NI-488M™
Software Reference Manual

National Instruments IEEE 488
Multitasking UNIX Device Driver

July 1994 Edition

Part Number 320062-01

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

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20, Canada (Ontario) (519) 622-9310,
Canada (Québec) (514) 694-8521, Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921, Mexico 95 800 010 0793,
Netherlands 03480-33466, Norway 32-84 84 00, Singapore 2265886, Spain (91) 640 0085, Sweden 08-730 49 70,
Switzerland 056/20 51 51, Taiwan 02 377 1200, U.K. 0635 523545

Limited Warranty

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

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

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

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

Copyright

Trademarks

NI-488M™ is a trademark of National Instruments Corporation.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

Preface

About This Manual

This manual describes the NI-488M GPIB software. All NI-488M function calls for the C language are described in detail.

Organization of This Manual

This manual is divided into the following chapters and appendixes:

Chapter 1, *Introduction*, introduces you to the product and the manual.

Chapter 2, *The C Language Library*, contains a general discussion of the C language programming interface to the NI-488M UNIX device driver.

Chapter 3, *Using ibic*, introduces you to `ibic`, the interactive control program that enables you to communicate with GPIB devices through functions you enter at your keyboard. `ibic` is designed to help you learn how to use the NI-488M functions to program your devices.

Chapter 4, *Using ibconf*, introduces you to `ibconf`, the screen-oriented interactive utility that is used to edit the device and board data structures in a UNIX kernel.

Appendix A, *NI-488M Functions and Utilities Reference*, contains detailed information for using the NI-488M UNIX driver functions and utilities.

Appendix B, *Multiline Interface Command Messages*, is a listing of Multiline Interface Messages.

Appendix C, *GPIB Programming Example*, illustrates the steps involved in programming a digital voltmeter in the C Language. This section is designed to help you learn how to use the driver software to execute certain programming and control sequences.

Background

This manual was developed as part of the documentation for a variety of NI-488M device driver kits. Common software reference material can be found in this manual. Software and hardware information specific to a particular interface board kit can be found in other documentation provided with that kit.

Customer Support

We hope your experience will be a rewarding one. If you encounter difficulties, National Instruments has a helpful staff of applications engineers ready to assist you with system setup or software development. You can use the following toll-free numbers between the hours of 8:00 a.m. and 5:30 p.m. (Central Time) to reach the National Instruments applications engineering department:

(512) 794-0100
(800) IEEE-488 (toll-free U.S. and Canada)

Contents

Tables

Chapter 1
Introduction

The National Instruments NI-488 Multitasking Handler is a UNIX device driver that supports concurrent access to different GPIB devices from separate application programs. The NI-488M software contains a C language library (`cib`) that provides a uniform programming interface for these application programs; an interactive control program (`ibic`) that is a controlled environment for testing and debugging GPIB programs; and a screen-oriented, interactive configuration program (`ibconf`) that changes default device parameters.

Note: In this manual, the term *UNIX* is used generally to refer to any operating system supported by the NI-488M package (UNIX, SunOS, XENIX, and so on).

The Device Driver

National Instruments provides the device driver in object module form. It is linked into the UNIX system using reconfiguration tools provided by the system vendor (see the installation guide or getting started manual that you received with your interface board for specific details). The driver supports multiple GPIB boards and multiple GPIB devices. Access to GPIB devices is controlled by the driver, which uses internal tables of device-specific information. Standard drivers support up to two boards and 16 devices. A device can be assigned any board to use as its access board. The utility `ibconf` is used to edit the internal board and device tables.

The device tables contain information such as GPIB primary and secondary addresses, end-of-string modes, and timeout limits. Once this information is properly set up, it is possible to communicate with GPIB devices without any knowledge of GPIB protocol. `stdout` can be redirected to a GPIB device as in `cat file > /dev/gpibplotter`. However, `stdin` cannot.

Different processes can access different devices concurrently; however, each device can be opened by only one process at a time. A typical application would be a GPIB printer and GPIB plotter each with its own spooler. The driver imposes no restrictions on access to the GPIB board itself, so it is up to the system administrator to restrict read/write access to the board nodes (for example, mode 600 owned by root).

The driver uses the minor device number to determine whether a node is a board or a device. Boards are assigned the highest minor device numbers and devices are assigned starting at one. In systems with two boards and 16 devices, minor device numbers 1 through 16 reference the 16 devices and minor numbers 254 and 255 reference the two boards. The first board is 255, the second is 254. Once the first node (`/dev/gpib0`) is created, the configuration utility (`ibconf`) creates and renames the other nodes.

In special cases, more direct control of GPIB protocol is needed. For these cases, the user can access the GPIB board directly. In this mode of operation, the user is responsible for all addressing, unaddressing, and polling operations. This is privileged access since direct board commands will interfere with device-level commands.

In most applications, it is not necessary to perform any direct board commands. When the first device is opened, its access board is brought online and, if it is System Controller (SC), the driver

sends Interface Clear (IFC). The driver also sends IFC whenever a GPIB bus error is detected. If the GPIB board has been configured so it is not SC, device commands cannot be sent until the board is passed control. When the last device is closed, the GPIB board is taken offline.

When opened to perform device commands, the driver will respond to a service request (SRQ) interrupt by serial polling all open devices. Serial poll status bytes are stored in a queue per device. Since serial polling is done automatically, it is possible for a process to wait for its device to request service without impeding other GPIB activity.

Chapter 2
The C Language Library

This chapter describes the GPIB device and GPIB board commands.

Global Variables

The next several paragraphs explain the status word (`ibsta`), the error variable (`iberr`), and the count variable (`ibcnt`). These variables are updated each time a handler call is made, to reflect the status of the most recently referenced board or device.

Status Word – ibsta

All functions return a status word which reports the success of the function call and information about the state of the GPIB. The status word is also available as the external variable `ibsta`.

The status word contains 16 bits. The valid bits depend on whether a function is a device or a board call. A bit value of one indicates that the corresponding condition is in effect; a bit value of zero indicates that the condition is not in effect. Table 2-1 lists the conditions and the bit position to be tested for that condition.

Table 2-1. Status Word Layout

Mnemonic	Bit Pos.	Hex Value	Function Type	Description
ERR	15	8000	dev, brd	GPIB error
TIMO	14	4000	dev, brd	Time limit exceeded
END	13	2000	dev, brd	END or EOS detected
SRQI	12	1000	brd	SRQ interrupt received
RQS	11	800	dev	Device requesting service
CMPL	8	100	dev, brd	I/O completed
LOK	7	80	brd	Lockout State
REM	6	40	brd	Remote State
CIC	5	20	brd	Controller-In-Charge
ATN	4	10	brd	Attention is asserted
TACS	3	8	brd	Talker
LACS	2	4	brd	Listener
DTAS	1	2	brd	Device Trigger State
DCAS	0	1	brd	Device Clear State

A description of each status word and its condition follows.

ERR The ERR bit is set in the status word following any call that results in an error; the particular error can be determined by examining the `iberr` variable. The ERR bit is cleared following any call that does not result in an error.

TIMO The TIMO bit indicates whether the time limit has been exceeded.

END The END bit indicates whether the END or EOS message has occurred during a read operation.

SRQI The SRQI bit indicates whether the GPIB line SRQ is asserted.

RQS The RQS bit indicates that the device is requesting service.

CMPL The CMPL bit indicates that the previous I/O operation is complete. Since I/O is synchronous, CMPL is always set.

LOK The LOK bit indicates whether the board is in a lockout state.

REM The REM bit indicates whether the board is in remote state.

CIC The CIC bit indicates whether the GPIB board is the Controller-In-Charge.

ATN The ATN bit indicates whether the GPIB line ATN is asserted.

TACS The TACS bit indicates whether the GPIB board is addressed to talk.

LACS The LACS bit indicates whether the GPIB board is addressed to listen.

DTAS The DTAS bit indicates whether the GPIB board has detected a device trigger command.

DCAS The DCAS bit indicates whether the GPIB board has detected a device clear command.

Error Variable – iberr

When the ERR bit is set in the status word, a GPIB error has occurred. One of the following error codes is returned in the external variable `iberr.`

Table 2-2. GPIB Error Codes

Suggested Mnemonic	Decimal Value	Explanation
EDVR	0	UNIX error (code in `ibcnt`)
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	Write handshake error (e.g., no listener)

(continues)

Table 2-2. GPIB Error Codes (continued)

Suggested Mnemonic	Decimal Value	Explanation
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted
ENEB	7	Non-existent GPIB board
EDMA	8	DMA hardware error
EBTO	9	DMA hardware bus timeout
ECAP	11	No capability for type of operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial Poll queue overflow
ESRQ	16	SRQ line asserted by unknown device

A description of each error and some conditions under which it may occur follow:

EDVR (0) This code is returned by the language interface when an error is returned from the operating system. When this error occurs, additional error information is placed in the count variable `ibcnt`. In most cases, this is the UNIX error code that was supplied by the variable `errno`. In other cases, it contains a system-specific internal error code.

ECIC (1) This code is returned when a call requiring the GPIB board to be Controller-In-Charge (CIC) is made, but the board is not CIC. This could have happened because the board was never made CIC, or it passed control to another Controller.

ENOL (2) The most common cause of this error code is when a write operation is attempted with no Listeners addressed. For a device write, this indicates that the GPIB address configured for that device in the handler does not match the GPIB address of any device connected to the bus. For a board write, the appropriate addressing commands were not previously sent.

This error may also occur in situations in which the GPIB board is not the Controller-In-Charge and the Controller asserts ATN before the write call in progress has ended.

EADR (3) This error results from the GPIB board not addressing itself before read and write calls when it is the Controller-In-Charge. This error can occur on a device call if board calls are mixed with device calls.

EARG (4) This error results when an invalid argument is passed to a function call.

ESAC (5) This error results when `ibsic` or `ibsre` is called and when the GPIB board does not have System Controller capability.

EABO (6) This error indicates that I/O has been cancelled. This is usually due to a timeout condition.

ENEB (7) This error occurs when there is no GPIB board at the I/O address specified in the configuration program.

EDMA (8) This error indicates that a DMA hardware error occurred during an I/O operation.

