buffer storing the received data
maxlen: the maximum bytes of data to receive. The valid value is from 0 to 65535
len:returns the actual number of received data bytes
addr: the GPIB address of the Talker
status: 0: read data successfully; 8: timeout error

FEATURE

Description This function returns the GPIB board settings or hardware features.

Syntax **Microsoft C/C++ and Borland C++**

```
long int feature (long int f)
```

Visual Basic

```
GPIBFeature (ByVal f As Long) As Long
```

Parameters **f**: the feature or setting information desired. Valid FEATURE values are contained in Table 1-1.

Table 1-1
FEATURE parameters

Feature (Constants)	Features (Values)	Returned Information
IEEEListener	0	Checking if ListenerPresent function is supported by the GPIB board; this information value is always 1.
IEEEIOBASE	100	the I/O base address of the board
IEEETIMEOUT	200	the I/O timeout setting of the board
IEEEINPUTEOS	201	the current setting of the input EOS character
IEEEOUTPUTEOS1	202	the current setting of the output EOS character 1
IEEEOUTPUTEOS2	203	the current setting of output EOS character 2
IEEEBOARDSELECT	204	the current active board number

Return Value The value of the feature or setting

INITIALIZE

Description This function opens and initializes a GPIB board.

Syntax **Microsoft C/C++ and Borland C++**

```
void initialize (long int addr,
  long int level)
```

Visual Basic

```
call initialize (ByVal addr As Long,
  ByVal level As Long)
```

Parameters **addr** : GPIB address assigned to the board
level: 0: specifies the board as a system controller
1: specifies the board as a device

LISTENERPRESENT

Description This function checks if a listener is present on the GPIB system.

Syntax **Microsoft C/C++ and Borland C++**

```
char listener_present(long int addr)
```

Visual Basic

```
ListenerPresent (ByVal addr As Long) As Long
```

Parameters **addr**: the listener address to check

Return Value 0: the specified listener is not present

1: the specified listener is on the GPIB system

PPOLL

Description This function performs a parallel poll and reads the status of devices.

Syntax **Microsoft C/C++ and Borland C++**

```
int ppoll (char  *poll)
```

Visual Basic

```
call ppoll(poll As Integer)
```

Parameters **poll**: returned parallel polling status

RARRAY

Description This function receives a block of binary data (up to 64K) from a device defined as the talker. The GPIB addressing must be performed using the **transmit** function.

Syntax **Microsoft C/C++ and Borland C++**

```
long int rarray (void *buf,
  unsigned long count, unsigned long *len,
  long int *status)
```

Visual Basic

```
call rarray(buf As Variant, ByVal count As Long,
  l As Integer, status As Integer)
```

Parameters **buf**: the buffer storing the received binary data
count: the maximum data bytes. The valid value is 0 to 65535
len: returns the actual number of received data bytes

Return Value 0: read data successfully
8: timeout error
32: data transfer terminated with EOI

RECEIVE

Description This function reads data from a specified device, but does not address a talker. The GPIB addressing must be performed using the **transmit** function.

Syntax **Microsoft C/C++ and Borland C++**

```
long int receive (char *buf, unsigned long maxlen,
  unsigned long *len, long int  *status)
```

Visual Basic

```
call receive (buf As String, maxlen As Integer,
  len As Integer, status As Integer)
```

Parameters **buf**: the buffer storing the received data
maxlen: sets maximum bytes of data to receive
len: returns the actual number of received data bytes

Return Value 0: read data successfully
8: timeout error

SEND

Description This function sends commands to a specified GPIB device.

Syntax **Microsoft C/C++ and Borland C++**

```
long int send (long int addr, char *buf,
  unsigned long maxlen, long int *status)
```

Visual Basic

```
call send(addr As Integer, buf As String,
  status As Integer)
```

Parameters **addr**: the listener address
buf: the buffer storing the data to send
maxlen: sets the maximum number of data bytes to send

Return Value 0: data sent successfully

8: timeout error

SETINPUTEOS

Description This function sets the terminating character for input data transfer.

Syntax **Microsoft C/C++ and Borland C++**

```
void setinputEOS (long int eos_c)
```

Visual Basic

```
call setinputEOS (ByVal eos_c As Long)
```

Parameters **eos_c**: the terminating character for input data transfer

SETOUTPUTEOS

Description This function sets the terminating characters for output data transfer.

Syntax **Microsoft C/C++ and Borland C++**

```
void setoutputEOS (long int e1, long int e2)
```

Visual Basic

```
call setoutputEOS (ByVal e1 As Long, ByVal e2 As Long)
```

Parameters **e1**: the first terminating character for output data transfer
e2: the second terminating character for output data transfer

SETPORT

Description This function sets the I/O address of a GPIB board. This function is not used for the Model KPXI-488.

Syntax **Microsoft C/C++ and Borland C++**

```
void setport (long int bd, unsigned io)
```

Visual Basic

```
call setport (ByVal bd As Long, ByVal io As Long)
```

Parameters **bd**: the board number
io: the I/O base address set to the device

SETTIMEOUT

Description This function sets the timeout period. The timeout period is the maximum duration allowed for a read/write operation.

Syntax **Microsoft C/C++ and Borland C++**

```
void settimeout (unsigned long int timeout)
```

Visual Basic

```
call settimeout (ByVal timeout As Long)
```

Parameters **timeout**: the timeout value in milli-seconds (msec)

SPOLL

Description This function performs serial polling and a read of the specified device's status.

Syntax **Microsoft C/C++ and Borland C++**

```
long int spoll (long int addr, char *poll,
  long int *status)
```

Visual Basic

```
call spoll(ByVal addr As Integer, poll As Integer,
  status As Integer)
```

Parameters **addr**: the address of the device to poll
poll: returns result of serial polling

Return Value 0: data sent successfully

8: timeout error

SRQ

Description This function checks if a device is requesting service.

Syntax **Microsoft C/C++ and Borland C++**

```
char srq(void)
```

Visual Basic

```
srq ( ) As Long
```

Return Value 0: the device is not requesting service
1: the device is requesting service

TARRAY

Description This function sends a block of binary data from memory to the devices defined as listeners. The GPIB addressing must be performed using the **transmit** function.

Syntax **Microsoft C/C++ and Borland C++**

```
long int tarray (void *buf,
  unsigned long count, long int eoi,
  long int  *status)
```

Visual Basic

```
call tarray (buf as variant, ByVal count As Long,
  ByVal eoi As Integer, status As Integer)
```

Parameters **buf**: the buffer storing the data to send
count: the maximum number of data bytes to be transmitted
eoi: enable or disable EOI mode of the device. 0 = disable EOI; 1 = enable EOI

Return Value 0: read data successfully
8: timeout error
32: data transfer terminated with EOI

TRANSMIT

Description This function sends GPIB commands and data according to a specified string composed a series of GPIB commands and data.

Syntax **Microsoft C/C++ and Borland C++**

```
long int transmit (char  * cmd,
  unsigned maxlen, long int  * status);
```

Visual Basic

```
call transmit(cmd As String, status As Integer)
```

Parameters **cmd**: the buffer containing the command string and data to send.
The valid **cmd** string values are contained in Table 1-2.
maxlen: the maximum number of command string bytes to send.

Return Value **status**:
0: sent command and data successfully
1: illegal command syntax
8: timeout error
16: unknown command
32: data transfer terminated with EOI

Table 1-2
TRANSMIT command string parameters

Commands	Description	Example
LISTEN	Sets the addresses of the listeners. The values following **LISTEN** are the GPIB addresses of the listeners.	"LISTEN 1 2 3" meaning: config devices whose GPIB address are 1, 2 and 3, as listeners.
TALK	Sets the address of the talker. The values following **TALK** are the GPIB addresses of the talker. There is only one talker at a time.	"TALK 0" meaning: config device whose GPIB address is 0, as talker.
SEC	Sets the second address of the talker or listener. This command should follow **TALK** or **LISTEN**.	"TALK 0 SEC 1" meaning: config device whose primary GPIB address is 0 and secondary address is 1 as talker.
UNT	Untalk.	"UNT"
UNL	Unlisten.	"UNL"
MTA	My Talk Address. Assigns the active GPIB board as the talker.	"MTA"
MLA	My Listen Address. Assigns the active GPIB board as the listener.	"MLA"
DATA	Starts the data part. Before the **DATA** command, the GPIB board has to be set as the talker. Strings are enclosed by quotes(') and sent as characters.	"DATA 'hello' 13 10"
END	Sends terminator characters. **DATA** command should be called before this command.	"DATA '*IDN?' END" meaning: send data message "*IDN?" and then send terminator bytes
REN	Remote Enable	"REN"
EOI	End-or-Identify. The data bytes following **EOI** are the last bytes to transmit. The last byte is sent with the **EOI** signal.	"DATA '*IDN?' EOI 10" meaning: send data message "*IDN?" and then send line feed with EOI signal.
GTL	Go To Local	"GTL"
SPE	Serial Poll Enable	"SPE"
SPD	Serial Poll Disable	"SPD"
PPC	Parallel Poll Configure	"PPC"
PPD	Parallel Poll Disable	"PPD"
PPU	Parallel Poll Unconfigure	"PPU"
DCL	Device Clear	"DCL"
LLO	Local Lockout	"LLO"
CMD	Starts **GPIB** command. The values followed by CMD are treated as GPIB command messages and sent as binary values.	"CMD 20" meaning: send GPIB command message, Device Clear (**DCL**).
GET	Group Execute Trigger	"GET"
SDC	Selected Device Clear	"SDC"
TCT	Take Control	"TCT"
IFC	Interface Clear	"IFC"

WAITSRQDEVICE

Description This function waits until a device is requesting service or a timeout error occurs.

Syntax **Microsoft C/C++ and Borland C++**

```
long int waitSRQDevice (long int addr,
  char *poll, long int *status)
```

Parameters **addr**: the device address
poll: the returned poll status
status: indicates whether or not a serial poll was performed

Section 2

NI Command Compatible Functions

In this section:

Topic	Page
Introduction	2-3
Using NI Command Compatible functions	2-3
Microsoft Visual Basic	2-3
Microsoft Visual C/C++	2-5
Overview of NI command compatible functions	2-5
IEEE 488 device-level functions	2-5
IEEE 488 board-level functions	2-6
IEEE 488.2 functions	2-7
Data Types	2-8
NI command compatible function reference	2-9
ibask	2-9
ibbna	2-11
ibcac	2-11
ibclr	2-12
ibcmd	2-12
ibcmda	2-13
ibconfig	2-14
ibdev	2-16
ibdma	2-17
ibeot	2-17
ibeos	2-17
ibfind	2-18
ibgts	2-19
ibist	2-19
iblines	2-20
ibln	2-20
ibloc	2-21
ibonl	2-22
ibnotify	2-22
ibpad	2-24
ibsad	2-24
ibpct	2-24
ibppc	2-25
ibrd	2-25
ibrda	2-26
ibrdf	2-27
ibrpp	2-28
ibrsc	2-28

Topic (continued)	Page

ibrsp 2-29
ibrsv 2-29
ibsic 2-30
ibsre 2-30
ibstop 2-31
ibtmo 2-31
ibtrg 2-32
ibwait 2-32
ibwrt 2-33
ibwrta 2-34
ibwrtf 2-35

Multi-device functions 2-36
AllSpoll 2-36
DevClear 2-36
DevClearList 2-36
EnableLocal 2-37
EnableRemote 2-37
FindLstn 2-38
FindRQS 2-38
PassControl 2-38
PPoll 2-39
PPollConfig 2-39
PPollUnConfig 2-40
RcvRespMsg 2-40
ReadStatusByte 2-41
Receive 2-41
ReceiveSetup 2-42
ResetSys 2-42
Send 2-43
SendCmds 2-43
SendDataBytes 2-43
SendList 2-44
SendIFC 2-45
SendLLO 2-45
SendSetup 2-45
SetRWLS 2-46
TestSRQ 2-46
TestSys 2-46
Trigger 2-47
TriggerList 2-47
WaitSRQ 2-48

Introduction

This section contains information about the National Instruments™ (NI)[1] command compatible functions and provides information on using the functions, as well as a reference section containing syntax examples (C/C++, Visual Basic, etc.). Appendix A contains information on NI command compatible status codes and NI command compatible function error codes.

