# GPIB-488

## GPIB-488 Programming Reference Manual

# Conventions

The following conventions are used in this manual:

[ ]                      Square brackets indicate the key to be pressed.

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

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

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

When symbol is marked on a product, it denotes a warning advising you to take precautions to avoid electrical shock.

When symbol is marked on a product, it denotes a component that may be hot. Touching this component may result in bodily injury.

**bold**           Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

*italic*          Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

`monospace`      Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

GPIB terms used within this manual are:

GPIB            General Purpose Interface Bus

System controller    The system controller has the unique ability to retrieve active control of the bus or to enable devices to be remotely programmed. It takes control of the bus by issuing an IFC (Interface Clear) message for at least 200 μsec. It also can put devices into the remote state by asserting the REN (Remote Enable) line.

There is always one system controller in a GPIB system. The system

controller is designated at system initialization either through the use of hardware switches or by some type of configuration software, and is not changed. The system controller can be the same controller as the one which is the current active controller or an entirely different one. Note that if a controller is both a system controller and the active controller and it passes control to another controller, the system controller capability is not passed along with it.

Active controller    The active controller is the controller which has the ability to mediate all communications which occur over the bus. In other words, the active controller designates (addresses) which device is to talk and which devices are to listen. The active controller is also capable of relinquishing its position as active controller and designating another controller to become the active controller.

Device    A device is any IEEE-488 instrument which is not a system controller or active controller. It can be idle or act as a talker and/or listener when addressed or unaddressed by the active controller.

Listener    A listener is any device which is able to receive data when properly addressed. There can be up to 14 active listeners on the bus concurrently. Some devices can also be a talker or controller; however, only one of these functions can be performed at a time.

Talker    A talker is a device which can transmit data over the bus when properly addressed. Only one device can transmit at a time. Some devices can also be a listener or controller; however, only one of these functions can be performed at a time.

# Contents

# Chapter 4
# GPIB 488.2 Library Reference

# Appendix A
# Multiline Interface Messages

# Appendix B
# IBSTA

# Appendix C
# IBERR

**1**

# GPIB Software Overview

The GPIB software includes the 488.1 library, the 488.2 library, and a set of utility programs. The 488.1 library consists of all of the functions and subroutines that begin with the letters "ib". The 488.1 library routines refer to devices on the GPIB bus by their device names and handles rather than by their GPIB addresses.

•   The 488.2 library consists of all the routines that do not begin with the letters "ib". The 488.2 library routines refer to devices on the GPIB bus by their GPIB addresses rather than by their names or handles.

✎   **Note**   The GPIB library is available only in a 32-bit version.

## Supported Languages

The GPIB library provides identical routines for each supported language. Languages supported by the GPIB library at the time this manual was published are listed below.

**Table 1-1.**  Programming Languages

| Programming Languages |
|:---:|
| Delphi |
| C |
| VB |

# GPIB Library Utility Programs

The following utility programs are installed with the GPIB library software.

| Utility program | Description |
|---|---|
| GPIBDIAGNOSTIC.EXE | Hardware test program (Win32) |
| GPIBINTERACTIVECONTROL.EXE | Interactive control program (Win32) |
| GPIBCONFIGURATION.EXE | Configuration utility program (Win 32) |

## Support For VISA Calls

VISA (Virtual Instrument Software Architecture) drivers are command drivers that convert company and program-independent VISA calls into company-dependent calls.

## GPIB-32.DLL function support

Each library function defined by GPIB 488.1 and GPIB 488.2 has a corresponding entry point in gpib-32.DLL.

## ibnotify

The GPIB library does not support ibnotify. Applications that utilize the ibnotify function will not run properly.

# 2

# Programming with the GPIB Library

The GPIB library contains two different and complete GPIB libraries.

**Original 488.1 library**—The 488.1 library (also referred to as the original library), consists of all of the functions and subroutines that begin with the letters "ib". This library uses a concept of device names and handles rather than GPIB addresses when referring to GPIB devices. There are two advantages to this approach:

*   The GPIB addresses of each device are not stored in the program, so the same program can run on different buses where the addresses of each device are different.

*   The program can refer to each device with an intelligible name rather than a number (the GPIB address).

**488.2 library**—This library consists of all the routines that do not begin with the letters "ib". These routines refer to all devices on the bus by their GPIB addresses rather than by names. The *Device I/O* section does not apply to the 488.2 library.

The GPIB library includes different routines that allow you to control the operations of the GPIB bus in a very specific manner. You may find the number of routines included in the GPIB library intimidating, however, in most applications you need to use only a small subset of these routines.

The routines are divided into two distinct libraries. All routines which begin with "ib" are part of the "488.1" or "Original GPIB library." All other routines are part of the "488.2 library." You only need to use one or the other library. Each library provides a different method of performing the same tasks. The choice of which library to use is a matter of personal preference. If you use the original GPIB library, you can perform either Board Level or Device Level operations.

# General Concepts

This section explains the difference between routines which use Device I/O and those which use Board I/O, the use of device handles, and the global variables used by the library routines.

## Device vs. Board I/O

The most typical GPIB operations are sending commands to a device attached to the bus and reading back responses. To do this, program the GPIB board to execute these steps:

1.  Address the selected device as a Listener.

2.  Send the secondary address if used.

3.  Address the board itself as the GPIB Talker.

4.  Send the command bytes to the device.

5.  Address the board itself as the Listener.

6.  Read the response from the device.

7.  Send the GPIB Unlisten (UNL) message.

8.  Send the GPIB Untalk (UNT) message.

The original GPIB library interface is comprised of two different types of routines: Board I/O and Device I/O. These routines are described in Chapter 3, *GPIB 488.1 Library Reference*. You can program the board using either Board I/O routines or Device I/O routines to perform the sequence of operations outlined above.