EBTO (9) This error indicates that a hardware bus timeout occurred during an I/O operation. This is usually the result of an attempt by a DMA Controller to access non-existent memory.

ECAP (11) This error results when a particular capability is unsupported or has been disabled in the handler, and a call is made that attempts to make use of that capability.

EFSO (12) This error results when an `ibrdf` or `ibwrtf` call encounters a problem performing a file operation.

EBUS (14) This error indicates a GPIB bus error during a device call. This is usually the result of the time limit being exceeded.

ESTB (15) This error indicates that a devices serial poll byte queue has become full and bytes have been discarded for lack of room.

ESRQ (16) This error indicates that the SRQ line remains asserted after all open devices have been serial polled. This is usually the result of an incorrect device address.

Count Variable – ibcnt

The `ibcnt` variable is updated after each read, write, or command function call with the number of bytes actually transferred by the operation.

Read and Write Termination

The IEEE-488 specification defines two methods of identifying the last byte of device-dependent (data) messages. The two methods permit a Talker to send data messages of any length without the Listener(s) knowing in advance the number of bytes in the transmission. The two methods are as follows:

- END message. In this method, the Talker asserts the End Or Identify (EOI) signal simultaneously with transmission of the last data byte. By design, the Listener stops reading when it detects a data message accompanied by EOI, regardless of the value of the byte.
- End-Of-String (EOS) character. In this method, the Talker uses a special character at the end of its data string. By prior arrangement, the Listener stops receiving data when it detects that character. Either a 7-bit ASCII character or a full 8-bit binary byte can be used.

These two methods can be used individually or in combination. However, it is important that the Listener be properly configured to unambiguously detect the end of a transmission.

The GPIB board always terminates ibrd operations on the END message. Using the configuration program, you can accommodate all permissible forms of read and write termination. The default configuration settings for read and write termination can be changed at run time using the ibeos and ibeot functions, if necessary.

Compiling C Programs

Always include the file <sys/ugpib.h> in every GPIB program. This file defines all status bits, error codes, and externals needed. Be sure to use the GPIB library cib when compiling. If, for example, the library /lib/libg.a was created, as described in the installation guide or getting started manual that you received with your interface board, programs can be compiled with the following command:

```
cc prog.c -lg
```

For more information, refer to cc(1) or the equivalent in your UNIX documentation. Also refer to the Getting Started or Installation Guide that you received with your GPIB interface board kit.

GPIB Function Descriptions

The remainder of this chapter is intended as a quick reference to the GPIB device and GPIB board functions. Refer to *Appendix A* for more thorough information and specific examples.

Device Commands

The following functions can be performed on a GPIB device:

ibbna(d,bname)	Changes the access board for device d. bname must be a valid interface board. This function is a combination of IBGET and IBSET.
ibclr(d)	Sends the message Selected Device Clear (SDC) to a device.
ibeos(d,v)	Changes the End-Of-String (EOS) mode. The low byte contains the EOS character and the high byte is any of REOS, XEOS, or BIN. v equal to 0 disables EOS checking. This function is a combination of IBGET and IBSET.
ibeot(d,v)	Enables sending END with the last byte of every GPIB write. v equal to 0 disables. This function is a combination of IBGET and IBSET.
ibfind(dname)	Opens a device for reading and writing. The /dev/ prefix on the name is optional. It returns the file descriptor for subsequent calls (or -1 for an error).
ibllo(d)	Places a device in local lockout mode.
ibloc(d)	Returns a device to local mode.
ibonl(d,v)	Reinitialize GPIB software. v equal to 0 performs a close operation.

`ibpad(d,v)`	Changes the value of the primary GPIB address. This function is a combination of `IBGET` and `IBSET`.
`ibpct(d)`	Will pass control to a device.
`ibppc(d,v)`	Configures a device for a parallel poll.
`ibrd(d,buf,cnt)`	Reads from the GPIB into a buffer.
`ibrdf(d,fname)`	Reads from the GPIB into a file.
`ibrpp(d,buf)`	Executes a parallel poll.
`ibrsp(d,buf)`	Performs a serial poll of a device or returns the first value from a serial poll queue.
`ibsad(d,v)`	Changes the secondary address. `v` equal to 0 disables secondary address recognition. This function is a combination of `IBGET` and `IBSET`.
`ibtmo(d,v)`	Changes the timeout value. `v` equal to 0 disables timeouts. Timeout values are given in `ugpib.h`. This function is a combination of `IBGET` and `IBSET`.
`ibtrg(d)`	Triggers a device (sends GET).
`ibwait(d,mask)`	Waits for events to occur. Valid events are TIMO and RQS.
`ibwrt(d,buf,cnt)`	Writes from a buffer to the GPIB.
`ibwrtf(d,fname)`	Writes from a file to the GPIB.
`ioctl(d,IBGET, &device)`	Gets device parameters.
`ioctl(d,IBSET &device)`	Sets device parameters.

Board Commands

The following commands can be performed on a GPIB board:

`ibcac(b,v)`	Takes a board from Controller Standby to Active Controller state (asserts ATN). If `v` equals 1, the take control is synchronous, otherwise asynchronous. The board must be CIC.
`ibcmd(b,buf,cnt)`	Sends a buffer of command messages. The board must be CIC, but need not be Active Controller.
`ibdma(b,v)`	Changes the DMA mode. Refer to the installation guide or getting started manual that you received with your interface board for specific values. This function is a combination of `IBGET` and `IBSET`.

`ibeos(b,v)`	Changes the end-of-string (EOS) mode. The low byte contains the eos character and the high byte is any of `REOS`, `XEOS`, or `BIN`. `v` equal to 0 disables EOS checking. This function is a combination of `IBGET` and `IBSET`.
`ibeot(b,v)`	Enables sending END with the last byte of every GPIB write. A value of 0 disables. This function is a combination of `IBGET` and `IBSET`.
`ibfind(bname)`	Opens a board for reading and writing. The `/dev/` prefix on the name is optional. It returns the file descriptor for subsequent calls or -1 for an error.
`ibgts(b,v)`	Puts a board in standby state (unasserts ATN). If `v` equals 1, the board listens in continuous mode (should always be followed by a wait for END).
`ibist(b,v)`	Changes the value of a board's individual status (ist) bit. A value of 0 clears. This function is a combination of `IBGET` and `IBSET`.
`ibloc(b)`	Returns a board to local mode.
`ibonl(b,v)`	Reinitializes GPIB software. `v` equal to 0 performs a close operation.
`ibpad(b,v)`	Changes the value of the primary GPIB address. This function is a combination of `IBGET` and `IBSET`.
`ibppc(b,v)`	Configures a board for a parallel poll. This function is a combination of `IBGET` and `IBSET`.
`ibrd(b,buf,cnt)`	Reads from the GPIB into a buffer. The board must have been previously addressed to listen.
`ibrdf(b,fname)`	Reads from the GPIB into a file. If performed on a board, the board must have been previously addressed to listen.
`ibrpp(b,buf)`	Executes a parallel poll. The board must be CIC.
`ibrsc(b,v)`	Requests or releases system control. This function is a combination of `IBGET` and `IBSET`.
`ibrsv(b,v)`	Sets the serial poll response byte of the board. If bit 0x40 is set, the board asserts SRQ. If the board is CIC, it will not assert SRQ.
`ibsad(b,v)`	Changes the secondary address. `v` equal to 0 disables secondary address recognition. This function is a combination of `IBGET` and `IBSET`.
`ibsgnl(b,mask)`	Requests that a UNIX signal be sent when any of the requested events occur. The mask is a bit vector of which only SRQI, LOK, REM, CIC, TACS, LACS, DTAS, and DCAS are valid.
`ibsic(b,v)`	Pulses Interface Clear (IFC). If `v` equals 2, IFC remains asserted; v equal to 0 clears IFC; any other value pulses IFC.
`ibsre(b,v)`	Asserts Remote Enable (REN). `v` equal to 0 clears.

`ibtmo(b,v)`	Changes the timeout value. `v` equal to 0 disables timeouts. Timeout values are given in `ugpib.h`. This function is a combination of `IBGET` and `IBSET`.
`ibwait(b,mask)`	Waits for events to occur. Valid events for a board are: TIMO, END, SRQI, LOK, REM, CIC, TACS, LACS, DTAS, and DCAS.
`ibwrt(b,buf,cnt)`	Writes from a buffer to the GPIB. The board must have been previously addressed to talk.
`ibwrtf(b,fname)`	Writes from a file to the GPIB. The board must have been previously addressed to talk.
`ioctl(b,IBGET, &board)`	Get board parameters.
`ioctl(b,IBSET, &board)`	Set board parameters.

Chapter 3
Using ibic

The interface bus interactive control utility (`ibic`) is an interactive environment in which all of the commands contained in the `cib` library (except `IBGET`, `IBSET`, and `IBINFO`) can be run.

Refer to *Appendix A* for detailed descriptions of the C language functions.

Syntax Translation Guide

To translate between C syntax and `ibic` syntax, use the following guide:

- Omit the parentheses and first argument. `ibic` uses the current board or device for all of its calls. A new board or device is made the current board or device with the `ibfind` function. To return to a previous board or device use `set`. The `ibic` prompt displays the name of the current board or device.
- Functions with only one argument have no argument in `ibic`.

  `ibclr(d)` becomes: `ibclr`
- Functions with a single numeric argument are followed by a number.

  `ibsre(b,1)` becomes: `ibsre 1`
  `ibeos(b,(BIN|REOS)<<8|'\n')` becomes: `ibeos 0x140A`
- Functions that write a buffer are followed by a string, but no count.

  `ibwrt(b,"text",4)` becomes: `ibwrt "text"`
- Functions that read a buffer are followed by a count only.

  `ibrd(b,buf,50)` becomes: `ibrd 50`
- Functions that perform a poll take no argument.

  `ibrpp(b,buf)` becomes: `ibrpp`
- Functions that take a board, device, or filename are followed by a name without quotation marks.

  `ibfind("plotter")` becomes: `ibfind plotter`
- Functions that take a mask argument are followed by a list of mask bits in parentheses.