NOTE *Refer to Section 1 for Keithley Command Compatible Functions.*

If you have any questions after reviewing this information, please contact your local Keithley Instruments representative or call one of our Applications Engineers at 1-800-KEITHLEY (US only) or visit our website at www.keithley.com.

Using NI Command Compatible functions

This section provides the fundamentals of building Windows® XP/2000 applications using NI command compatible functions and either Microsoft® Visual Basic or Microsoft Visual C/C++.

Microsoft Visual Basic (Version 6.0)

To create an application with NI command compatible functions and Visual Basic, follow these steps:

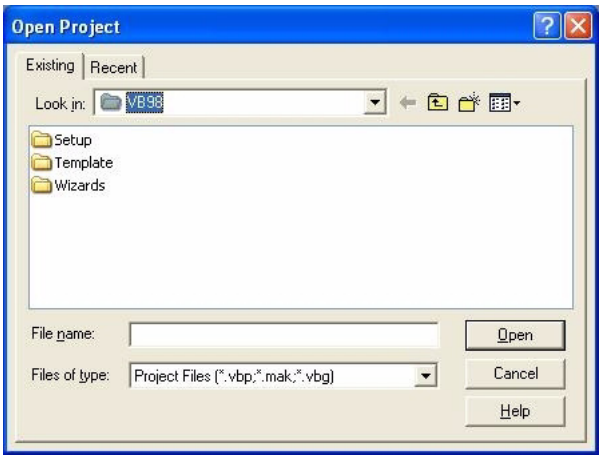
Step 1: Enter Visual Basic and open or create a project in which to use the NI command compatible functions

To create a new project, select **New Project** from the **File** menu.

To use an existing project:

1. Open the file by selecting **Open Project** from the **File** menu. The **Open Project** dialog box appears (Figure 2-1).

Figure 2-1
Open Project dialog box



2. Load the project by finding and double-clicking the project file name in the applicable directory.

1. National Instruments™ and NI are trademarks of the National Instruments Corporation.

Step 2: Include function declarations and constants file (GPIB.BAS)

If it is not already included in the project, add the **GPIB.BAS** file. All NI command compatible function declarations and constants are contained in this file. These function declarations and constants are used to develop applications.

Step 3: Design the application interface

Add elements, such as a command button, list box, or text box, etc., on the Visual Basic form used to design the interface. These elements are standard controls from the Visual Basic Toolbox. To place a needed control on the form:

1. Select the needed control from the **Toolbox**.
2. Draw the control on the form. Alternatively, to place the default-sized control on the form, click the form. Use the **Select Objects** tool to reposition or resize controls.

Step 4: Set control properties

Set control properties from the properties list. To view the properties list, select the desired control and do one of the following:

- Press **F4**
- Select the **Properties** command in the **View** menu

  or

- Click the **Properties** button on the Toolbar.

Step 5: Write the event codes

The event codes define the action desired when an event occurs. To write the event codes:

1. Double-click the control or form needing event code (the code module will appear).
2. Add new code as needed. All functions that are declared in **GPIB.BAS** can be called to perform operations (refer to Table 2-1 through Table 2-4).

Step 6: Run your application

To run the application, either:

- Press **F5**
- Select **Start** from the **Run** menu

  or

- Click the **Start** icon on the Toolbar

Microsoft Visual C/C++

To create an application with NI command compatible functions and Microsoft Visual C/C++, follow these steps:

Step 1: Enter Microsoft Visual C/C++ and open or create a project in which you wish to use NI command compatible functions

NOTE *The project can be a new or existing one.*

Step 2: Include the function declarations and constants file (GPIB.H)

Include **GPIB.H** in the C/C++ source files that call NI command compatible functions by adding the following statement in the source file:

```
#include "GPIB.H"
```

NOTE *NI command compatible function declarations and constants are contained in **GPIB.H**. Use the functions and constants to develop user self data-acquisition applications.*

Step 3: Build your application as follows:

1. Set suitable compile and link options.
2. Select **Build** from the **Build** menu (Visual C/C++ 4.0 and higher).
3. Remember to link the NI command compatible import library **GPIB-32.lib**.

Overview of NI command compatible functions

The NI command compatible functions are grouped into three classes:

- IEEE 488 device-level functions
- IEEE 488 board-level functions
- IEEE 488.2 functions

IEEE 488 device-level functions

Table 2-1 contains IEEE 488 device-level functions.

Table 2-1
IEEE 488 device-level functions

Function	Description
ibask	Returns the current value of the selected configuration item.
ibbna	Assigns the access board of the designated device.
ibclr	Sends the GPIB Selected Device Clear (SDC) message to the designated device.
ibconfig	Sets the value of the selected configuration item.
ibdev	Opens and initializes a device descriptor.
ibeos	Configures the EOS termination mode or character.
ibeot	Enables or disables the action that is setting GPIB EOI line to enable while the I/O operation is completed.
ibln	Checks if there is an available device on the bus.
ibloc	Sets the device to local control mode.
ibonl	Sets the device online or offline.
ibpad	Sets a device primary GPIB address.

Table 2-1 (continued)
IEEE 488 device-level functions

Function	Description
ibpct	Passes Controller-in-Charge (CIC) status to another GPIB device that has controller capability.
ibppc	Configures parallel polling.
ibrd	Reads data from a device to the indicated buffer.
ibrda	Reads data from a device to the indicated buffer asynchronously.
ibrdf	Reads data from a device to a file.
ibrdi	Reads data from a device to the indicated buffer.
ibrdia	Reads data from a device to the indicated buffer asynchronously.
ibrpp	Performs a parallel polling.
ibrsp	Performs a sequential polling.
ibsad	Sets or disables a device secondary GPIB address.
ibstop	Stops the asynchronous I/O operation.
ibtmo	Sets the board or device timeout period.
ibtrg	Sends the Group Execute Trigger (GET) message to device.
ibwait	Monitors event(s) until one or more events that are described by mask or delay operating occur.
ibwrt	Writes data from a buffer to a device.
ibwrta	Writes data from a buffer to a device asynchronously.
ibwrtf	Writes data from a file to a device.

IEEE 488 board-level functions

Table 2-2 contains IEEE 488 board-level functions.

Table 2-2
IEEE 488 board-level functions

Function	Description
ibask	Returns the current value of the selected configuration item.
ibcac	Sets the assigned GPIB board to be the active controller by setting the ATN line to enable.
ibcmd	Sends GPIB commands.
ibcmda	Sends GPIB commands asynchronously.
ibconfig	Sets the value of the selected configuration item.
ibdma	Enables or disables DMA.
ibeos	Configures the EOS termination mode or character.
ibeot	Enables or disables the action that is setting GPIB EOI line to enable while the I/O operation is completed.
ibfind	Opens and initializes the GPIB board descriptor.
ibgts	Sets the board from active control status to standby control status.
ibist	Sets or clears the board individual status (ist) bit for parallel polling.
iblines	Returns the GPIB control lines status.
ibln	Checks if there is an available device on the bus.
ibloc	Sets the device to local control mode.
ibonl	Sets the device online or offline.
ibpad	Sets the device's primary GPIB address.
ibppc	Configures parallel polling.
ibrd	Reads data from a device to the indicated buffer.
ibrda	Reads data from a device to the indicated buffer asynchronously.
ibrdf	Reads data from a device to a file.
ibrdi	Reads data from a device to the indicated buffer.
ibrdia	Reads data from a device to the indicated buffer asynchronously.

Table 2-2 (continued)
IEEE 488 board-level functions

Function	Description
ibrpp	Performs parallel polling.
ibrsc	Sends Interface Clear (IFC) message or Remote Enable (REN) message to request or release the System Control.
ibrsv	Requests service and changes the status byte of the sequential polling.
ibsad	Sets or disables a board secondary GPIB address.
ibsic	Sets the GPIB interface's clear (IFC) line to enable at least 100ns if the GPIB interface is system controller.
ibsre	Sets or clears the Remote Enable (REN) line.
ibstop	Stops the asynchronous I/O operation.
ibtmo	Sets the board timeout period.
ibwait	Monitors event(s) until one or more events that are described by mask or delay operating occur.
ibwrt	Writes data from a buffer to a device.
ibwrta	Writes data from a buffer to a device asynchronously.
ibwrtf	Writes data from a file to a device.

IEEE 488.2 functions

Table 2-3 contains IEEE 488.2 Functions.

Table 2-3
IEEE 488.2 functions

Function	Description
AllSpoll	Polls one or more devices sequentially.
DevClear	Sends the **Selected Device Clear (SDC)** GPIB message to clear the selected device.
DevClearList	Clears multiple devices.
EnableLocal	Sends **Go To Local (GTL) GPIB message** to multiple devices to allow local operation of the devices.
EnableRemote	Sets **Remote Enable (REN) line** to allow remote programming of devices.
FindLstn	Finds listening devices on the GPIB bus.
FindRQS	Sequentially polls devices to determine which device is requesting service.
PassControl	Sends **Take Control (TCT) GPIB message**. Sending this message allows control to pass to another GPIB device having control capability.
PPoll	Performs parallel polling once.
PPollConfig	Controls or releases GPIB data line to configure the device to respond to parallel polling.
PPollUnconfig	Removes configuration that allows device to respond to parallel polling.
RcvRespMsg	Reads data from a device.
ReadStatusByte	Sequentially polls a device.
Receive	Reads data bytes from a device and then stores them in the assigned buffer.
ReceiveSetup	Configures device and interface to a talker and a receiver.
ResetSys	Resets and initializes the devices.
Send	Writes data bytes from the buffer to the device.
SendCmds	Sends GPIB commands.
SendDataBytes	Sends data from the buffer to the device.

Table 2-3 (continued)
IEEE 488.2 functions

Function	Description
SendIFC	Sends **Interface Clear command** to reset GPIB.
SendList	Sends data bytes to multiple GPIB devices.
SendLLO	Sends **Local Lockout (LLO)** message to all devices.
SendSetup	Configures device to receive data.
SetRWLS	Configures device to lockout status of remote control mode.
TestSRQ	Detects current status of the GPIB Service Request (SRQ) line.
TestSys	Causes devices to process self tests. TestSys sends the "TST?" message to the devices.
Trigger	Sends Group Execute Trigger (GET) GPIB message to a device.
TriggerList	Sends Group Execute Trigger (GET) GPIB message to multiple devices.
WaitSRQ	Waits until the device controls the GPIB SRQ line.

Data Types

GPIB.BAS defines some data types. The defined data types are used by the NI command compatible function library and are suggested for your applications. Table 2-4 shows the names, ranges, and the corresponding data types in C/C++, Visual Basic and Delphi. These data types are not defined in either GPIB.BAS or GPIB.PAS (they are just listed for reference).

Table 2-4
Data types

Type Name	Description	Range	Type		
			C/C++ (32-bit compiler)	Visual Basic	Byte
U8	8-bit ASCII character	0 ~ 255	Unsigned character	Byte	Small Integer
I16	16-bit signed integer	-32768 ~ 32767	Short	Integer	Word
U16 Addr4882_t	16-bit unsigned integer	0 ~ 65535	Unsigned short	Not supported. Placed by I16	Long Integer
I32 ssize_t	32-bit signed integer	-2147483648 ~ 2147483647	Long	Long	Cardinal
U32 size_t	32-bit unsigned integer	0 ~ 4294967295	Unsigned long	Not supported. Placed by I32	Single
F32	32-bit single-precision floating-point	-3.402823E38 ~ 3.402823E38	Float	Single	Double
F64	64-bit double-precision floating-point	-1.797683134862315E308 ~ 1.797683134862315E309	Double	Double	Double

NI command compatible function reference

Use this section as a function reference for NI command compatible functions. Refer to Section 1 for information on Keithley Command Compatible Functions.

ibask

Description This command returns the current value of the selected configuration item.

Support Level Board / Device Level

Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

```
ibask (ByVal ud As Integer, ByVal opt As Integer,
  rval As Integer) As Integer
```

-or-

```
call ibask (ByVal ud As Integer, ByVal opt As
  Integer, rval As Integer)
```

Parameters **ud**: board or device unit descriptor

option: the configuration item value will be returned (refer to valid options as shown in Table 2-5 and Table 2-6)

value: the current value of the selected configuration item returned

Return Value The value of the **ibsta**

Error Codes EARG, ECAP, EDVR

Table 2-5
ibask board configuration parameter options

Options (Constants)	Options (Value)	Returned Information
ibaPAD	0x0001	The board current primary address.
ibaSAD	0x0002	The board current secondary address.
ibaTMO	0x0003	The board current I/O timeout.
ibaEOT	0x0004	0: After termination of the writing operation, the **GPIB EOI** line is not set to enable.
		1: After termination of the writing operation, the **GPIB EOI** line is set to enable.
ibaPPC	0x0005	The current parallel polling configuration board setting.
ibaAUTOPOLL	0x0007	0: Disable the automatic sequential polling.
		1: Enable the automatic sequential polling.
ibaCICPROT	0x0008	0: Disable the CIC protocol.
		1: Enable the CIC protocol.
ibaIRQ	0x0009	0: Disable the Interrupts.
		1: Enable the Interrupts.
ibaSC	0x000A	0: The board is not the GPIB System Controller.
		1: The board is the GPIB System Controller.

Table 2-5 (continued)
ibask board configuration parameter options

Options (Constants)	Options (Value)	Returned Information
ibaSRE	0x000B	0: While the board becomes the System Controller, the **GPIB REN** line is not set to enable automatically.
		1: While the board becomes the System Controller, the **GPIB REN** line is set to enable automatically.
ibaEOSrd	0x000C	0: Ignore the EOS character during reading.
		1:The reading is stopped while the EOS character is read.
ibaEOSwrt	0x000D	0: The EOI line is not set to enable while the EOS character is sent during writing.
		1: The EOI line is set to enable while the EOS character is sent during writing.
ibaEOScmp	0x000E	0: Compare all EOS with 7 bits.
		1: Compare all EOS with 8 bits.
ibaEOSchar	0x000F	The board current EOS character.
ibaPP2	0x0010	0: The board in the PP1 mode (Remote Parallel Polling Configuration).
		1: The board in the PP2 mode (Local Parallel Polling Configuration).
ibaTIMING	0x0011	The current board bus timing.
		1: Normal timing (2μs T1 delay).
		2: High speed timing (500ns T1 delay).
		3: Very high speed timing (350ns T1 delay).
ibaDMA	0x0012	0: DMA is not used for GPIB transfer.
		1: DMA is used for GPIB transfer.
ibaSpollBit	0x0016	0: Disable the SPOLL bit of the **ibsta**.
		1: Enable the SPOLL bit of the **ibsta**.
ibaSendLLO	0x0017	0: The **GPIB LLO** command is not sent while the device is connected by **ibfind** or **ibdev**.
		1: The **GPIB LLO** command is sent while the device is connected by **ibfind** or **ibdev**.
ibaPPollTime	0x0019	0: Use standard continue time (2μs) during parallel polling.
		1~17: Use different continue time during parallel polling; time corresponds to the **ibtmo** timing value.
ibaEndBitIsNormal	al0x001A	0: The END bit of the **ibsta** is set only when the EOI or both EOI and EOS are received; if the EOS is received without EOI, the END bit is not set.
		1: When EOI, EOS, or both EOI and EOS is received, the END bit is set.
ibaist	0x0020	The individual status (ist) bit of the board.
ibaRsv	0x0021	The current status word of the sequential polling of the board.

Table 2-6
ibask device configuration parameter options

Options (Constants)	Options (Values)	Returned Information
ibaPAD	0x0001	The current device primary address.
ibaSAD	0x0002	The current device secondary address.
ibaTMO	0x0003	The current device I/O timeout.
ibaEOT	0x0004	0: After termination of the writing operation, the **GPIB EOI line** is not set to enable.
		1: After termination of the writing operation, the **GPIB EOI line** is set to enable.

Table 2-6 (continued)
ibask device configuration parameter options

Options (Constants)	Options (Values)	Returned Information
ibaREADDR	0x0006	0: The unnecessary addressing is not operated during the device-level writing or reading.
		1: The addressing is operated continuously during the device-level writing or reading.
ibaEOSrd	0x000C	0: Ignore the **EOS character** during reading.
		1: The reading is stopped while the **EOS character** is read.
ibaEOSwrt	0x000D	0: The EOI line is not set to enable when the **EOS character** is sent during writing.
		1: The EOI line is set to enable when the **EOS character** is sent during writing.
ibaEOScmp	0x000E	0: Compare all EOS with 7 bits.
		1: Compare all EOS with 8 bits.
ibaEOSchar	0x000F	The board current **EOS character**.
ibaSPollTime	0x0018	The waiting time of the driver for the sequential polling response. The time is represented by ibtmo timing value.
ibaEndBitIsNormal	al0x001A	0: The END bit of the **ibsta** is set only when the **EOI** or both **EOI** and **EOS** are received; if the **EOS** is received without **EOI**, the END bit is not set.
		1: When the **EOI**, **EOS**, or both **EOI** and **EOS** is received, the **END bit** is set.
ibaBNA	0x0200	The index of the GPIB access board for the assigned device descriptor.

ibbna

Description This command assigns the device unit descriptor to the boardname.

Support Level Device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibbna (ByVal ud As Integer, ByVal udname As String)
  As Integer
```

- or -

```
call ibbna (ByVal ud As Integer, ByVal udname As String)
```

Parameters **ud**: device unit descriptor

board_name: the access board name; gpib0 for example

Return Value The value of the **ibsta**

Error Codes EARG, ECAP, EDVR, EOIP, ENEB

ibcac

Description This command sets the assigned GPIB board to be the active controller by setting the **ATN line** to enable. The GPIB board must be the CIC (controller in charge) before calling **ibcac**. Use **ibsic** to set the board as the CIC. The board can take control synchronously (1), asynchronously (2), or either (v). If either, the GPIB

board tries to create the ATN signal but does not terminate the data transfer (synchronous control is tried first). If this fails, the board takes asynchronous control by immediately creating the ATN signal without considering any current data transfer for asynchronous control.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibcac(int ud, int synchronous)
```

Visual Basic

```
idcac (ByVal ud As Integer, ByVal v As Integer) As
  Integer
```

- or -

```
call ibcac (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board unit descriptor

v: either synchronous or asynchronous control

0: asynchronously

1: synchronously

Return Value The value of the **ibsta**

Error Codes EARG, ECIC, EDVR, EOIP, ENEB

ibclr

Description This command sends the **GPIB Selected Device Clear** (SDC) message to the assigned device.

Support Level Device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
idclr (ByVal ud As Integer) As Integer
```

- or -

```
call ibclr (ByVal ud As Integer)
```

Parameters **ud**: device unit descriptor

Return Value The value of the **ibsta**

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB

ibcmd

Description This command sends GPIB commands. Command words are used to configure the GPIB status. **ibwrt** is used to send the device self-control command. To return the number of transferred command bytes in the global variable, use **ibcntl**.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibcmd (int ud, const void *cmd, long cnt)
```

Visual Basic

```
idcmd (ByVal ud As Integer, ByVal buf As String,
  ByVal cnt As Long) As Integer
```

- or -

```
call ibcmd (ByVal ud As Integer, ByVal buf As
  String)
```

Parameters **ud**: device unit descriptor

buf: the buffer contains the sent command string

cnt: the number of the command bytes; the command bytes that are to be sent

Return Value The value of the **ibsta**

Error Codes EARG, ECIC, EDVR, EOIP, ENEB, EABO, ENOL

ibcmda

Description This command sends GPIB commands asynchronously. Command words are used to configure the GPIB status and control GPIB devices. **ibwrt** is used to send the device self-control command. To return the number of transferred command bytes in the global variable, use **ibcntl**.

The design of the asynchronous I/O commands (**ibcmda**, **ibrda**, **ibwrta**) is that applications can perform other non-GPIB operations while the I/O is in progress. If asynchronous I/O has begun, later GPIB commands are strictly limited — any commands that would interfere with the I/O that is in progress are not allowed. If the I/O has completed, the application and the driver must be re-synchronized.

Use one of the following functions to re-synchronize:

ibwait: If the CMPL bit of the returned **ibsta** is set, the driver and application are re-synchronized.

ibnotify: If the **ibsta** value sent to the **ibnotify** callback contains CMPL, the driver and application are re-synchronized.

ibstop: The I/O is stopped, and the driver and application are re-synchronized.

ibonl: The I/O is stopped and the interface is reset; the driver and application are re-synchronized.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibcmda (int ud, const void *cmd, long cnt)
```

Visual Basic

```
idcmda (ByVal ud As Integer, ByVal buf As String,
  ByVal cnt As Long) As Integer
```

- or -

```
call ibcmda (ByVal ud As Integer, ByVal buf As
  String)
```

Parameters **ud**: device unit descriptor

buf: the buffer contains the sent command string

cnt: the number of the command bytes; the command bytes to be sent

Return Value The value of the **ibsta**

Error Codes EARG, ECIC, EDVR, EOIP, ENEB, EABO, ENOL

ibconfig

Description This command sets the value of the selected configuration item.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
idconfig (ByVal ud As Integer, ByVal opt As Integer,
  ByVal v As Integer) As Integer
```

- or -