  `ibwait(d,TIMO|RQS)` becomes: `ibwait (timo rqs)`

Sample Session

The following is a sample session of `ibic` that triggers a digital voltmeter, waits for a service request, and reads in a buffer of data. User inputs are underlined.

```
: ibfind dvm
[0100]  ( cmpl )

dvm: ibclr
[0100]  ( cmpl )

dvm: ibwrt  "F3R7T3"
[0100] (  cmpl  )
count: 6

dvm: ibwait (rqs timo)
[0900]  ( rqs  cmpl  )

dvm: ibrsp
[0100]    (  cmpl  )
Poll: 0xC0

dvm: ibrd 10000
[2100]    (  end  cmpl  )
count: 10

01  02  03  04  05  06  25  07       . . . . . % .
62 03                                a .
```

Auxiliary Functions

Table 3-1 summarizes the auxiliary functions that `ibic` supports.

Table 3-1. Auxiliary Functions that ibic Supports

Function Syntax	Description
`set udname`	Change current device or board to a device or board already opened.
`help [option]`	Display help information. All available functions are briefly described.
`!`	Repeat previous command.
`-`	Turn printing *off*. This is most often used with the `$` command.
`+`	Turn printing *on*.
`n* function`	Execute command `n` times.
`n* !`	Execute previous command `n` times.
`$ filename`	Execute indirect file.
`print string`	Display string on screen.
`e`, `q`, or `^d`	Exit or quit `ibic`.

Chapter 4
Using ibconf

The interface bus configuration utility (`ibconf`) is a screen-oriented, interactive program. It is used to edit the device and board data structures in a UNIX kernel. `ibconf` also creates and renames device special files.

Before running `ibconf`, complete the following tasks:

- Become super-user; that is, log in as root.
- Make a backup copy of the UNIX kernel to be edited.
- Set the TERM environment variable correctly for the current terminal or console device.
- If you have not already done so, create a device special file for the first interface board, `/dev/gpib0` (refer to the Getting Started manual or Installation Guide that came with your interface board).

Run `ibconf` by entering `ibconf`, followed by the name of the UNIX kernel, and pressing <Return>. The following example runs `ibconf` on the UNIX kernel `/unix`.

```
ibconf /unix
```

`ibconf` first displays a map of GPIB devices and their access board. Use the cursor keys indicated on the display to position the cursor on the desired board or device. Then use the edit key to move to a new screen where software configuration parameters can be edited. Online help, which is available at all times, explains how to use `ibconf`, what the legal settings are, and what these settings mean.

For a detailed explanation of the operation of `ibconf`, refer to *ibconf(1)* in Appendix A.

Appendix A
NI-488M Functions and Utilities Reference

This appendix is a comprehensive reference for the functions and utilities contained in the NI-488M software package. This material is presented in a format familiar to most users of the UNIX operating system.

IBCONF(1) GPIB IBCONF(1)

Name

`ibconf` - interface bus configuration utility

Synopsis

```
ibconf [filename]
```

Description

`ibconf` is a screen-oriented, interactive program. It is used to edit the description of characteristics of the GPIB devices and boards in the system. The settings created by `ibconf` become the default settings for a device or board when it is first opened.

Commands

The following special commands are used to move between screens and between fields within a particular screen. (The caret (`^`) symbol represents the <Ctrl> key.)

`^b` – previous screen
When editing board or device parameters, `^b` moves to the previous board or device.

`^f` – next screen
When editing board or device parameters, `^f` moves to the next board or device.

`^i` – edit device
`^i` causes the screen to change from the device map page to the board or device characteristics page so that you can examine or change values associated with the board or device indicated by the current cursor position.

`^o` – return
If you are on the board or device characteristics page, `^o` returns you to the device map page. If you are on the device map page, `^o` lets you exit `ibconf.` In the latter case, the user will have the option of either saving or ignoring the changes made in the current editing session.

`^q` – help
`^q` displays help information.

`^r` – rename
If you are on the device map page, `^r` lets you rename the device indicated by the current cursor position. The user can give a device a new name of up to 14 characters. Although the device map page will only display the first seven characters, all 14 will be saved as that name of the device.

`^t` – (dis)connect
If you are on the device map page, `^t` has the effect of either connecting or disconnecting the selected device to or from the currently active board.

`^w` – explain field
On the board or device characteristics page, `^w` explains the current field.

`^y` – reset value
On the board or device characteristics page, before leaving a field, `^y` will reset the value to what it was when the field was first entered.

`h` – cursor left

`H` – home

`j` – cursor down

`k` – cursor up

`l` – cursor right

See Also
ibic(1) and *cib(3)*
Chapter 2, *The C Language Library*

IBIC(1) GPIB IBIC(1)

Name

`ibic` - interface bus interactive control program

Synopsis

`ibic`

Description

`ibic` is a command language for controlling the National Instruments GPIB interface. It executes commands read from `stdin` or a file and returns detailed status information. All commands from the GPIB library `cib` are supported.

Commands

Commands are directed to the current board or device. New boards and devices are opened with `ibfind.` Once opened, the current board or device is changed with `set`. Table A-1 summarizes the NI-488M functions and syntax when called from `ibic`.

Table A-1. Syntax of NI-488M Functions in ibic

Description	Function Syntax	Function Type	Note
Change access board of device	`ibbna brdname`	dev	1
Become active controller	`ibcac [v]`	brd	2,3
Clear specified device	`ibclr`	dev	
Send commands from string	`ibcmd string`	brd	4
Enable/disable DMA	`ibdma v`	brd	2,3
Change/disable EOS message	`ibeos v`	dev, brd	2,3
Enable/disable END message	`ibeot v`	dev, brd	2,3
Return unit descriptor	`ibfind bdname`	dev, brd	5
Go from active controller to standby	`ibgts v`	brd	2,3
Set/clear ist	`ibist v`	brd	2,3
Place device in local lockout state	`ibllo`	dev	
Go to local	`ibloc`	dev, brd	
Place device online or offline	`ibonl [v]`	dev, brd	2,3
Change primary address	`ibpad v`	dev, brd	3
Pass control	`ibpct`	dev	
Parallel poll configure	`ibppc v`	dev, brd	3
Read data	`ibrd v`	dev, brd	6

(continues)

Table A-1. Syntax of NI-488M Functions in ibic (continued)

Description	Function Syntax	Function Type	Note
Read data to file	`ibrdf fnam`	dev, brd	7
Conduct a parallel poll	`ibrpp`	dev, brd	
Request/release system control	`ibrsc [v]`	brd	2,3
Return serial poll byte	`ibrsp`	dev	
Request service	`ibrsv v`	dev	3
Change secondary address	`ibsad v`	dev, brd	3
Request UNIX signal on specific events	`ibsgnl [mask]`	brd	
Send interface clear	`ibsic [v]`	brd	3
Set/clear remote enable line	`ibsre [v]`	brd	2,3
Abort asynchronous operation	`ibstop`	dev, brd	
Change/disable time limit	`ibtmo v`	dev, brd	3
Trigger selected device	`ibtrg`	dev	
Wait for selected event	`ibwait [mask]`	dev, brd	2,8
Write data	`ibwrt string`	brd	4
Write data to file	`ibwrtf fnam`	dev, brd	7

Notes

1. `brdname` is the symbolic name of the new board (for example, `ibbna gpib1`).

2. Values enclosed in square brackets (`[]`) are optional. The default value is zero for `ibwait` and 1 for all other functions.

3. `v` is a hex, octal, or decimal integer. Hex numbers must be preceded by zero and x (for example, `0xD`). Octal numbers must be preceded by zero only (for example, `015`). Other numbers are assumed to be decimal.

4. `string` consists of a list of ASCII characters, octal or hex bytes, or special symbols. The entire sequence of characters must be enclosed in quotation marks. An octal byte consists of a backslash character followed by the octal value. For example, octal 40 would be represented by `\40`. A hex byte consists of a backslash character and a character `x` followed by the hex value. For example, hex 40 would be represented by `\x40`. The two special symbols are `\r` for a carriage return character and `\n` for a linefeed character. These symbols provide a more convenient method for inserting the carriage return and linefeed characters into the string, as shown in the following string: `"F3R5T1\r\n"`. Since the carriage return can be represented equally well in hex, `\xD` and `\r` are equivalent strings.

5. `bdname` is the symbolic name of the new device or board (for example, `ibfind dev1` or `set gpib0`).

6. `v` is the number of bytes to read.

7. `fnam` is the UNIX pathname of the file to be read or written (for example, `\test\meter` or `printr.buf`).

8. `mask` is a hex, octal, or decimal integer (see note 3) or a mask bit mnemonic.

Return Values

All `ibic` functions return a status word in both hex and bit mnemonic form. Table A-2 lists the mnemonics of the status word. (This is the same information that is given in Table 2-1.)

Table A-2. Status Word Layout

Mnemonic	Bit Pos.	Hex Value	Function Type	Description
ERR	15	8000	dev, brd	GPIB error
TIMO	14	4000	dev, brd	Time limit exceeded
END	13	2000	dev, brd	END or EOS detected
SRQI	12	1000	brd	SRQ interrupt received
RQS	11	800	dev	Device requesting service
CMPL	8	100	dev, brd	I/O completed
LOK	7	80	brd	Lockout State
REM	6	40	brd	Remote State
CIC	5	20	brd	Controller-In-Charge
ATN	4	10	brd	Attention is asserted
TACS	3	8	brd	Talker
LACS	2	4	brd	Listener
DTAS	1	2	brd	Device Trigger State
DCAS	0	1	brd	Device Clear State

If the ERR bit is set, an error mnemonic will be displayed as shown in Table A-3. (This is the same information that is given in Table 2-2.)

Table A-3. GPIB Error Codes

Suggested Mnemonic	Decimal Value	Explanation
EDVR	0	UNIX error (code in `ibcnt`)
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	Write handshake error (e.g., no listener)
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted
ENEB	7	Non-existent GPIB board
EDMA	8	DMA hardware error
EBTO	9	DMA hardware bus timeout
ECAP	11	No capability for type of operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial Poll queue overflow
ESRQ	16	SRQ line asserted by unknown device

Auxiliary Functions

Table A-4 summarizes the auxiliary functions that `ibic` supports. (This is the same information that is given in Table 3-1.)

Table A-4. Auxiliary Functions that ibic Supports

Function Syntax	Description
`set udname`	Change current device or board to a device or board already opened with `ibfind`.
`help [option]`	Display help information. All available functions are briefly described.
`!`	Repeat previous command.
`-`	Turn printing *off*. This is most often used with the `$` command.
`+`	Turn printing *on*.

(continues)

Table A-4. Auxiliary Functions that ibic Supports (continued)

Function Syntax	Description
`n* function`	Execute command `n` times.
`n* !`	Execute previous command `n` times.
`$ filename`	Execute indirect file.
`print string`	Display string on screen.
`e`, `q`, or `^d`	Exit or quit `ibic`.

Files

```
/dev/gpib?
/dev/*
```

See Also

ibconf(1) and *cib(3)*
Chapter 2, *The C Language Library*

Bugs

Catches only `SIGINT`. If `ibsgnl` uses a different signal, `ibic` will die.

IBBNA(3) device only IBBNA(3)

Name

`ibbna` - change access board of the device

Synopsis

```
#include <sys/ugpib.h>
ibbna (d,bname)
int d;
char *bname;
```

Description

`d` is a file descriptor returned from an `ibfind` call. `bname` is a null-terminated string corresponding to a GPIB board special file.

The `ibbna` function assigns the board that will be used in subsequent device-level functions.

The assignment made by this function remains in effect until `ibbna` is called again, the `ibonl` function is called, or the file is closed. The original configuration is not permanently changed.

Example

Associate device `dvm` with interface board `gpib0`.

```
ibbna (dvm,"gpib0");
```

See Also

ibget(3), *ibfind(3)*, and *ibonl(3)*.
Chapter 2, *The C Language Library*

IBCAC(3) board only IBCAC(3)

Name

`ibcac` - become Active Controller

Synopsis

```
#include <sys/ugpib.h>
ibcac (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` identifies the type of take control.

If `v` is non-zero, the GPIB board takes control synchronously with respect to data transfer operations; otherwise, the GPIB board takes control immediately (and possibly asynchronously).

To take control synchronously, the GPIB board waits before asserting the ATN signal so that data being transferred on the GPIB will not be corrupted. If a data handshake is in progress, the take control action is postponed until the handshake is complete; if a handshake is not in progress, the take control action is done immediately. Synchronous take control is not guaranteed if an `ibrd` or `ibwrt` operation completed with a timeout or error.

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

It is generally not necessary to use the `ibcac` function. Functions, such as `ibcmd` and `ibrpp` (which require that the GPIB board take control), take control automatically.

The ECIC error results if the GPIB board is not Controller-In-Charge.

Examples

1. Take control immediately without regard to any data handshake in progress.

```
ibcac(brd0,0);
```

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

```
ibrd(brd0,rd,512);
ibcac(brd0,1);
```

IBCLR(3) device only IBCLR(3)

Name

`ibclr` - send Selected Device Clear (SDC)

Synopsis

```
#include <sys/ugpib.h>
ibclr (d)
int d;
```

Description

`d` is a file descriptor returned from an `ibfind` call.

The `ibclr` function sends the message SDC, the meaning of which depends on the specific device. SDC usually resets all device functions. `ibclr` sends the following commands and information:

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

Example

Clear device `vmtr`.

```
ibclr(vmtr);
```

See Also

ibcmd(3) and *ibgts(3)*
Chapter 2, *The C Language Library*

IBCMD(3) board only IBCMD(3)

Name

`ibcmd` - send command message to GPIB

Synopsis

```
#include <sys/ugpib.h>
ibcmd (b,cmd,cnt)
int b,cnt;
char cmd[ ];
```

Description

`b` is a file descriptor returned from an `ibfind` call. `cmd` contains the commands to be sent over the GPIB. `cnt` specifies the number of bytes to be sent over the GPIB.

The `ibcmd` function is used to transmit interface messages (commands) over the GPIB. These commands, which are listed in *Appendix B*, include device talk and device listen addresses, secondary addresses, serial and parallel poll configuration messages, and device clear and device trigger instructions. The `ibcmd` function is also used to pass GPIB control to another device. This function is *not* used to transmit programming instructions to devices; programming instructions and other device-dependent information are transmitted with the `ibwrt` function.

The `ibcmd` operation terminates on any of the following events:

- All commands are successfully transferred.
- Error is detected.
- Time limit is exceeded.
- Take Control (TCT) command is sent.
- Interface Clear (IFC) message is received from the System Controller (not the GPIB board).

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

An ECIC error results if the GPIB board is not Controller-In-Charge. If it is not Active Controller, it takes control and asserts ATN prior to sending the command bytes. It remains Active Controller afterward.

In the examples that follow, GPIB commands and addresses are coded as printable ASCII characters. When the hex values to be sent over the GPIB correspond to printable ASCII characters, this is the simplest means of specifying the values. Refer to *Appendix B* for conversions of hex values to ASCII characters.

Examples

1. Unaddress all Listeners with the Unlisten command (ASCII ?) and address a Talker at 0x46 (ASCII F) and a Listener at 0x31 (ASCII 1).

```
ibcmd(brd0,"?F1",3);       /* UNL TAD LAD                    */
```

2. Unaddress all Listeners with the Unlisten command (ASCII ?) and address a Talker at 0x46 (ASCII F) and a Listener at 0x6E (ASCII n).

```
ibcmd(brd0,"?F1n",4);      /* UNL TAD LAD SAD                */
```

3. Clear all GPIB devices (that is, reset internal functions) with the Device Clear (DCL) command (0x14).

```
ibcmd(brd0,"\024",1);      /* DCL                            */
```

4. Clear two devices with Listen addresses of 0x21 (ASCII !) and 0x28 (ASCII () with the Selected Device Clear (SDC) command (0x4).

```
ibcmd(brd0,"?!(\004",4);   /* UNL LAD LAD SDC                */
```

5. Trigger any devices previously addressed to listen with the Group Execute Trigger (GET) command (0x8).

```
ibcmd(brd0,"\010",1);      /* GET                            */
```

6. Unaddress all Listeners and serially poll a device at talk address 0x52 (ASCII R) using the Serial Poll Enable (0x18) and Serial Poll Disable (0x19) commands (the brd0 listen address is 0x20 or ASCII blank).

```
ibcmd(brd0,"?R \030",4);   /*UNL TAD MLA SPE                */
ibrd(brd0,rd,1);           /* read one byte                 */
/*  After checking the status byte in rd[0], disable this   */
/*  device and unaddress it with the Untalk (UNT) command   */
/*  (0x5F or ASCII _) before polling the next one.          */
ibcmd(brd0,"031_",2);      /*SPD UNT                        */
```

See Also

ibtrg(3), *ibpct(3)*, *ibclr(3)*, *ibrsp(3)*, *ibppc(3)*, *ibcac(3)*, *ibgts(3)*, *ibloc(3)*, *ibtmo(3)*, and *ibllo(3)*.
Chapter 2, *The C Language Library*

IBDMA(3) device or board IBDMA(3)

Name

`ibdma` - change DMA mode

Synopsis

```
#include <sys/ugpib.h>
ibdma (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` is the DMA mode.

Some GPIB boards support more than one type of DMA transfer. For these boards, `v` will select the DMA type. Consult the Getting Started or Installation Guide that you received with your GPIB interface board for more information.

The assignment made by this function remains in effect until `ibdma` is called again, `ibonl` is called, or the file is closed.

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

Examples

Change the DMA mode to hex 33.

```
ibdma(b,0x33)
```

See Also

ibset(3) and *ibonl(3)*.
Chapter 2, *The C Language Library*

IBEOS(3) device or board IBEOS(3)

Name

`ibeos` - change or disable end-of-string mode

Synopsis

```
#include <sys/ugpib.h>
ibeos (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` is the EOS mode.

`v` selects the EOS character and the data transfer termination method according to Table A-5. `ibeos` is needed only to alter the value from its configuration setting.

When `ibeos` is called and an error does not occur, the previous value of `v` is stored in `iberr`. The assignment made by this function remains in effect until `ibeos` is called again, the `ibonl` function is called, or the file is closed.

Table A-5. Data Transfer Termination Method

Method	Value of v	
	High Byte	Low Byte
A. Terminate read when EOS is detected.	0x04 (`REOS`)	EOS
B. Set EOI with EOS on write function.	0x08 (`XEOS`)	EOS
C. Compare all 8 bits of EOS byte rather than low 7 bits (all read and write functions).	0x04 (`BIN`)	EOS

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

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

When `bd` specifies a device, the options coded in `v` are used for all device-level reads and writes in which that device is specified.

When `bd` specifies a board, the options coded in `v` become associated with all board-level reads and writes.

Examples

1. Send END when the linefeed character is written for operations involving device dvm.

```
v = (XEOS<<8) | '\n';    /* or v = 0x080A                    */
beos(dvm,v);
wrt[0] = '1';            /* data bytes to be written         */
wrt[1] = '2';
wrt[2] = '3';
wrt[3] = '\n';           /* EOS character is last byte       */
ibwrt(dvm,wrt,4);
```

2. Program interface board brd0 to terminate a read on detection of the linefeed character ('\n'==0x0A) that is expected to be received within 512 bytes.

```
v = (REOS<<8) | '\n';   /* or v = 0x040A                     */
ibeos(brd0,v);
/* assume board has been addressed; do board level read     */
ibrd(brd0,rd,512);
/* The END bit in ibsta is set if the read terminated       */
/* on the EOS character, with the actual number of bytes    */
/* received contained in ibcnt.                             */
```

3. Program interface board brd0 to terminate read operations on the 8-bit value 0x82 rather than the 7-bit character 0x0A.

```
v = ((BIN | REOS)<<8) | 0x82;  /* or v = 0x1482             */
ibeos(brd0,v);
/* assume board has been addressed; do board level read     */
ibrd(brd0,rd,512);
/* The END bit in ibsta is set if the read terminated       */
/* on the EOS character, with the actual number of bytes    */
/* received contained in ibcnt.                             */
```

4. Disable read termination on receiving the EOS character for operations involving board brd0.