```
call ibconfig (ByVal ud As Integer,
  ByVal opt As Integer, ByVal v As Integer)
```

Parameters **ud**: board or device unit descriptor

opt: the configuration item that needs to be changed (valid options are shown in Table 2-7 and Table 2-8)

v: the value of the configuration item that needs to be changed

Return Value The value of the **ibsta**

Error Codes EARG, ECAP, EDVR, EOIP

Table 2-7
Board configuration parameter options

Options (Constants)	Options (Value)	Valid Values
ibcPAD	0x0001	Set the board current primary address.
ibcSAD	0x0002	Set the board current secondary address.
ibcTMO	0x0003	Set the board current I/O timeout.
ibcEOT	0x0004	Set the data termination mode for writing.
ibcPPC	0x0005	Configure the board for parallel polling. Default: zero.
ibcAUTOPOLL	0x0007	0: Disable the automatic sequential polling.
		1: Enable the automatic sequential polling.
ibcSC	0x000A	Request or release system control. The same as **ibrsc**.
ibcSRE	0x000B	Control the **Remote Enable (REN) line**. The same as **ibsre**. Default: 0.
ibcEOSrd	0x000C	0: Ignore the EOS character during reading.
		1: The reading is stopped while the EOS character is read.
ibcEOSwrt	0x000D	0: The **EOI line** is not set to enable while the **EOS** character is sent during writing.
		1: The **EOI line** is set to enable while the **EOS** character is sent during writing.

Table 2-7 (continued)
Board configuration parameter options

Options (Constants)	Options (Value)	Valid Values
ibcEOScmp	0x000E	0: Compare all **EOS** with 7 bits.
		1: Compare all **EOS** with 8 bits.
ibcEOSchar	0x000F	Any 8-bit value. This byte becomes the new **EOS** character.
ibcPP2	0x0010	0: The board in the PP1 mode (Remote Parallel Polling Configuration).
		1: The board in the PP2 mode (Local Parallel Polling Configuration).
		Default: 0.
ibcTIMING	0x0011	0: Disable (Default).
		1: Normal timing (2μs T1 delay).
		2: High speed timing (500ns T1 delay).
		3: Very high speed timing (350ns T1 delay).
		The T1 delay is the GPIB source handshake timing.
ibcReadAdjust	0x0013	0: No byte swapping.
		1: Swap pairs of bytes during reading.
		Default: 0.
ibcWriteAdjust	0x0014	0: No byte swapping.
		1: Swap pairs of bytes during writing.
		Default: 0.
ibcSpollBit	0x0016	0: Disable the **SPOLL** bit of the **ibsta**.
		1: Enable the **SPOLL** bit of the **ibsta**.
		Default: 0.
ibcSendLLO	0x0017	0: The **GPIB LLO** command is not sent while the device is connected by **ibfind** or **ibdev**.
		1: The **GPIB LLO** command is sent while the device is connected by **ibfind** or **ibdev**.
		Default: 0.
ibcPPollTime	0x0019	0: Use standard continue time (2μs) during parallel polling.
		1 ~ 17: Select a different continue time during parallel polling; the time selected corresponds with the **ibtmo** timing value. Default: zero.
ibcEndBitIsNormal	al0x001A	0: While the EOS is received, the **END** bit of the **ibsta** is not set.
		1: While the EOS is received, the **END** bit of the **ibsta** is set. Default: 1
ibcist	0x0020	Set the individual status (**ist**) bit of the board.
ibcRsv	0x0021	Set the status byte of the board sequential polling. Default: 0.

Table 2-8
Device configuration parameter options

Options (Constants)	Options (Values)	Returned Information
ibcPAD	0x0001	Set the current device primary address.
ibcSAD	0x0002	Set the current device secondary address.
ibcTMO	0x0003	Set the current device I/O timeout.
ibcEOT	0x0004	Set the data termination mode for writing.
ibcREADDR	0x0006	0: Unnecessary addressing is not operated during device-level writing or reading.
		1: Addressing is operated continuously during the device-level writing or reading.
ibcEOSrd	0x000C	0: Ignore the **EOS** character during reading.
		1: The reading is stopped while the **EOS** character is read.

Table 2-8 (continued)
Device configuration parameter options

Options (Constants)	Options (Values)	Returned Information
ibcEOSwrt	0x000D	0: The **EOI line** is not set to enable while the **EOS** character is sent during writing.
		1: The **EOI line** is set to enable while the **EOS** character is sent during writing.
ibcEOScmp	0x000E	0: Compare all **EOS** with 7 bits.
		1: Compare all **EOS** with 8 bits.
ibcEOSchar	0x000F	Any eight-bit value. This byte becomes the new **EOS** character.
ibcSPollTime	0x0018	0 ~ 17: Set the waiting time of the driver for the sequential polling response. The time is represented by **ibtmo** timing value.
		Default: 11
ibcEndBitIsNormal	al0x001A	0: When the **EOS** is received, the **END** bit of the **ibsta** is not set.
		1: When the **EOS** is received, the **END** bit of the **ibsta** is set. Default: 1.

ibdev

Description This command opens and initializes a device descriptor. If **ibdev** cannot get a valid device descriptor, -1 is returned; the **ERR** bit of the **ibsta** and the **EDVR** bit of the **iberr** are set.

Support Level Device level

Syntax

Microsoft C/C++ and Borland C++

```
int ibdev (int board_index, int pad, int sad,
  int tmo, int send_eoi, int eosmode)
```

Visual Basic

```
ildev (ByVal bdid As Integer, ByVal pad As Integer,
  ByVal sad As Integer, ByVal tmo As Integer,
  ByVal eot As Integer, ByVal eos As Integer)
  As Integer
```

- or -

```
call ibdev (ByVal bdid As Integer, ByVal pad As
  Integer, ByVal sad As Integer, ByVal tmo As Integer,
  ByVal eot As Integer, ByVal eos As Integer,
  ud As Integer)
```

Parameters **board_index**: the index of the device access board

pad: the device primary GPIB address

sad: the device secondary GPIB address

tmo: the I/O timeout value

eot: enable or disable the device EOI mode

eos: configure the device EOS character and device EOS modes

Return Value The device descriptor or -1

Error Codes EARG, EDVR, ENEB

ibdma

Description This command enables or disables DMA. This function is not supported for the Model KPXI-488.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibdma (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibdma (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board descriptor

dma: enable or disable DMA mode

Return Value The value of the **ibsta**

Error Codes EARG, ECAP, EDVR, ENEB, EOIP

ibeot

Description This command enables or disables the action that is setting GPIB EOI line to enable while the I/O operation is completed. If the EOT mode is enabled, the EOI line is set to enable while the last GPIB is written to bytes. Otherwise, there is no operation to be performed while the last byte is sent.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ileot (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibeot (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board or device descriptor

v: enable or disable eot mode

Return Value The value of the **ibsta**

Error Codes EDVR, ENEB, EOIP

ibeos

Description This command configures the EOS termination mode or character.

NOTE *Defining an **EOS** byte does not automatically send it when I/O writing is terminated; the user must set the **EOS** byte after the data strings have been defined by the application.*

Support Level Board / device level

Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

```
ibeos (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibeos (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board or device descriptor

v: The information of the **EOS** mode and character. If **v** is zero, the **EOS** configuration is disabled. Otherwise, the low byte is the **EOS** character and the upper byte contains the flags that define the **EOS** mode. Table 2-9 shows the different **EOS** configurations and the corresponding values of **v**.

Configure **bit A** and **bit C** to determine how to terminate the I/O reading. If **bit A** is set and **bit C** is clear, the I/O reading is terminated when a byte that matches the low seven bits of the **EOS** character is received. If both **bit A** and **bit C** are set, the I/O reading is terminated when a byte matching the entire eight bits of the **EOS** character is received.

Configure **bit B** and **bit C** to determine how to control the **GPIB EOI** line during I/O writing. If **bit B** is set and **bit C** is clear, the **EOI line** is set to enable when a byte that matches the low seven bits of the **EOS** character is written. If both **bit B** and **bit C** are set, the **EOI line** is set to enable when a byte matching the entire eight bits of the **EOS** character is written.

Table 2-9
EOS mode V value

EOS mode	V Value		
	Bit	**Upper Byte**	**Low Byte**
Terminate reading when the EOS is detected.	A	00000100	EOS character
Through the write function, set EOI with EOS.	B	00001000	EOS character
Compare the entire eight bits of the EOS byte rather than the low 7 bits.	C	00010000	EOS character

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibfind

Description This command opens and initializes the GPIB board descriptor. The returned board descriptor can be used in later commands. Similar to **ibonl** 1, **ibfind** performs a board description initialization. Before the board is put offline by using **ibonl** 0, the descriptor that is returned by **ibfind** is valid; -1 is returned if **ibfind** is unable to get a valid descriptor. At the same time, the **ERR** bit of the **ibsta** and the **EDVR** bit of the **iberr** are set.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibfind (const char *boardname)
```

Visual Basic

```
ibfind (ByVal boardname As String) As Integer
```

- or -

```
call ibfind (ByVal boardname As String, ud As
  Integer)
```

Parameters **boardname**: board name; for example, gpib0

Return Value The board descriptor or -1

Error Codes EBUS, ECIC, EDVR, ENEB

ibgts

Description This command sets the board from active control status to standby control status. **ibgts** sets the GPIB board as the standby control unit and releases the control of the **GPIB ATN** line.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibgts (int ud, int shadow_handshake)
```

Visual Basic

```
ibgts (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibgts (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board descriptor

v: determines whether to handshake with receiver

Return Value The value of the **ibsta**

Error Codes EADR , EARG, ECIC, EDVR, ENEB, EOIP

ibist

Description This command sets or clears the board individual **status (ist)** bit for parallel polling.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibist (int ud, int ist)
```

Visual Basic

```
ibist (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibist (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board descriptor

v: shows whether to set or clear the **ist** bit

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

iblines

Description Returns the **GPIB control** lines status. The low-order lines byte (bits 0 to 7) shows that the GPIB interface has the capability to automatically detect the status of each GPIB control line. The upper byte (bits 8 to 15) shows the status of the GPIB control line. A description of each byte is listed in Table 2-10.

To determine whether a GPIB line is controlled, complete the following steps:

1. Check the appropriate bit of the low byte to ensure the line can be monitored.

2. Check whether the corresponding bit of the upper byte can be monitored (the appropriate bit of the low byte is 1).

If the checked bit of the upper byte is set (1), the corresponding line is in controlled status; if the checked bit of the upper byte is clear (0), the corresponding line is not in controlled status.

Table 2-10
iblines

7	6	5	4	3	2	1	0
EOI	ATN	SRQ	REN	INF	NRFD	NDAC	DAV

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
iblines (ByVal ud As Integer, lines As Integer)
  As Integer
```

- or -

```
call iblines (ByVal ud As Integer, lines As Integer)
```

Parameters **ud**: board descriptor

line_status: the status information of the returned GPIB control line

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibln

Description This command determines if there is an available device on bus.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibln (ByVal ud As Integer, ByVal pad As Integer,
  ByVal sad As Integer, found_listener As Integer)
  As Integer
```

- or -

```
call ibln (ByVal ud As Integer, ByVal pad As Integer,
  ByVal sad As Integer, found_listener As Integer)
```

Parameters **ud**: board or device descriptor. The board tests for listeners if **ud** is a board descriptor. **ibln** tests for listeners with the interface related with the device if **ud** is a device descriptor. If a listener is detected, a non-zero value is returned in the **found_listener**.

pad: device primary address (addressing value between 0 and 30)

sad: the device secondary address (addressing value is between 96 and 126, **NO_SAD** or **ALL_SAD**. **NO_SAD** is no secondary addressing, only a primary addressing for example. **ALL_SAD** is set to test all secondary addresses)

found_listener: shows if there is a device available

Return Value The value of the **ibsta**

Error Codes EARG, ECIC, EDVR, ENEB, EOIP

ibloc

Description If a board is not in lockout status, **ibloc** sets the board in local control mode. If **LOK** does not exist in the status word, **ibsta**, the board is in a lockout state. If a board is in lockout, calling **ibloc** has no effect.

If the computer is used as an apparatus, **ibloc** is used to simulate a panel **RTL** (Return to Local) switch.

All device-level commands automatically set the device to remote mode except the **Remote Enable (REN) line** is not controlled by **ibsre**; **ibloc** is used to temporarily set the device from **remote** mode to **local** mode before the next device-level command is executed.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibloc (ByVal ud As Integer) As Integer
```

- or -

```
call ibloc (ByVal ud As Integer)
```

Parameters **ud**: board or device descriptor

Return Value The value of the **ibsta**

Error Codes EBUS, ECIC, EDVR, ENEB, EOIP

ibonl

Description This command resets the board or device parameters to default settings and sets the device online or offline. If the device or interface is set to offline, the board or device descriptor is no longer effect. Once called, use **ibdev** or **ibfind** to access the board or device.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibonl (int ud, int onl)
```

Visual Basic

```
ibonl (ByVal ud As Integer, ByVal onl As Integer)
  As Integer
```

- or -

```
call ibonl (ByVal ud As Integer, ByVal onl As
  Integer)
```

Parameters **ud**: board or device descriptor

onl: online (1) or offline (0)

Return Value The value of the **ibsta**

Error Codes EARG, ENEB

ibnotify

Description This command uses the selected callback function to notify the user of one or more GPIB events. The re-synchronization handler is needed after the completion of the asynchronous I/O operation; the global variable is passed to the callback function while the operation of the I/O status is completed.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Parameters **ud**: board or device descriptor

mask: GPIB event code. Table 2-11 contains the valid event codes.

Table 2-11
GPIB event codes for mask

Event code	Description
- 0	No mask
- TIMO	The notify period is limited by the timeout period (see **ibtmo**)
- END	END or EOS is detected
- SRQI	SRQ signal is sent (only board level)
- RQS	Device requested service (only device level)
- CMP	I/O completion
- LOK	GPIB interface is in **Lockout Status** (only board level)
- REM	GPIB interface is in **Remote Status** (only board level)
- CIC	GPIB interface is **CIC** (only board level)
- ATN	Attention signal is sent (only board level)
- TACS	GPIB interface is a talker (only board level)
- LACS	GPIB interface is a listener (only board level)
- DTAS	GPIB interface is in **Device Trigger Status** (only board level)
- DCAS	GPIB interface is in **Device Clear Status** (only board level)

If GPIB mask is non-zero, the events specified by mask are monitored by **ibnotify**. while one or more of the events appears, the callback function is called. For board-level **ibnotify** call, all mask bits are valid except for ERR and RQS. For device-level ibnotify call, **CMPL**, **TIMO**, **END**, and **RQS** are the only valid mask bits. If **TIMO** is set in the notify mask, **ibnotify** calls the callback function even if no events have occurred while the limited time is gone. If **TIMO** is not set in the notify mask, the callback function is not called until one or more specified events occur.

Callback: the address callback function (Table 2-12 contains a description of the function's properties).

Table 2-12
Callback description (for ibnotify)

Property	Description
Prototype	int_std call Callback (int LocalUd, int Local**ibsta**, int LocalIberr, long LocalIbcntl, void *RefData)
Parameters	**LocalUd**: board or device descriptor **Localibsta**: the **ibsta** value **LocalIberr**: the **iberr** value **LocalIbcntl**: the **ibcntl** value **RefData**: the reference data for the callback function defined by user
Return value	the next mask of the notified GPIB event
Error code	EDVR

RefData: the reference data for the callback function defined by user

Return Value The value of the **ibsta**

Error Codes EARG, ECAP, EDVR, ENEB, EOIP

ibpad

Description This command sets a board or a device primary GPIB address.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibpad (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibpad (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board or device descriptor

v: the GPIB primary address (the valid range is 0 to 30)

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibsad

Description This command sets or disables a board or device secondary GPIB address.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibsad (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibsad (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board or device descriptor

v: Set or disable the GPIB secondary address. If *v* is zero, the secondary address is disabled. If *v* is non-zero, the secondary address is enabled with a secondary address valid range of 96 to 126 (0x60 to 0x7E).

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibpct

Description This command passes Controller-in-Charge (CIC) status to another GPIB device that has controller capability. The interface automatically releases the **ATN line** and goes to Controller Idle Status (CIDS).

Support Level Device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibpct (ByVal ud As Integer) As Integer
```

- or -

```
call ibpct (ByVal ud As Integer)
```

Parameters **ud**: device descriptor

Return Value The value of the **ibsta**

Error Codes EARG, EBUS, ECIC, EDVR, ENEB, EOIP

ibppc

Description This command configures parallel polling.

If **ud** is a device descriptor, **ibppc** enables or disables the device response to parallel polling. The addressed device sends the Parallel Poll Enable (PPE) or Parallel Poll Disable (PPD) message. Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero corresponding to sent PPD.

If **ud** is a board descriptor, **ibppc** uses the parallel poll configuration value **v** to perform a local parallel poll configuration. Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero corresponding to send PPD. If there is no error happening within the calling period, **iberr** maintains the previous value of the local parallel poll configuration.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibppc (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibppc (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: device descriptor

v: enable/disable parallel polling

Return Value The value of the **ibsta**

Error Codes EARG, EBUS, ECAP, ECIC, EDVR, ENEB, EOIP

ibrd

Description This command reads data from a device to the indicated buffer.

The GPIB is addressed by **ibrd**, which reads count data bytes (count is the counting value in the counter); when **ud** is the device descriptor, the count data bytes are placed in the user buffer. The operation ends when the count data bytes have been read or when **END** is read. If the count bytes reading does not finish

before the timeout period ends, the operation stops with an error. The actual number of transferred bytes is returned in the global variable, **ibcntl**.

When **ud** is the board descriptor**,** count data bytes are read by **ibrd** and placed in the user buffer. The GPIB has already been addressed by the board-level **ibrd**; the operation ends when the count data bytes or **END** are read. If the count bytes reading is not complete within the timeout period (or the board is not CIC, and CIC sends the **Device Clear** message on the GPIB bus), the operation stops with an error. The actual number of transferred bytes is returned in the global variable, **ibcntl**.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibrd (ByVal ud As Integer, buf As String,
  ByVal cnt As Long) As Integer
```

- or -

```
call ibrd (ByVal ud As Integer, buf As String)
```

Parameters **ud**: device descriptor

buf: the buffer that stores the data that is read from the GPIB

cnt: the number of the bytes read from the GPIB

Return Value The value of the **ibsta**

Error Codes EABO, EADR, EBUS, ECIC, EDVR, ENEB, EOIP

ibrda

Description This command asynchronously reads data from a device to the designated buffer.

The GPIB is addressed by **ibrda**, which reads count data bytes (count is the counting value in the counter); when **ud** is the device descriptor, the count data bytes are placed in the user buffer. The operation ends when the count data bytes have been read or when **END** is read. If the count bytes reading does not finish before the timeout period ends, the operation stops with an error. The actual number of transferred bytes is returned in the global variable, **ibcntl**.

Count data bytes are read by **ibrda** and placed in the user buffer when **ud** is the board descriptor. The GPIB has already been addressed by the board-level **ibrda**; the operation ends when the count data bytes or **END** are read. If the count bytes reading is not complete within the timeout period (or the board is not CIC, and CIC sends the **Device Clear** message on the GPIB bus), the operation stops with an error. The actual number of transferred bytes is returned in the global variable, **ibcntl**.

The design purpose of the asynchronous I/O commands (**ibcmda**, **ibrda**, **ibwrta**) is that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O has begun, later GPIB commands are

strictly limited; any command that would interfere with the I/O in progress will not be allowed. In this case, **EOIP** is returned by the driver.

When the I/O is complete, the application and the driver must be re-synchronized.

Use one of the following functions to re-synchronize:

ibwait: If the CMPL bit of the returned **ibsta** is set, the driver and application are re-synchronized.

ibnotify: If the **ibsta** value sent to the ibnotify callback contains **CMPL**, the driver and application are re-synchronized.

ibstop: The I/O is stopped, and the driver and application are re-synchronized.

ibonl: The I/O is stopped and the interface is reset; the driver and application are re-synchronized.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibrda (ByVal ud As Integer, buf As String, ByVal cnt
  As Long) As Integer
```

- or -

```
  call ibrda (ByVal ud As Integer, buf As String)
```

Parameters **ud**: device descriptor

buf: the buffer that stores the data that is read from the GPIB

cnt: the number of the bytes that is read from the GPIB

Return Value The value of the **ibsta**

Error Codes EABO, EADR, EBUS, ECIC, EDVR, ENEB, EOIP

ibrdf

Description This command reads data from a device and saves it to a file.

The GPIB is addressed by **ibrdf**, which reads the data bytes from the GPIB device, then saves them to a file (when **ud** is a device descriptor). The operation stops when **END** is read. If the data transfer does not finish before the timeout period ends, the operation stops with an error. The actual number of transferred bytes is returned in the global variable, **ibcntl**.

Data bytes are read from the GPIB device by **ibrdf**, then saved to a file when **ud** is the board descriptor. The GPIB has already been addressed by the board-level **ibrdf**; the operation stops when **END** is read. If the data transfer is not complete within the timeout period (or the board is not CIC, and CIC sends the **Device Clear** message on the GPIB bus), the operation stops with an error. The actual number of transferred bytes is returned in the global variable, **ibcntl**.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibrdf (int ud, const char *filename)
```

Visual Basic

```
ibrdf (ByVal ud As Integer, ByVal filename As
  String ) As Integer
```

- or -

```
call ibrdf (ByVal ud As Integer, ByVal filename
  As String)
```

Parameters **ud**: device descriptor

filename: the file name; the file stores the read data.

Return Value The value of the **ibsta**

Error Codes EABO, EADR, EBUS, ECIC, EDVR, EFSO, ENEB, EOIP

ibrpp

Description This command performs parallel polling.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibrpp (ByVal ud As Integer, ppr As Integer)
  As Integer
```

- or -

```
call ibrpp (ByVal ud As Integer, ppr As Integer)
```

Parameters **ud**: device descriptor

ppr: the parallel polling result

Return Value The value of the **ibsta**

Error Codes EBUS, ECIC, EDVR, ENEB, EOIP

ibrsc

Description This command sends the **Interface Clear (IFC)** message or **Remote Enable (REN)** message to request or release the system control. The operations that request system controller capability are not allowed if the board releases system control; when the board requests system control, operations that request system controller capability are allowed.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibrsc (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibrsc (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: device descriptor

v: 0: release system control; 1: request system control

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibrsp

Description This command performs sequential polling. The device is requesting service if bit 6 of the response is set. If automatic sequential polling is enabled, the device has already been polled and the previous status byte value is returned by **ibrsp**.