```
ibeos(brd0,0);          /* No EOS modes enabled             */
```

5. Send END with linefeeds and to terminate reads on linefeeds for operations involving board brd0.

```
v = ((REOS | XEOS)<<8) | 0x0A; /* or v = 0x180A             */
ibeos(brd0,v);
wrt[0] = '1';               /* data bytes to be written     */
wrt[1] = '2';
wrt[2] = '3';
wrt[3] = 0x0A;              /* EOS character is last byte   */
ibwrt(brd0,wrt,4);
```

See Also

ibset(3), *ibeot(3)*, and *ibonl(3)*
Chapter 2, *The C Language Library*

IBEOT(3) device or board IBEOT(3)

Name

`ibeot` - change or disable END termination mode

Synopsis

```
#include <sys/ugpib.h>
ibeot (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` is the EOS mode.

If `v` is non-zero, the END message is sent automatically with the last byte of each write operation. If `v` is zero, END is not sent. `ibeot` is needed only to alter the value from its configuration setting.

The END message is sent when the GPIB EOI signal is asserted during a data transfer. It is used to identify the last byte of a data string without having to use an End-Of-String character. `ibeot` is used primarily to send variable length binary data.

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

The assignment made by this function remains in effect until `ibeot` is called again, the `ibonl` function is called, or the file is closed.

Examples

1. Send the END message with the last byte of all subsequent writes to device `plotter`.

```
ibeot(plotter,1);          /* enable sending of EOI          */
/* It is assumed that wrt contains the data to be written   */
/* to the GPIB                                              */
ibwrt(plotter,wrt,3);      /* write 3 bytes                  */
```

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

```
ibeot(brd0,0);             /* disable sending EOI            */
```

See Also

ibset(3), *ibeos(3)*, and *ibonl(3)*
Chapter 2, *The C Language Library*

IBFIND(3) device or board IBFIND(3)

Name

`ibfind` - open GPIB special file

Synopsis

```
#include <sys/ugpib.h>
ibfind (name)
char *name;
```

Description

`name` specifies a null-terminated string corresponding to a GPIB device or GPIB board.

If the `name` begins with "`/dev/`", "`/dev/`" can be omitted. `ibfind` returns a file descriptor to be used in subsequent function calls. `ibfind` performs the equivalent of `ibonl(bd,1)` to open the specified device or board and initialize software parameters to their default configuration settings. The unit descriptor is valid until `ibonl(bd,0)` is used to place that device or board offline. If the `ibfind` call fails, a negative number is returned in place of the file descriptor.

Examples

1. Assign the unit descriptor associated with the device name "`fsdvm`" (Fluke Sampling Digital Voltmeter) to the variable `fsdvm`.

```
fsdvm = ibfind("fsdvm");
if (fsdvm < 0) error ();
```

2. Assign the unit descriptor associated with board "`gpib0`" to the variable `brd0`.

```
brd0 = ibfind("gpib0");
if (brd0 < 0) error ();
```

or

```
brd0 = ibfind("/dev/gpib0");
if (brd0 < 0) error();
```

See Also

ibbna(3) and *ibonl(3)*
Chapter 2, *The C Language Library*

IBGTS(3) board only IBGTS(3)

Name

`ibgts` - go from Active Controller to standby

Synopsis

```
#include <sys/ugpib.h>
ibgts (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` is the type of go-to-standby.

The `ibgts` function causes the GPIB board to go to the Controller Standby state and to unassert the ATN signal if it is the Active Controller. `ibgts` permits GPIB devices to transfer data without the GPIB board being a party to the transfer.

If `v` is non-zero, the GPIB board shadows data transfer handshakes as an Acceptor, and when the END message is detected, the GPIB board enters a Not Ready For Data (NRFD) handshake holdoff state on the GPIB. If `v` is zero, no shadow handshake or holdoff is done.

If the shadow handshake option is activated, the GPIB board participates in data handshake as an Acceptor without actually reading the data. It monitors the transfers for the END message and holds off subsequent transfers. This mechanism allows the GPIB board to take control synchronously on a subsequent operation such as `ibcmd` or `ibrpp`. `ibgts(b,1)` should always be followed by a wait for END (see *Example 2*).

The ECIC error results if the GPIB board is not Controller-In-Charge.

Examples

1. Turn the ATN line off.

```
ibgts(brd0,0);
```

2. Turn the ATN line off with the `ibgts` function after unaddressing all Listeners with the Unlisten (ASCII ?) command, addressing a Talker at 0x46 (ASCII F), and addressing a Listener at 0x31 (ASCII 1) to allow the Talker to send data messages.

```
ibcmd(brd0,"?F1",3);      /* UNL TAD LAD                    */
ibgts(brd0,1);            /* listen in continuous mode      */
ibwait(brd0,END|TIMO);
```

See Also

ibcmd(3), *ibcac(3)*, and *ibwait(3)*
Chapter 2, *The C Language Library*

IBIST(3) board only IBIST(3)

Name

`ibist` - set or clear individual status (ist) bit

Synopsis

```
#include <sys/ugpib.h>
ibist (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` is the sense of the ist bit.

If `v` is non-zero, the ist bit is set. If `v` is zero, the ist bit is cleared.

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

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

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

Examples

1. Set the individual status bit.

```
ibist(brd0,1);
```

2. Clear the individual status bit.

```
ibist(brd0,0);
```

See Also

ibset(3), *ibppc(3)*, and *ibrpp(3)*
Chapter 2, *The C Language Library*

IBLLO(3) device only IBLLO(3)

Name

`ibllo` - place device in Local Lockout state

Synopsis

```
#include <sys/ugpib.h>
ibllo (d)
int d;
```

Description

`d` is a file descriptor returned from an `ibfind` call.

The `ibllo` function sends the message LLO, which places a device in the Local Lockout state. This usually inhibits recognition of front panel input.

All devices are unaddressed. `ibllo` sends the following commands and information.

- Listen address of the device
- Secondary address of the device, if applicable
- Local Lockout (LLO)
- Unlisten

Example

Place device `vmtr` in Local Lockout state.

```
ibllo(vmtr);
```

See Also

ibcmd(3)
Chapter 2, *The C Language Library*

IBLOC(3) device or board IBLOC(3)

Name

`ibloc` - go to local mode

Synopsis

```
#include <sys/ugpib.h>
ibloc (bd)
int bd;
```

Description

`bd` is a file descriptor returned from an `ibfind` call.

`ibloc` is used to move devices temporarily from a remote program mode to a local mode. A device enters remote mode when the REN line is asserted and the device detects its listen address.

If `bd` specifies a GPIB device, `ibloc` places the indicated device in local mode by sending the following command sequence:

1. Untalk
2. Unlisten
3. Listen address of the device
4. Secondary address of the device, if applicable
5. Go To Local (GTL)
6. Unlisten

If `bd` specifies an interface board, the board is placed in a local state by sending the local message Return To Local (rtl), provided it is not locked in remote mode (indicated by the LOK bit of the status word, `ibsta`). This `ibloc` function is used to simulate a front panel return to local switch when the computer is used as an instrument.

Examples

1. Return device `dvm` to local state.

```
ibloc(dvm);
```

2. Return board `brd0` to local state.

```
ibloc(brd0);
```

See Also

ibsre(3) and *ibllo(3)*
Chapter 2, *The C Language Library*

IBONL(3) device or board IBONL(3)

Name

`ibonl` - place device or board online or offline

Synopsis

```
#include <sys/ugpib.h>
ibonl (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` specifies online or offline.

`ibonl` reinitializes all software as though bringing the device or board online for the first time. If `bd` specifies a board, the GPIB hardware will be reset. If `v` is zero, the GPIB board will be left offline, not participating in GPIB activity.

Call `ibonl` with `v` non-zero to reset a board or device to its power-on state. Call `ibonl` with `v` zero only when finished with a board or device, as this also closes the file descriptor.

Examples

1. Disable device `plotter`.

   ```
   ibonl(plotter,0);
   ```

2. Re-enable device `plotter` after taking it offline temporarily.

   ```
   ibonl(plotter,0);
   plotter = ibfind("plotter");
   ```

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

   ```
   ibonl(plotter,1);
   ```

4. Disable interface board `brd0`.

   ```
   ibonl(brd0,0);
   ```

See Also

ibfind(3)
Chapter 2, *The C Language Library*

IBPAD(3) device or board IBPAD(3)

Name

`ibpad` - change primary address

Synopsis

```
#include <sys/ugpib.h>
ibpad (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` specifies the primary GPIB address.

`ibpad` is used to alter the value from its configuration setting. A device listen address is formed by adding 0x20 to the primary address; the talk address is formed by adding 0x40 to the primary address.

Only the low five bits of `v` are significant and they must be in the range from 0 to 0x1E.

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

The assignment made by this function remains in effect until `ibpad` is called again, the `ibonl` function is called, or the file is closed.

When `bd` specifies a device, `ibpad` determines the talk and listen addresses based on the value of `v` for use in all I/O directed to that device. The actual GPIB address of any device is set within that device, either with hardware switches, or a software program. Refer to your device documentation for instructions.

When `bd` specifies a board, `ibpad` programs the interface board to respond to the primary talk and primary listen address indicated by `v`.

Examples

1. Change the primary GPIB listen and talk address of the device `plotter` from the configuration setting to 0x2A and 0x4A, respectively.

   ```
   ibpad(plotter,0xA);
   ```

2. Change the primary GPIB listen and talk address of board `brd0` from the configuration setting to 0x27 and 0x47, respectively.

   ```
   ibpad(brd0,7);
   ```

See Also

ibset(3) and *ibsad(3)*
Chapter 2, *The C Language Library*

IBPCT(3) device only IBPCT(3)

Name

`ibpct` - pass control

Synopsis

```
#include <sys/ugpib.h>
ibpct (d)
int d;
```

Description

`d` is a file descriptor returned from an `ibfind` call.

The `ibpct` function passes Controller-In-Charge authority to the specified device. The board automatically goes to a Controller idle state. The function assumes that the device has Controller capability.