Support Level Device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibrsp (ByVal ud As Integer, spr As Integer)
  As Integer
```

- or -

```
call ibrsp (ByVal ud As Integer, spr As Integer)
```

Parameters **ud**: device descriptor

spr: the sequential polling result

Return Value The value of the **ibsta**

Error Codes EABO, EARG, EBUS, ECIC, EDVR, ENEB, EOIP, ESTB

ibrsv

Description This command requests service and changes the status byte of the sequential polling.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibrsv (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibrsv (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: device descriptor

v: the status byte of the sequential polling

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibsic

Description This command enables the **GPIB interface clear (IFC)** line to allow at least 100ns when the GPIB interface is the system controller by initializing the GPIB interface, designating it as CIC, and activating the controller by setting **ATN line**.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibsic (ByVal ud As Integer) As Integer
```

- or -

```
call ibsic (ByVal ud As Integer)
```

Parameters **ud**: device descriptor

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP, ESAC

ibsre

Description This command sets or clears the **Remote Enable (REN) line**. The **Remote Enable (REN) line** is used by devices to choose local or remote modes of operation; **ibsre** sets or clears the **REN line**. The **GPIB REN line** is enabled when the remote enable line is set, and disabled when the remote enable line is cleared. A device cannot enter remote mode before it receives its listen address and the **REN** is initiated.

Support Level Board level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibsre (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibsre (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board descriptor

v: Sets or clears **REN** line. 0: clear; 1: set

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP, ESAC

ibstop

Description This command stops asynchronous I/O operation. If the **ibsta** command is used when asynchronous I/O is operating, the error code EABO is returned to show the I/O was successfully stopped.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibstop (ByVal ud As Integer) As Integer
```

- or -

call ibstop (ByVal ud As Integer)

Parameters **ud**: board or device descriptor

Return Value The value of the **ibsta**

Error Codes EABO, EBUS, EDVR, ENEB

ibtmo

Description This command sets the board or device timeout period. The timeout period is the maximum continuous time allowed for synchronous I/O operation (**ibrd** and **ibwrt** for example); or the maximum waiting time of **ibwait** or **ibnotify** that uses **TIMO** in the mask. If the operation is not completed within the timeout period, the operation is stopped and returns **TIMO** in **ibsta**.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibtmo (ByVal ud As Integer, ByVal v As Integer)
  As Integer
```

- or -

```
call ibtmo (ByVal ud As Integer, ByVal v As Integer)
```

Parameters **ud**: board or device descriptor

v: timeout period value. The valid timeout values are shown in Table 2-13:

Table 2-13
ibtmo timeout

Constant	V Value	Minimum Timeout
TNONE	0	Disabled - no timeout period
T10µs	1	10µs
T30µs	2	30µs
T100µs	3	100µs
T300µs	4	300µs
T1ms	5	1ms
T3ms	6	3ms
T10ms	7	10ms

Table 2-13 (continued)
ibtmo timeout

Constant	V Value	Minimum Timeout
T30ms	8	30ms
T100ms	9	100ms
T300ms	10	300ms
T1s	11	1s
T3s	12	3s
T10s	13	10s
T30s	14	30s
T100s	15	100s
T300s	16	300s
T1000s	17	1000s

Return Value The value of the **ibsta**

Error Codes EARG, EDVR, ENEB, EOIP

ibtrg

Description This command sends the **Group Execute Trigger (GET)** message to a device.

Support Level Device level

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
ibtrg (ByVal ud As Integer) As Integer
```

- or -

```
call ibtrg (ByVal ud As Integer)
```

Parameters **ud**: device descriptor

Return Value The value of the **ibsta**

Error Codes EARG, EBUS, ECIC, EDVR, ENEB, EOIP

ibwait

Description **ibwait** waits for one or more events described by mask (including **TIMO**) to occur. If **TIMO** in the wait mask is set, **ibwait** returns when the timeout period has expired even if no other GPIB events occur. Setting **TIMO** to zero returns the newest **ibsta** immediately. If the **TIMO** in the wait mask is cleared, the function waits indefinitely for a GPIB event (described by mask).

The present **ibwait** mask bits are the same as **ibsta** bits. Only the **TIMO**, **END**, **RQS**, and **CMPL** are valid wait mask bits if **ud** is a device descriptor. Except for **RQS**, if **ud** is a board descriptor, all wait mask bits are valid.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

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

Syntax **Visual Basic**

```
ibwait (ByVal ud As Integer, ByVal mask As Integer)
  As Integer
```

- or -

```
call ibwait (ByVal ud As Integer, ByVal mask As
  Integer)
```

Parameters **ud**: board or device descriptor

mask: GPIB events that can be monitored. The valid code values are shown in Table 2-14:

Table 2-14
ibwait valid mask codes

Mask	Bit Position	Hex Value	Description
ERR	15	8000	GPIB error
TIMO	14	4000	Mask timeout
END	13	2000	**END** or **EOS** is detected by GPIB board
SRQI	12	1000	Send SRQ signal (only board)
RQS (only device level)	11	800	Device requesting service
SPOLL	10	400	Controller sequentially polls the board
EVENT	9	200	A **DTAS**, **DCAS**, or **IFC** event occur
CMPL	8	100	I/O completed
LOC	7	80	GPIB board is in Lockout Status
REM	6	40	GPIB board is in Remote Status
CIC	5	20	GPIB board is in CIC status
ATN	4	10	Send Attention signal
TACS	3	8	GPIB board as a talker
LACS	2	4	GPIB board as a listener
DTAS	1	2	GPIB board is in Device Trigger Status
DCAS	0	1	GPIB board is in Device Clear Status

Return Value The value of the **ibsta**

Error Codes EARG, EBUS, ECIC, EDVR, ENEB, ESRQ

ibwrt

Description This command writes data from a buffer to a device.

When **ud** is a device descriptor, **ibwrt** addresses the GPIB and writes count data bytes (**cnt** is the tallying value in the counter) from the board's memory to the GPIB device. The operation normally ends when **cnt** number of data bytes have been written; if **cnt** number of bytes are not written completely during the timeout period, the operation stops with an error. The number of bytes actually transferred is returned in the global variable, **ibcntl**.

When **ud** is a board descriptor, the board-level **ibwrt** automatically writes **cnt** data bytes from the buffer to the GPIB device. Normally, this operation ends when the **cnt** number of data bytes are completely written; if **cnt** number of bytes are not completely written during the timeout period (or, if the board is not CIC and CIC sends the **Device Clear** message on the GPIB bus), the operation stops with an error. The number of bytes actually transferred is returned in the global variable **ibcntl**.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibwrt (int ud, const void *buf, long cnt)
```

Visual Basic

```
ibwrt (ByVal ud As Integer, ByVal buf As String,
  ByVal cnt As Long) As Integer
```

- or -

```
call ibwrt (ByVal ud As Integer, ByVal buf As
  String)
```

Parameters **ud**: device unit descriptor

buf: the buffer that contains the sent data bytes

cnt: the number of sent data bytes

Return Value The value of the **ibsta**

Error Codes EADR, EABO, EBUS, ECIC, EDVR, EOIP, ENEB, ENOL

ibwrta

Description This command asynchronously writes data from a buffer to a device .

When ud is a device descriptor, **ibwrta** addresses the GPIB and writes count data bytes (**cnt** is the tallying value in the counter) from the board's memory to the GPIB device. The operation normally ends when the count data bytes have been written; if the count bytes are not written completely during the timeout period, the operation stops with an error. The number of bytes actually transferred is returned in the global variable **ibcntl**.

When ud is a board descriptor, the board-level **ibwrt** automatically writes cnt data bytes from the buffer to the GPIB device. Normally, this operation ends when the count data bytes are completely written; if cnt bytes are not written during the timeout period (or, if the board is not CIC and CIC sends the **Device Clear message** on the GPIB bus), the operation stops with an error. The number of bytes actually transferred is returned in the global variable **ibcntl**.

The design purpose of the asynchronous I/O commands (**ibcmda**, **ibrda**, **ibwrta**) is that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O has begun, later GPIB commands are strictly limited; any command that would interfere with the I/O in progress will not be allowed. In this case the **EOIP** is returned by the driver.

When the I/O is complete, the application and the driver must be re-synchronized.

Use one of the following functions to re-synchronize:

ibwait: If the CMPL bit of the returned **ibsta** is set, the driver and application are re-synchronized.

ibnotify: If the **ibsta** value sent to the ibnotify callback contains **CMPL**, the driver and application are re-synchronized.

ibstop: The I/O is stopped, and the driver and application are re-synchronized.

ibonl: The I/O is stopped and the interface is reset; the driver and application are re-synchronized.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibwrta (int ud, const void *buf, long cnt)
```

Visual Basic

```
ibwrta (ByVal ud As Integer, ByVal buf As String,
  ByVal cnt As Long) As Integer
```

- or -

```
call ibwrta (ByVal ud As Integer, ByVal buf As
  String)
```

Parameters **ud**: device unit descriptor

buf: the buffer that contains the sent data bytes

cnt: the number of sent data bytes

Return Value The value of the **ibsta**

Error Codes EADR, EABO, EBUS, ECIC, EDVR, EOIP, ENEB, ENOL

ibwrtf

Description This command writes data from a file to a device.

When **ud** is a device descriptor, **ibwrtf** addresses the GPIB and writes all data bytes in **filename** to the GPIB device. The operation normally ends when all the data bytes have been written; if all the bytes are not written during the timeout period, the operation stops with an error. The number of bytes actually transferred is returned in the global variable **ibcntl**.

When **ud** is a board descriptor, the board-level **ibwrtf** automatically writes all data bytes in **filename** to the GPIB device. Normally, this operation ends when all the data bytes are completely written; if all data bytes are not written during the timeout period (or, if the board is not CIC and CIC sends the **Device Clear** message on the GPIB bus), the operation stops with an error. The number of bytes actually transferred is returned in the global variable, **ibcntl**.

Support Level Board / device level

Syntax **Microsoft C/C++ and Borland C++**

```
int ibwrtf (int ud, const char *filename)
```

Visual Basic

```
ibwrtf (ByVal ud As Integer, ByVal filename As
  String) As Integer
```

- or -