`ibpct` sends the following commands and information.

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

Example

Pass control to device `pcxt`.

```
ibpct(pcxt);
```

See Also

ibcmd(3)
Chapter 2, *The C Language Library*

IBPPC(3) device or board IBPPC(3)

Name

`ibppc` - parallel poll configure

Synopsis

```
#include <sys/ugpib.h>
ibppc (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` is a valid parallel poll enable/disable command.

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

When `bd` specifies a device, the `ibppc` function enables or disables the device from responding to parallel polls.

`ibppc` sends the following commands and information.

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

Each of the 16 PPE messages specifies the GPIB data line (DIO1 through DIO8) and sense (one or zero) that the device must use when responding to the Identify (IDY) message during a parallel poll. The assigned message is interpreted by the device along with the current value of the individual status (ist) bit to determine if the selected line is driven true or false. For example, if PPE=0x64, DIO5 is driven true if ist=0 and false if ist=1. And if PPE=0x68, DIO1 is driven true if ist=1 and false if ist=0. Any PPD message or zero value cancels the PPE message in effect.

Which PPE and PPD messages are sent, and the meaning of a particular parallel poll response are all system-dependent protocol matters to be determined by the user.

When `bd` specifies an interface board, the board itself is programmed to respond to a parallel poll by setting its local poll enable (lpe) message to the value of `v`.

Examples

1. Configure device `dvm` to respond to a parallel poll by sending data line DIO3 true if ist=0.

   ```
   ibppc(dvm,0x62);
   ```

2. Configure device `dvm` to respond to a parallel poll by sending data line DIO1 true if ist=1.

```
ibppc(dvm,0x68);
```

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

```
ibppc(dvm,0x70);
```

4. Configure board `brd0` to respond to a parallel poll by sending data line DIO8 true if ist=0.

```
ibppc(brd0,0x67);
```

See Also

ibcmd(3) and *ibist(3)*
Chapter 2, *The C Language Library*

IBRD(3) device or board IBRD(3)

Name

`ibrd` - read data from the GPIB into a buffer

Synopsis

```
#include <sys/ugpib.h>
ibrd (bd,buf,cnt)
int bd,cnt;
char buf[ ];
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `buf` identifies the buffer to use. `cnt` specifies the number of bytes to read from the GPIB.

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

Device Call

When `bd` specifies a device, the following steps are performed:

1. The device is addressed to talk and the access board is addressed to listen, if not already addressed to do so.

2. The board reads the data from the device.

3. Attention (ATN) is re-asserted.

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

Board Call

When `bd` specifies an interface board, the `ibrd` function attempts to read from a GPIB device that is assumed to be already properly initialized and addressed.

If the board is Controller-In-Charge (CIC), the `ibcmd` function must be called prior to `ibrd` to address a device to talk and the board to listen. If the board is not CIC, the device on the GPIB that is the CIC must perform the addressing.

If the access board is Active Controller, the board is first placed in Standby Controller state, with ATN off, and remains there after the read operation is completed. An EADR error results if the board is CIC but has not been addressed to listen with the `ibcmd` function. An EABO error results if the board is not the CIC and is not addressed to listen within the time limit. An EABO error also results if the device that is to talk is not addressed and/or the operation does not complete for whatever reason within the time limit.

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

- Allocated buffer becomes full.
- Error is detected.
- Time limit is exceeded.
- END message is detected.
- EOS character is detected (if this option is enabled).
- Device Clear (DCL) or Selected Device Clear (SDC) command is received from another device which is the Controller-In-Charge.

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

Examples

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

```
ibrd(tape,rdbuf,56);
/* Check ibsta to see how the read terminated: on CMPL,     */
/* END, TIMO, or ERR.                                       */
/* Data is stored in rdbuf.                                 */
```

2. Read 56 bytes of data from a device at talk address 0x4C (ASCII L) and then unaddress it (the GPIB board is at listen address 0x20 or ASCII blank).

```
ibcmd(brd0,"?L ",3);    /* UNL TAD MLA                      */
ibrd(brd0,rdbuf,56);
/* Check ibsta to see how the read terminated: on CMPL,     */
/* END, TIMO, or ERR.                                       */
/* Data is stored in rdbuf.                                 */
/* Unaddress the Talker and Listener.                       */
ibcmd(brd0,"_?",1);     /* UNT UNL                          */
```

See Also

ibeos(3), *ibcmd(3)*, and *ibrdf(3)*
Chapter 2, *The C Language Library*

IBRDF(3) device or board IBRDF(3)

Name

`ibrdf` - read data from GPIB into a file

Synopsis

```
#include <sys/ugpib.h>
ibrdf (bd,fname)
int bd;
char fname;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `fname` specifies a null-terminated UNIX pathname.

The `ibrdf` function makes multiple calls to the `ibrd` function, terminating when END or EOS is received.

Example

Read data from device `tape` into file `tapedata`.

```
ibrdf(tape,"tapedata");
```

See Also

ibrd(3)
Chapter 2, *The C Language Library*

IBRPP(3) device or board IBRPP(3)

Name

`ibrrp` - conduct a parallel poll

Synopsis

```
#include <sys/ugpib.h>
ibrpp (bd,ppr)
int bd;
char *ppr;
```

Description

`bd` is a file descriptor returned from an `ibfind` call.

`ppr` identifies the address where the parallel poll response byte is stored.

The `ibrrp` function causes the identified board to conduct a parallel poll of previously configured devices by sending the Identify (IDY) message (ATN and EOI both asserted).

An ECIC error results if the GPIB board is not Controller-In-Charge. If the GPIB board is Standby Controller, it takes control and asserts ATN (becomes Active) prior to polling. It remains Active Controller afterward.

Examples

1. Remotely configure device `lcrmtr` to respond positively on DIO3 if its individual status bit is one, then parallel poll all configured devices on the same access board as `lcrmtr`.

```
ibppc(lcrmtr, 0x6A);
ibrpp(lcrmtr, &ppr);
```

2. Remotely configure a device at listen address 0x23 to respond positively on DIO3 if its individual status bit is one, and then parallel poll all configured devices.

```
cmd[0] = 0x23;          /* device listen address            */
cmd[1] = PPC;
cmd[2] = PPE | S | 2;   /* send PPR3 if ist = 1              */
cmd[3] = UNL;
ibcmd(brd0,cmd,4);
ibrpp(brd0,&ppr);       /* PPR returned in ppr              */
```

3. Disable and unconfigure all GPIB devices from parallel polling using the PPU command.

```
ibcmd(brd0,"\x15",1);   /* PPU                              */
```

See Also

ibcmd(3), *ibppc(3)*, and *ibist(3)*
Chapter 2, *The C Language Library*

IBRSC(3) board only IBRSC(3)

Name

`ibrsc` - request or release system control

Synopsis

```
#include <sys/ugpib.h>
ibrsc (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` specifies request or release system control.

If `v` is non-zero, functions requiring System Controller capability are subsequently allowed. If `v` is zero, functions requiring System Controller capability are not allowed.

The `ibrsc` function is used to enable or disable the capability of the GPIB board to send the Interface Clear (IFC) and Remote Enable (REN) messages to GPIB devices using the `ibsic` and `ibsre` functions. The interface board must not be System Controller to respond to Interface Clear sent by another Controller.

In most applications, the GPIB board will always be the System Controller. In other applications, the GPIB board will never be the System Controller. In either case, the `ibrsc` function is used only if the computer is not going to be System Controller for the duration of the program execution. While the IEEE-488 standard does not specifically allow schemes in which system control can be passed dynamically from one device to another, the `ibrsc` function would be used in such a scheme.

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

Example

Request to be System Controller if the board `brd0` is currently not so designated.

```
ibrsc(brd0,1);
```

See Also

ibset(3)
Chapter 2, *The C Language Library*

IBRSP(3) device only IBRSP(3)

Name

`ibrsp` - return serial poll status byte

Synopsis

```
#include <sys/ugpib.h>
ibrsp (d,spr)
int d;
char spr[];
```

Description

`d` is a file descriptor returned from an `ibfind` call. `spr` is the buffer in which the poll response is stored.

The `ibrsp` function is used to serial poll one device and obtain its status byte or to obtain a previously stored status byte. If the 0x40 bit of the response is set, the status response is positive, that is, the device is requesting service.

If automatic serial polling is enabled (default configuration), devices are polled the instant they request service. Positive responses are saved in a queue, causing the RQS bit of the responding device's status word to be set. A call to `ibrsp` when RQS is set returns the oldest previously acquired status byte. If the RQS bit of the status word is not set when `ibrsp` is called, the device is serial polled to obtain the status byte.

The interpretation of the response in `spr`, other than the RQS bit, is device-specific. For example, the polled device might set a particular bit in the response byte to indicate that it has data to transfer, and another bit to indicate a need for reprogramming. Consult the documentation for the device for interpretation of the response byte.

Example

Obtain the serial poll response byte from device `tape`.

```
ibrsp (tape,spr);
/* The application program would then analyze the response  */
/* in spr.   */
```

See Also

ibrsv(3) and *ibcmd(3)*
Chapter 2, *The C Language Library*

IBRSV(3) board only IBRSV(3)

Name

`ibrsv` - request service and/or set serial poll status byte

Synopsis

```
#include <sys/ugpib.h>
ibrsv (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` specifies the serial poll response byte.

If the 0x40 bit is set in `v`, the GPIB board additionally requests service from the Controller by asserting the GPIB SRQ line.

The `ibrsv` function is used to request service from the Controller using the SRQ signal and to provide a system-dependent status byte when the Controller serial polls the GPIB board.

It is not an error to call the `ibrsv` function when the GPIB board is the Controller-In-Charge (CIC), although doing so makes sense only if control will be passed later to another device. In this case, the call updates the status byte, but the SRQ signal is asserted only if the 0x40 bit is set and only when control is passed.

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

Examples

1. Set the serial poll status byte to 0x41, which simultaneously requests service from an external CIC.

   ```
   ibrsv(brd0,0x41);
   ```

2. Stop requesting service (unassert SRQ).

   ```
   ibrsv(brd0,0);
   ```

3. Change the status byte without requesting service.

   ```
   ibrsv(brd0,0x01);  /* new status byte value                    */
   ```

See Also

ibrsp(3)
Chapter 2, *The C Language Library*

IBSAD(3) device or board IBSAD(3)

Name

`ibsad` - change or disable secondary address

Synopsis

```
#include <sys/ugpib.h>
ibsad (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` is a valid secondary address.

If `v` is a number between 0x60 and 0x7E, that number becomes the secondary GPIB address of the device or interface board. If `v` is 0 or 0x7F, secondary addressing is disabled. `ibsad` is needed only to alter the value from its configuration setting.

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

The assignment made by this function remains in effect until `ibsad` is called again, the `ibonl` function is called, or the file is closed.

When `bd` specifies a device, the function enables or disables extended GPIB addressing for the device. When extended addressing is enabled, `ibsad` records the secondary GPIB address of that device to be used in subsequent device-level I/O function calls. The actual GPIB secondary address of any device is set within that device, either with hardware switches or a software program. Refer to your device documentation for instructions.

When `bd` specifies an interface board, the `ibsad` function enables or disables extended GPIB addressing and, when enabled, assigns the secondary address of the GPIB board.

Examples

1. Change the secondary GPIB address of device `dvm` from its current value to 0x6A.

   ```
   ibsad(dvm,0x6A);
   ```

2. Disable secondary addressing for device `dvm`.

   ```
   ibsad(dvm,0);
   ```

See Also

ibset(3), *ibpad(3)*, and *ibcmd(3)*
Chapter 2, *The C Language Library*

IBSET(3) device or board IBSET(3)

Name

IBSET - set device or board parameters
IBGET - get device or board parameters
IBINFO - get board/device counts

Synopsis

```
#include <sys/ugpib.h>
ioctl(bd,IBSET,&device)  or ioctl(bd,IBSET,&board)
ioctl(bd,IBGET,&device)  or ioctl(bd,IBGET,&board)
ioctl(bd,IBINFO,&ibinfo)
int bd;
struct device device;
struct board board;
struct ibinfo ibinfo;
```

Description

bd is a file descriptor returned from an ibfind call. IBSET, IBGET, and IBINFO are constants from ugpib.h. board, device, and ibinfo are structures defined in ugpib.h.

The IBGET and IBSET ioctl functions are used to get and set board or device information. Their use is analogous to the UNIX stty/gtty functions. IBGET can be called alone to retrieve all current settings.

Refer to include file ugpib.h for structure fields and their meanings.

The IBINFO ioctl function is used to determine the number of boards and devices in the system and whether a file descriptor specifies a board or a device. It uses the following structure:

```
struct ibinfo {
char    nbrds,   /* number of boards                        */
        ndevs,   /* number of devices                       */
        isdev;   /* non-zero if called with a device
                    file descriptor   */
};
```

See Also

ibbna(3), *ibdma(3)*, *ibeos(3)*, *ibeot(3)*, *ibist(3)*, *ibpad(3)*, *ibppc(3)*, *ibrsc(3)*, *ibsad(3)*, and *ibtmo(3)*
Chapter 2, *The C Language Library*

IBSGNL(3) board only IBSGNL(3)

Name

`ibsgnl` - request UNIX signal on specific events

Synopsis

```
#include <sys/ugpib.h>
ibsgnl (b,mask)
int b,mask;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `mask` is a bit mask with the same bit assignments as the status word, `ibsta`.

A mask bit is set to request a signal when the corresponding event occurs. A mask of zero disables signals. Table A-6 displays the recognized bits.

Table A-6. Signal Mask Layout

Mnemonic	Bit Position	Hex Value	Description
SRQI	12	1000	SRQ on
LOK	7	80	GPIB board is in Lockout State
REM	6	40	GPIB board is in Remote State
CIC	5	20	GPIB board is Controller-In-Charge
TACS	3	8	GPIB board is Talker
LACS	2	4	GPIB board is Listener
DTAS	1	2	GPIB board is in Device Trigger State
DCAS	0	1	GPIB board is in Device Clear State

`ibsgnl` is similar to the `ibwait` function except that it returns immediately, freeing the application program to perform other tasks. Except for SRQI, a signal will be sent on any transition into or out of the specified state (for example, from TACS to non-TACS).

The default signal is SIGINT.

You must arrange for the signal to be caught or your program will terminate when the signal is sent.

An `ibsgnl` call remains in effect until an `ibonl` of 0, an `ibsgnl` of 0, or the program terminates.

Example

Request a signal on service request.

```
ibsgnl(brd0,SRQI);
```

See Also

signal(2), *ibwait(3)*, and *ibset(3)*
Chapter 2, *The C Language Library*

Bugs

`ibsgnl` cannot be used with device-level calls. In particular `ibsgnl(b,SRQI)` will override automatic serial polling. A signal may be missed if the signal condition occurs during an `ibrd`, `ibwrt`, or `ibcmd` call.

IBSIC(3) board only IBSIC(3)

Name

`ibsic` - send Interface Clear (IFC)

Synopsis

```
#include <sys/ugpib.h>
ibsic (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` specifies how IFC is sent.

If `v` equals 1, the `ibsic` function causes the GPIB board to assert the IFC signal for at least 100 µsec, provided the GPIB board has System Controller authority. This action initializes the GPIB and makes the interface board Controller-In-Charge (CIC). It is generally used to become CIC or to clear a bus fault condition.

Some non-standard devices may require a pulse of IFC longer than 100 µsec. To help these devices, a value of `v==2` will leave IFC asserted and a value of `v==0` will unassert IFC. Any other value will pulse IFC.

The IFC signal is supposed to reset only the GPIB interface functions of bus devices and not intended to reset internal device functions. Device functions are reset with the Device Clear (DCL) and Selected Device Clear (SDC) commands. To determine the effect of these messages, consult the device documentation.

The ESAC error occurs if the GPIB board does not have System Controller capability.

Example

Initialize the GPIB and become CIC at the beginning of a program.

```
ibsic(brd0,1);
```

See Also

ibrsc(3)
Chapter 2, *The C Language Library*

IBSRE(3) board only IBSRE(3)

Name

`ibsre` - set or clear the Remote Enable (REN) line

Synopsis

```
#include <sys/ugpib.h>
ibsre (b,v)
int b,v;
```

Description

`b` is a file descriptor returned from an `ibfind` call. `v` specifies set or clear.

If `v` is non-zero, the Remote Enable (REN) signal is asserted. If `v` is zero, the signal is unasserted.

The `ibsre` function turns the REN signal on and off. REN is used by devices to select between local and remote modes of operation. REN enables the remote mode. A device does not actually enter remote mode until it receives its listen address.

The ESAC error occurs if the GPIB board is not System Controller.

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

Examples

1. Place a device at listen address 0x23 (ASCII #) in remote mode with local ability to return to local mode.

```
ibsre(brd0,1);          /* set REN to true                    */
ibcmd(brd0,"#",1);      /* LAD                                */
```

2. Exclude the local ability of the device to return to local mode by sending the Local Lockout command (0x11), or include it in the command string in Example 1.

```
ibcmd(brd0,"\x11");     /* LLO                                */
```

or

```
ibsre(brd0,1);          /* REN true                           */
ibcmd(brd0,"#\x11");    /* LAD LLO                            */
```

3. Return all devices to local mode.

```
ibsre(brd0,0);          /* set REN to false                   */
```

See Also

ibset(3), *ibrsc(3)*, and *ibsic(3)*
Chapter 2, *The C Language Library*

IBTMO(3) device or board IBTMO(3)

Name

`ibtmo` - change or disable time limit

synopsis

```
#include <sys/ugpib.h>
ibtmo (bd,v)
int bd,v;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `v` is a code specifying the time limit. Table A-7 lists the timeout settings.

Table A-7. Timeout Settings

Code	Actual Value	Minimum Timeout
`TNONE`	0	disabled*
`T10us`	1	10 µsec
`T30us`	2	30 µsec
`T100us`	3	100 µsec
`T300us`	4	300 µsec
`T1ms`	5	1 msec
`T3ms`	6	3 msec
`T10ms`	7	10 msec
`T30ms`	8	30 msec
`T100ms`	9	100 msec
`T300ms`	10	300 msec
`T1s`	11	1 sec
`T3s`	12	3 sec
`T10s`	13	10 sec
`T30s`	14	30 sec
`T100s`	15	100 sec
`T300s`	16	300 sec
`T1000s`	17	1000 sec
* If you select `TNONE`, no limit will be in effect and I/O operations could proceed indefinitely.		

`ibtmo` is needed only to alter the value from its configuration setting.

The time limit is an escape mechanism used to exit gracefully from a "hung bus" condition. Since the GPIB is an asynchronous bus, read and write operations can be held up indefinitely.

Timeout values are approximate, though never less than indicated.

Examples

1. Change the time limit for device level I/O operations involving device `tape` to approximately 300 msec.

```
ibtmo(tape,T300ms);
```

2. Perform I/O operations with no timeout in effect (not recommended).

```
ibtmo(brd0,0);
```

See Also

ibset(3)
Chapter 2, *The C Language Library*

IBTRG(3) device only IBTRG(3)

Name

`ibtrg` - Group Execute Trigger (GET)

Synopsis

```
#include <sys/ugpib.h>
ibtrg (d)
int d;
```

Description

`d` is a file descriptor returned from an `ibfind` call.

The `ibtrg` function addresses and triggers the specified device, then unaddresses all devices on the GPIB.

`ibtrg` sends the following commands and information:

- Listen address of the device
- Secondary address of the device, if applicable
- Group Execute Trigger (GET)
- Untalk (UNT) and Unlisten (UNL)

The response to a trigger is device-dependent.

Example

Trigger device `analyz`.

```
ibtrg(analyz);
```

See Also

ibcmd(3) in this appendix.
Chapter 2, *The C Language Library*

IBWAIT(3) device or board IBWAIT(3)

Name

`ibwait` - wait for selected events

Synopsis

```
#include <sys/ugpib.h>
ibwait (bd,mask)
int bd,mask;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `mask` is a bit mask with the same bit assignments as the status word, `ibsta`.