```
call ibwrtf (ByVal ud As Integer, ByVal filename
  As String)
```

Parameters **ud**: device descriptor

filename: the file name; the file contains the data written

Return Value The value of the **ibsta**

Error Codes EABO, EADR, EBUS, ECIC, EDVR

Multi-device functions

This section provides a NI command compatible multi-device IEEE 488 function reference. Refer to Section 1 for information on Keithley Command Compatible Functions.

AllSpoll

Description This command sequentially polls one or more devices. The responses and number of responses of poll are individually stored in **resultList** and **ibcntl**.

Syntax **Microsoft C/C++ and Borland C++**

```
void AllSpoll (int board_desc,
  const Addr4882_t addressList[], short resultList[])
```

Visual Basic

```
call AllSpoll (ByVal board_desc As Integer,
  addressList ( ) As Integer,
  resultList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**

resultList: the list of sequential poll responses of the devices; the devices correspond to the device addresses in addrlist

Error Codes EARG, EABO, EBUS, ECIC, EDVR, EOIP, ENEB

DevClear

Description This command sends the **Selected Device Clear (SDC) GPIB message** to clear the selected device. If the address is constant **NOADDR** (the end point of the list), the universal **Device Clear (DCL)** message is sent to all devices.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call DevClear (ByVal board_desc As Integer,
  ByVal address As Integer)
```

Parameters **board_desc**: board ID

address: the device address; the device that needs to be cleared

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB

DevClearList

Description This command clears multiple devices. If the address is the constant **NOADDR**, the **DCL message** is sent to all devices.

Syntax **Microsoft C/C++ and Borland C++**

```
void DevClearList (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call DevClearList (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**; the devices that need to be cleared

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB

EnableLocal

Description This command sends a **Go To Local (GTL)** GPIB message to multiple devices. This sets multiple devices in local mode, allowing local operation. If only the constant in **addrlist** is **NOADDR**, the **Remote Enable (REN) GPIB line** is set to disable.

Syntax **Microsoft C/C++ and Borland C++**

```
void EnableLocal (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call EnableLocal (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **Board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**; the devices are waiting to return to local mode

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ESAC

EnableRemote

Description This command sets the **Remote Enable (REN) line** to enable, which places **addressList** devices into a listen-active state. This allows devices to be programmed remotely (remote GPIB programmable).

Syntax **Microsoft C/C++ and Borland C++**

```
void EnableRemote (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call EnableRemote (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**; the devices are waiting to go to remote control mode

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ESAC

FindLstn

Description This command finds listening devices on the GPIB bus. This function tests all primary addresses in **padlist** as follows: if a device exists in a given **padlist**, the device primary address is stored in **resultlist**. Otherwise, the function tests all the secondary addresses of the primary ones, and stores the addresses of any finding devices. **ibcntl** includes the actual numbers of addresses stored in **resultlist**.

Syntax **Microsoft C/C++ and Borland C++**

```
void FindLstn (int board_desc,
  const Addr4882_t padList[], Addr4882_t resultList[],
  int maxNumResults)
```

Visual Basic

```
call FindLstn (ByVal ud As Integer, padList ( )
  As Integer, resultList ( ) As Integer,
  ByVal maxNumResults As Integer)
```

Parameters **board_desc**: board ID

padList: the list of the GPIB primary addresses ended by **NOADDR**

resultList: the list of all listening device addresses; the listening devices found by **FindLstn** function

maxNumResults: the maximum number of the resultList

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ETAB

FindRQS

Description This command sequentially polls devices to determine which device is requesting service; the resulting byte is returned in **ibcntl. ibcntl** contains the index of the device requesting service in **addrList**. If no device is requesting service, **ETAB** and the index of **NOADDR** are individually returned in **iberr** and **ibcntl**.

Syntax **Microsoft C/C++ and Borland C++**

```
void FindRQS (int board_desc, const Addr4882_t
addressList[], short *result)
```

Visual Basic

```
call FindRQS (ByVal ud As Integer, addressList ( )
  As Integer, result As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the GPIB primary addresses ended by NOADDR

result: the sequentially poll return byte of the device requesting service

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ETAB

PassControl

Description This command sends the **Take Control (TCT) GPIB message** to the device for passing control to another GPIB device with control capability. The device

changes to Controller-In-Charge (CIC) status when the interface is no longer CIC status.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call PassControl (ByVal board_desc As Integer,
  ByVal address As Integer)
```

Parameters **board_desc**: board ID

address: the list of the GPIB primary addresses ended by **NOADDR**

Error Codes EAGR, EBUS, ECIC, EDVR, EOIP, ENEB

PPoll

Description This command performs parallel polling one time. The board sends a command to all devices (see **PPollConfig** and **PPollUnconfig**). The controller can simultaneously obtain one-bit status messages relayed from up to eight devices when parallel polling is performed.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call PPoll (ByVal board_desc As Integer,
  result As Integer)
```

Parameters **board_desc**: board ID

result: the result of the parallel polling

Error Codes EBUS, ECIC, EDVR, EOIP, ENEB

PPollConfig

Description This command controls or releases the GPIB data line to configure the device to respond to parallel polling. If **lineSense** is equal to the ist bit of the device, the assigned GPIB data line is controlled in a parallel polling duration. Otherwise, the assigned data line is not controlled in a parallel polling duration. The controller can simultaneously obtain one-bit status messages related with it from up to eight devices by a parallel polling.

Syntax **Microsoft C/C++ and Borland C++**

```
void PPollConfig (int board_desc, Addr4882_t address,
  int dataLine, int lineSense)
```

Visual Basic

```
call PPollConfig (ByVal ud As Integer,
  ByVal address As Integer, ByVal dataLine As Integer,
  ByVal lineSense As Integer)
```

Parameters **board_desc**: board ID

address: the device address of the device is waiting to be configured.

dataLine: data line on which the device responds to parallel polling; its range is from 1 to 8.

lineSense: senses the parallel polling response; its value is either 0 or 1.

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB

PPollUnConfig

Description This command un-configures the devices to respond to parallel polling. If there is only constant **NOADDR** in the address list, addrlist, the **Parallel Poll Un-configure (PPU) GPIB** message is sent to all GPIB devices. The devices un-configured by this function will not be included in the following parallel polling.

Syntax **Microsoft C/C++ and Borland C++**

```
void PPollUnconfig (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call PPollUnconfig (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**

Error Codes EAGR, EBUS, ECIC, EDVR, EOIP, ENEB

RcvRespMsg

Description This command reads data from a device. The **RcvRespMsg** function assumes that the interface is in the listen-active status and addresses a device as a talker. The function reads data continuously, until either "count" data have been read or the terminal condition is detected. If the terminal condition is **DTOPend**, the reading action is stopped and the **EOI** line is set to enable while the **STOPend** is received. Otherwise, the reading action is stopped while the eight-bit **EOS** character is detected. Returns the actual number of transferred bytes in the global variable, **ibcntl**.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call RcvRespMsg (ByVal ud As Integer, buf As String,
  ByVal termination As Integer)
```

Parameters **board_desc**: board ID

buffer: the buffer for storing the read data

count: the number of read bytes

termination: the description of the data termination mode

Error Codes EABO, EADR, EARG, ECIC, EDVR, EOIP, ENEB

ReadStatusByte

Description This command sequentially polls a device. If the sixth bit (hex 40) of the response is set, the device is requesting service.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call ReadStatusByte (ByVal us As Integer,
  ByVal addr As Integer, result As Integer)
```

Parameters **board_desc**: board ID

address: device address

result: response byte of the sequential polling

Error Codes EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB

Receive

Description This command reads data bytes from a device, and then stores them in the assigned buffer. Receive the device address described by addressing to a talker, setting the interface to a receiver, reading count data bytes from the device, and storing these data bytes into the buffer. The operation is normally stopped when the count data bytes have been read or the terminal condition is detected. If the terminal condition is **STOPend**, the **EOI** line is set to enable while the **STOPend** byte is received. Otherwise, the reading operation is stopped while the eight-bit **EOS** character is detected. Returns the actual number of transferred bytes in the global variable, **ibcntl**.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call Receive (ByVal ud As Integer,
  ByVal addr As Integer, buf As String,
  ByVal termination As Integer)
```

Parameters **board_desc**: board id

address: the device address; the device is read by the function for data

buffer: the buffer that stores the read data

termination: device termination mode (STOPend or EOS character)

Error Codes EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB

ReceiveSetup

Description This command configures the device to be a talker and the interface to a receiver. After the function **ReceiveSetup**, **RcvRespMsg** function is usually called to transfer the data from the device to the interface. **ReceiveSetup** is helpful for multiple **RcvRespMsg** calls. When **ReceiveSetup** is adopted, the re-addressing is not necessary when each data block is received.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call ReceiveSetup (ByVal ud As Integer,
  ByVal addr As Integer)
```

Parameters **board_desc**: board ID

address: the device address; the device you want the talker to address

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB

ResetSys

Description This command resets and initializes devices. The function contains three steps. First, reset the GPIB by controlling the **Remote Enable (REN) line** and then controlling the **Interface Clear (IFC) line**. Second, send the **Universal Device Clear (DCL)** GPIB message to clear all devices. Finally, send the **"*RST\n"** message to the address list, **addrlist**, to complete resetting and initialization of the device.

Syntax **Microsoft C/C++ and Borland C++**

```
void ResetSys (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call ResetSys (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**

Error Codes EABO, EARG, EBUS, ECIC, EDVR, ENOL, EOIP, ENEB, ESAC

Send

Description This command writes data bytes from the buffer to the device. The operation is normally stopped until the count data bytes have been written. If **eotmode** is **DABend**, the **EOI** line is set to enable while the final byte is sent. If **eotmode** is **NULLend**, the **EOI** line is set to disable while the final byte is sent. If **eotmode** is **NLend**, the **EOI** line is controlled while the final byte and the following new character "\n" have been sent. Returns the actual number of transferred bytes in the global variable, **ibcntl**.

Syntax

Microsoft C/C++ and Borland C++

```
void Send (int board_desc, Addr4882_t address,
  const void *buffer, long count, int eot_mode)
```

Visual Basic

```
call Send (ByVal ud As Integer,
  ByVal addr As Integer, ByVal buf As String,
  ByVal eot_mode As Integer)
```

Parameters

board_desc: board ID

address: the device address

buffer: the sent data bytes

count: data count

eot_mode: data termination mode (**DABend**, **NULLend**, or **NLend**)

Error Codes EABO, EARG, EBUS, ECIC, EDVR, ENOL, EOIP, ENEB

SendCmds

Description This command sends GPIB commands. Returns the number of transferred command bytes in the global variable, **ibcntl**.

Syntax

Microsoft C/C++ and Borland C++

```
void SendCmds (int board_desc, const void *cmdbuf,
  long count)
```

Visual Basic

```
call SendCmds (ByVal ud As Integer,
  ByVal cmdbuf As String)
```

Parameters

board_desc: board ID

cmdbuf: the sent command bytes

count: data count

Error Codes EABO, ECIC, EDVR, ENOL, EOIP, ENEB

SendDataBytes

Description This command sends data from the buffer to the device. The **SendDataBytes** function assumes that the interface on the GPIB bus is in the talk-active status and already addresses the devices as listeners. If **eotmode** is **DABend**, the **EOI**

line is controlled while the final byte is sent. If **eotmode** is **NULLend**, the **EOI line** is not controlled while the final byte is sent. If **eotmode** is **NLend**, the **EOI line** is set to enable while the final byte and the following new character "\n" have been sent. Returns the actual number of transferred bytes in the global variable, **ibcntl**.

Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