A `mask` bit is set to wait for the corresponding event to occur.

The `ibwait` function is used to monitor the events selected in `mask` and to delay processing until any of them occur. These events and bit assignments are shown in Table A-8.

Table A-8. Wait Mask Layout

Mnemonic	Bit Pos.	Hex Value	Description
TIMO	14	4000	Time limit exceeded
END	13	2000	GPIB board detected END or EOS
SRQI	12	1000	SRQ on
RQS	11	800	Device requesting service
LOK	7	80	GPIB board is in lockout state
REM	6	40	GPIB board is in remote state
CIC	5	20	GPIB board is CIC
TACS	3	8	GPIB board is Talker
LACS	2	4	GPIB board is Listener
DTAS	1	2	GPIB board is in device trigger state
DCAS	0	1	GPIB board is in device clear state

If `mask`=0, the function returns immediately. This is used to report the current device or board state.

If the TIMO bit is 0 or the time limit is set to 0, timeouts are disabled. Disabling timeouts should be done only when it is certain the selected event will occur.

When `bd` specifies a device, the only valid wait event is RQS, with or without TIMO. When waiting for RQS, a device frees its access board to perform other tasks. The function will return when a positive serial poll status byte is received from the device.

When `bd` specifies a board, all activity on that board is suspended until the event occurs.

Examples

1. Wait indefinitely for device `logger` to request service.

   ```
   ibwait(logger,RQS);
   ```

2. Wait for a service request or a timeout on `brd0`.

   ```
   ibwait(brd0,SRQI|TIMO);
   ```

3. Report the current status for `ibsta`.

   ```
   ibwait(bd,0);
   ```

4. Wait indefinitely until control is passed from another Controller-In-Charge (CIC).

   ```
   ibwait(bd,CIC);
   ```

5. Wait indefinitely until addressed to talk or listen by another CIC.

   ```
   ibwait(bd,TACS|LACS);
   ```

See Also

ibsgnl(3), *ibtmo(3)*, *ibgts(3)*, and *ibrsp(3)*
Chapter 2, *The C Language Library*

IBWRT(3) device or board IBWRT(3)

Name

`ibwrt` - write data to GPIB from a buffer

Synopsis

```
#include <sys/ugpib.h>
ibwrt (bd,buf,cnt)
int bd,cnt;
char buf[];
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `buf` contains the data to be sent over the GPIB. `cnt` specifies the number of bytes to be sent over the GPIB.

The `ibwrt` function writes `cnt` bytes of data to a GPIB device.

Device Call

When `bd` specifies a device, the following steps are performed:

1. The device is addressed to listen and the access board is addressed to talk, if they are not already so addressed.

2. The board writes the data to the device.

When the device-level `ibwrt` function returns, `ibsta` holds the latest device status, `ibcnt` is the actual number of data bytes written to the device, and `iberr` is the first error detected if the ERR bit in `ibsta` is set.

Board Call

When `bd` specifies an interface board, the `ibwrt` function attempts to write to a GPIB device that is assumed to be already properly initialized and addressed.

If the access board is Controller-In-Charge (CIC), the `ibcmd` function must be called prior to `ibwrt` to address the device to listen and the board to talk. Otherwise, the device on the GPIB that is the CIC must perform the addressing.

If the access board is Active Controller, the board is first placed in Standby Controller state with ATN off and remains there after the write operation has completed. Otherwise, the write operation commences immediately. An EADR error results if the board is CIC but has not been addressed to talk with the `ibcmd` function. An EABO error results if the board is not the CIC and is not addressed to talk within the time limit. An EABO error also results if the operation does not complete for whatever reason within the time limit.

The `ibwrt` operation terminates on any of the following events:

- All bytes are transferred.
- Error is detected.
- Time limit is exceeded.
- Device Clear (DCL) or Selected Device Clear (SDC) command is received from another device which is the Controller-In-Charge.

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

Examples

1. Write ten instruction bytes to device `dvm`.

```
ibwrt(dvm,"F3R1X5P2G0",10);
```

2. Write five instruction bytes terminated by a carriage return and a linefeed to device `ptr`.

```
ibwrt(ptr,"IP2X5\r\n",7);
```

3. Write ten instruction bytes to a device at listen address 0x35 (ASCII 5) and then unaddress it (the talk address of the access board is 0x40 or ASCII @).

```
ibcmd(brd0,"?@5",3);  /* UNL MTA LAD                          */
/* send instruction bytes                                     */
ibwrt(brd0,"F3R1X5P2G0",10);
/* unaddress all listeners and talkers                        */
ibcmd(brd0,"_?",2);  /* UNT UNL                              */
```

See Also

ibcmd(3) and *ibwrtf(3)*
Chapter 2, *The C Language Library*

IBWRTF(3) device or board IBWRTF(3)

Name

`ibwrtf` - write data to GPIB from a file

Synopsis

```
#include <sys/ugpib.h>
ibwrtf (bd,fname)
int bd;
char *fname;
```

Description

`bd` is a file descriptor returned from an `ibfind` call. `fname` is the null-terminated UNIX pathname of the file to be sent over the GPIB.

The `ibwrtf` function is called repeatedly until EOF is reached on the data file.

Example

Write file `dvmdata` to device `dvm`.

```
ibwrtf(dvm,"dvmdata");
```

See Also

ibwrt(3)
Chapter 2, *The C Language Library*

Appendix B
Multiline Interface Command Messages

The following tables are multiline interface messages (sent and received with ATN TRUE).

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US		3F	077	63	?	UNL

Message Definitions

DCL Device Clear
GET Group Execute Trigger
GTL Go To Local
LLO Local Lockout
MLA My Listen Address
MSA My Secondary Address
MTA My Talk Address
PPC Parallel Poll Configure
PPD Parallel Poll Disable

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

PPE	Parallel Poll Enable	SPE	Serial Poll Enable
PPU	Parallel Poll Unconfigure	TCT	Take Control
SDC	Selected Device Clear	UNL	Unlisten
SPD	Serial Poll Disable	UNT	Untalk

Appendix C
GPIB Programming Example

This appendix illustrates the programming steps that could be used to program a representative IEEE-488 instrument from a terminal using the driver functions. The application program is written in C. The target instrument is a digital voltmeter (DVM). This instrument is otherwise unspecified. The purpose here is to explain how to use the driver software to execute certain programming and control sequences, not how to determine those sequences.

Because the instructions that are sent to program a device as well as the data that might be returned from the device are called *device-dependent messages*, the format and syntax of the messages used in this example are unique to this device. Furthermore, the *interface messages* or *bus commands* that must be sent to devices will also vary, but to a lesser degree. The exact sequence of messages to program and to control a particular device are contained in its documentation.

For example, the following sequence of actions is assumed to be necessary to program this DVM to make and return measurements of a high-frequency AC voltage signal in the autoranging mode:

1. Initialize the GPIB interface circuits of the DVM so that it can respond to messages.
2. Place the DVM in remote programming mode and turn off the front panel control.
3. Initialize the internal measurement circuits.
4. Program the DVM to perform the proper function (F3 for high-frequency AC volts), range (R7 for autoranging), and trigger source (T3 for external or remote).
5. For each measurement:
   a. Send the Group Execute Trigger (GET) command to trigger the DVM.
   b. Wait until the DVM asserts Service Request (SRQ) to indicate that the measurement is ready to be read.
   c. Serial poll the DVM to determine if the measured data is valid (status byte = 0xC0) or if a fault condition exists (the 0x40 bit and another bit of the status byte, other than 0x80, are set).
   d. If the data is valid, read 16 bytes from the DVM.
6. End the session.

The example program given here also assumes that the GPIB board is the designated System Active Controller of the GPIB and that there is no change to the GPIB board default hardware and software parameters.

Example Program

```
#include <sys/ugpib.h>

char   cmd[512];      /*  command buffer          */
char   rd[512];       /*  read buffer             */
char   wrt[512];      /*  write buffer            */

unsigned int mask;    /*  events to be waited for */

main()     {
       int dvm;
       struct device device;

       /* find device "dvm" */
       dvm = ibfind ("dvm");
       if  (dvm  <  0)
           error("cannot find dvm");

       /* set device parameters */
       ioctl (dvm,  IBGET, &device);
       device.d_tmo = T10s;      /*  10 second timeout            */
       device.d_pad = 3;         /*  primary address octal 3      */
       device.d_sad = 0;         /*  no secondary address         */
       device.d_uflags = EOT;    /*  send END with last byte,
                                     no EOS modes                 */
       ioctl(dvm, IBSET, &device);

       /* Send the Selected Device Clear (SDC) message to clear
          internal device functions.                              */
       if (ibclr(dvm) & ERR) err();

       /* Send the Local Lockout (LLO) message.                   */
       if (ibllo(dvm) & ERR) err();

       /* Write the function, range, and trigger source
          instructions to the DVM.                               */
       if (ibwrt(dvm,"F3R7T3",6) & ERR) err();

       /* Send the Group Execute Trigger (GET) message to
          trigger a measurement reading.                          */
       if (ibtrg(dvm) & ERR) err();

       /* Wait for the DVM to set SRQ or for a timeout.           */
       if (ibwait(dvm,TIMO|RQS) & (ERR|TIMO)) err();

       /* Read serial poll response; if not equal to 0xC0,
          report dvm error.                                       */
       if (ibrsp(dvm,rd) & ERR) err();
       if ( (rd[0] & 0xFF) != 0xC0)  dvmerr();

       /* read the measurement.                                   */
       if (ibrd(dvm,rd,16) & ERR) err();
```

```
        /* Disable the device dvm.                                    */
        ibonl(dvm,0);

}

err()      {
/* An error checking routine at this location would, among other
   things, check iberr to determine the exact cause of the error
   condition and then take action appropriate to the application.
   For errors during data transfers, ibcnt can be examined to
   determine the actual number of bytes transferred.                  */
  }

dvmerr() {
/* A routine at this location would analyze the fault code
   returned in the DVM's status byte and take appropriate
   action.                                                            */
  }
```

User Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **NI-488M™ Software Reference Manual**

Edition Date **July 1994**

Part Number: **320062-01**

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678