```
call SendDataBytes (ByVal ud As Integer,
  ByVal buf As String, ByVal term As Integer)
```

Parameters

board_desc: board ID

buffer: the sent data bytes

count: data count

eot_mode: data terminal mode (**DABend**, **NULLend**, **NLend**)

Error Codes EABO, EADR, EARG, EBUS, ECIC, EDVR, ENOL, EOIP, ENEB

SendList

Description This command sends data bytes to multiple GPIB devices. The **SendList** function addresses all devices listed in address list, **addrlist**, as listeners, addresses the interface to talk, and then transfers the data from the buffer to the devices. If **eotmode** is **DABend**, the **EOI line** is set to enable while the final byte is sent. If **eotmode** is **NULLend**, the **EOI line** is set to disable while the final byte is sent. If **eotmode** is **NLend**, the **EOI line** is set to enable while the final byte and the following new character "\n" have been sent. Returns the actual number of transferred bytes in the global variable, **ibcntl**.

Syntax

Microsoft C/C++ and Borland C++

```
void SendList (int board_desc,
  const Addr4882_t addressList[], const void  *buffer,
  long count, int eotmode)
```

Visual Basic

```
call SendList (ByVal ud As Integer,
  addressList ( ) As Integer, ByVal buf As String,
  ByVal term As Integer)
```

Parameters

board_desc: board ID

addressList: the list of the device addresses; the devices that send data bytes

buffer: the sent data bytes

count: data count

eotmode: data termination mode (**DABend**, **NULLend**, or **NLend**)

Error Codes EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB

SendIFC

Description This command sends the **Interface Clear command** to reset GPIB. **SendIFC** is used to be a part of GPIB initialization. The function forces the interface to **Controller-in-Charge** of GPIB. At the same time, the function ensures that the devices connected with it are not addressed and that the interface calls of the devices are in idle status.

Syntax **Microsoft C/C++ and Borland C++**

```
void SendIFC (int board_desc)
```

Visual Basic

```
call SendIFC (ByVal ud As Integer)
```

Parameters **board_desc**: board ID

Error Codes ENEB, ESAC, EDVR, EOIP

SendLLO

Description This command sends the **Local Lockout (LLO) message** to all devices. While the LLO is to be effective, only the **Controller-In-Charge** can change device states by sending appropriate GPIB messages. **SendLLO** is reserved for use in uncommon local/remote situations. Under typical situations, **SetRWLS** is used to place a device in remote with lockout.

Syntax **Microsoft C/C++ and Borland C++**

```
void SendLLO (int board_desc)
```

Visual Basic

```
call SendLLO (ByVal ud As Integer)
```

Parameters **board_desc**: board ID

Error Codes EBUS, ECIC, ENEB, ESAC, EDVR, EOIP

SendSetup

Description This command configures the device to receive data. SendSetup sets the devices listed in addressList as listeners and sets the interface **talk-active**. After the **SendSetup** call, **SendDataBytes** sends data from the interface to the devices. While multiple **SendDataBytes** calls are used for transferring data, the address setting capability of **SendSetup** is especially useful, since each device does not tneed to be addressed while each data block is transferred.

Syntax **Microsoft C/C++ and Borland C++**

```
void SendSetup (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call SendSetup (ByVal ud As Integer,
  addrs ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the devices ended by **NOADDR**

Error Codes EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB

SetRWLS

Description This command configures the device to lockout status of remote control mode. **SetRWLS** sets the devices listed in addrlist to remote control mode by controlling the **Remote Enable (REN) GPIB line**. Then, the **LLO GPIB** message sets the devices to lockout status. Before the **Controller-In-Charge** calls **EnableLocal** to release **Local Lockout**, the user can not locally operate these devices.

Syntax **Microsoft C/C++ and Borland C++**

```
void SetRWLS ((int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call SetRWLS (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**

Error Codes EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ESAC

TestSRQ

Description This command detects the current status of the **GPIB Service Request (SRQ) line**. If the SRQ is controlled, the result contains a non-zero value. Otherwise, the result contains a zero value. **TestSRQ** is used to get the current status of GPIB SRQ line. **WaitSRQ** is used to wait until the device controls the GPIB SRQ line.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call TestSRQ (ByVal ud As Integer, result As Integer)
```

Parameters **board_desc**: board ID

result: the status of the SRQ line

Error Codes EDVR, EOIP, ENEB

TestSys

Description This command causes devices to process self tests. **TestSys** sends the "TST?" message to the devices. The "TST?" message makes the devices test themselves individually. Then it reads sixteen-bit self-test results from the devices. The self test result 0\n shows that the device passed its self test (if the self test result is not 0\n, it means that the device did not pass its self test). Refer to the documents that came with the device to determine cause of the failed self test. If **TestSys** does not return **Error** (i.e., the **ERR** bit is not set in **ibsta**), the failure number of the self tests is contained in **ibcntl**. Otherwise, the meaning of the **ibcntl** depends on the returned failure. If the device does not send a response in a limited time, then the test result, ?, is reported, and the error **EABO** is returned.

Syntax **Microsoft C/C++ and Borland C++**

```
void TestSys (int board_desc, Addr4882_t *addrlist,
  short resultList[])
```

Visual Basic

```
call TestSys (ByVal ud As Integer,
  addrlist ( ) As Integer, resultList ( ) As Integer)
```

Parameters **board_desc**: board ID

addrlist: the list of the device addresses ended by **NOADDR**

resultList: the list of the self test results; each test item corresponds to each address listed in **addrlist**

Error Codes EABO, EARG, EBUS, EDVR, ECIC, EOIP, ENEB, ENOL

Trigger

Description This command sends **the Group Execute Trigger (GET) GPIB message** to a device. If the address is constant **NOADDR**, the **GET** messages are sent to the devices that are currently listen-active on the GPIB bus.

Syntax **Microsoft C/C++ and Borland C++**

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

Visual Basic

```
call Trigger (ByVal ud As Integer,
  ByVal address As Integer)
```

Parameters **board_desc**: board ID

address: the device address; the device to be triggered

Error Codes EARG, EBUS, EDVR, ECIC, EOIP, ENEB

TriggerList

Description This command sends the **Group Execute Trigger (GET) GPIB message** to multiple devices. If there is only constant **NOADDR** in the **addrlist**, no device is addressed and the **GET** message is sent to the devices that are currently listen-active on the GPIB bus.

Syntax **Microsoft C/C++ and Borland C++**

```
void TriggerList (int board_desc,
  const Addr4882_t addressList[])
```

Visual Basic

```
call TriggerList (ByVal ud As Integer,
  addressList ( ) As Integer)
```

Parameters **board_desc**: board ID

addressList: the list of the device addresses ended by **NOADDR**

Error Codes EARG, EBUS, EDVR, ECIC, EOIP, ENEB

WaitSRQ

Description This command waits until the device controls the GPIB SRQ line. When **WaitSRQ** returns, the result contains a non-zero value if the SRQ line in controlled. Otherwise the result contains a zero value. Get the current status of the GPIB SRQ line by using **TestSRQ**. Use **WaitSRQ** to wait before the SRQ line can be controlled.

Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

```
call WaitSRQ (ByVal ud As Integer,
  result As Integer)
```

Parameters **board_desc**: board ID

result: the status of the SRQ line

Error Codes EDVR, EOIP, ENEB

Appendix A
Status/Error Codes

In this section:

Topic	Page
NI command compatible status codes	A-2
NI command compatible function error codes.....	A-3

NI command compatible status codes

This section contains information about possible error codes produced when using the National Instruments™ (NI)[1] command compatible functions. All commands update global status word **ibsta** which contains the GPIB status and the message from the user's GPIB hardware. After every command, the user can use the **ERR** bit of the **ibsta** to detect errors. The **ibsta** is a sixteen-bit word. A bit value equal to one means the condition occurred while a bit value equal to zero means the condition did not occur.

Table A-1
NI command compatible status codes

Mnemonic	Position	Hex	Type	Description
ERR	15	8000	device, board	GPIB error
TIMO	14	4000	device, board	Timeout
END	13	2000	device, board	END or EOS has been detected
SRQI	12	1000	board	SRQ interrupt occurred
RQS	11	800	device	Device requesting service
SPOLL	10	400	board	Board has been sequentially polled by controller
EVENT	9	200	board	DCAS, DTAS, or IFC event occurred
CMPL	8	100	device, board	I/O completion
LOK	7	80	board	Lockout status
REM	6	40	board	Remote status
CIC	5	20	board	Control-In-Charge
ATN	4	10	board	Send attention message
TACS	3	8	board	Talk status
LACS	2	4	board	Listen status
DATS	1	2	board	Device trigger status
DCAS	0	1	board	Device clear status

1. National Instruments™ and NI are trademarks of the National Instruments Corporation.

NI command compatible function error codes

NI command compatible function error codes are listed in the following table. Note that, the error variable is meaningful only when the **ERR** bit of the status variable, **ibsta**, is placed. Click the error mnemonic, and you can obtain a detailed description and the solution for each error.

Table A-2
NI command compatible function error codes

Error Mnemonic	iberr Value	Meaning Description
EDVR	0	OS error
ECIC	1	Function requests GPIB board as CIC
ENOL	2	No listen device on the GPIB bus
EADR	3	GPIB board addressing error
EARG	4	Invalid argument
ESAC	5	GPIB board is not on the system controller requesting status
EABO	6	I/O operation is aborted (timeout)
ENEB	7	GPIB board does not exit
EDMA	8	DMA error
EOIP	10	Asynchronous I/O in progress
ECAP	11	The operation is not performed
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	The status byte queue of the sequential polling overflow
ESRQ	16	SRQ is stuck in ON state
ETAB	20	Table problem

This page left blank intentionally.

KEITHLEY

Service Form

Model No. _____ **Serial No.** _____ **Date** _____

Name and Telephone No. _____

Company _____

List all control settings, describe problem and check boxes that apply to problem. _____

_____

_____

❒ Intermittent ❒ Analog output follows display ❒ Particular range or function bad; specify

_____

❒ IEEE failure ❒ Obvious problem on power-up ❒ Batteries and fuses are OK

❒ Front panel operational ❒ All ranges or functions are bad ❒ Checked all cables

Display or output (check one)

❒ Drifts ❒ Unable to zero ❒ Unstable

❒ Overload ❒ Will not read applied input

❒ Calibration only ❒ Certificate of calibration required ❒ Data required

(attach any additional sheets as necessary)

Show a block diagram of your measurement including all instruments connected (whether power is turned on or not). Also, describe signal source.

Where is the measurement being performed? (factory, controlled laboratory, out-of-doors, etc.) _____

_____

What power line voltage is used? _____ Ambient temperature? _____ °F

Relative humidity? _____ Other? _____

Any additional information. (If special modifications have been made by the user, please describe.)

_____

_____

Be sure to include your name and telephone number on this service form.

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.
All other trademarks and trade names are the property of their respective companies.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY • www.keithley.com

12/06