The 488.2 library is all "Board I/O" in that you always supply the board ID and the device address. Refer to Chapter 4, *GPIB 488.2 Library Reference*.

## Device I/O

It is usually easier to use the Device I/O routines. Device I/O is very simple to use and understand. Device I/O routines are higher-level routines which conceal most of the underlying complexity of GPIB operations. The Device I/O routines automatically take care of all of the low-level details involving GPIB messages and addressing. For example, to accomplish the seven steps listed above, you use only three routines:

- `ibdev`—to open the device

- `ibwrt`—to send the instrument command

- `ibrd`—to read the data back from the device

## Board I/O

In comparison, the Board I/O routines are low-level routines. If you use them, you must understand how the GPIB operates in detail. Generally, the only time you need to use Board I/O is if it is impossible to perform the same operation using device I/O, such as passing control from one controller to another.

To perform the same task as the seven steps outlined in *Device vs. Board I/O* (send a command to a device), you need to know the codes for the various forms of addressing and the codes for the GPIB Unlisten and Untalk commands.

Use the routines in this sequence:

ibfind—to open the board

ibcmd—to send the address of the talker and listener

ibwrt—to send the command to the device

ibcmd—to send the address of the talker and listener

ibrd—to read the data back from the device

ibcmd—to send the Unlisten (UNL) and Untalk (UNT) commands

## Device Handles

Most of the routines in the 488.1 library have a device handle as the first argument. The first GPIB call in your program is usually ibfind. This routine "opens" a board or device and returns a GPIB board or device handle. If you pass the name of a board, it returns a board handle. Likewise, if a device name is passed, a device handle is returned. Some library routines only work with device handles, some only with board handles, and some with both.

# Global Variables

The following global variables are used in all programming languages:

| | |
|---|---|
| `ibsta` | Status Word |
| `iberr` | Error Codes |
| `ibcnt`, `ibcntl` | Count Variables (short/long) |

The `iberr` variables are briefly explained here. For additional information about `iberr`, refer to Appendix C, *IBERR*.

For additional information about `ibcnt` and `ibcntl`, refer to the routines which return them.

## iberr—The Error Variable

If a GPIB error occurs during a routine, its corresponding error code is returned into the variable `iberr`. Possible error codes and their meanings are listed in Appendix C, *IBERR*.

## ibcnt and ibcntl—Count Variables

These variables contain an integer which describes how many bytes were actually transferred during a read or write operation. `ibcnt` is an integer value (16-bits wide) and `ibcntl` is a long integer value (32-bits wide).

# 3

# GPIB 488.1 Library Reference

This chapter describes each of the 488.1 GPIB library routines. A short description of the routine, its syntax, parameters, any values that are returned, any special usage notes, and an example are included for each routine. The routines are listed in alphabetical order. The following table lists all of the 488.1 GPIB library routines. A full description of each routine follows the table.

**Table 3-1.** 488.1 Library routines

| Name | Description |
|---|---|
| ibask | Returns software configuration information |
| ibcac | Become Active Controller |
| ibclr | Clear specified device |
| ibcmd | Send GPIB commands from a string |
| ibcmda | Send GPIB commands asynchronously from a string |
| ibconfig | Configure the driver |
| ibdev | Open and initialize a device when the device name is unknown |
| ibdma | Enable/Disable DMA |
| ibeos | Change EOS |
| ibeot | Change EOI |
| ibfind | Open a device and return its unit descriptor |
| ibgts | Go from Active Controller to standby |
| ibist | Define IST bit |
| iblines | Return status of GPIB bus lines |
| ibln | Check for presence of device on bus |
| ibloc | Go to Local |
| ibonl | Place device online/offline |

**Table 3-1.** 488.1 Library routines (Continued)

| Name | Description |
|------|-------------|
| ibpad | Change Primary address |
| ibpct | Pass Control |
| ibppc | Parallel Poll Configure |
| ibrd | Read data to a string |
| ibrda | Read data asynchronously |
| ibrdf | Read data to file |
| ibrpp | Conduct parallel poll |
| ibrsc | Request/release system control |
| ibrsp | Return serial poll byte |
| ibrsv | Request service |
| ibsad | Define secondary address |
| ibsic | Send IFC |
| ibsre | Set/clear REN line |
| ibstop | Stop asynchronous I/O operation |
| ibtmo | Define time limit |
| ibtrg | Trigger selected device |
| ibwait | Wait for event |
| ibwrt | Write data from a string |
| ibwrta | Write data asynchronously from a string |
| ibwrtf | Write data from file |

# IBASK

Returns software configuration information.

## Syntax

```
C                   ibask (int boarddev, int option, unsigned int
                    *value)
```

## Parameters

| | |
|---|---|
| boarddev | A board handle or device handle |
| option | Specifies which configuration item to return; see Table. |
| value | Current value of specified item returned here |

**Table 3-2.** ibask Options

| Option | Valid for | Information returned |
|---|---|---|
| IbaPAD | bd/dev | Primary address of board or device |
| IbaSAD | bd/dev | Secondary address of board or dev |
| IbaTMO | bd/dev | The current timeout value for I/O commands (refer to ibtmo for a list of possible values) |
| IbaEOT | bd/dev | 0 = EOI asserted at end of write<br><br>non zero = EOI is not asserted at end of write |
| IbaPPC | bd | The current parallel poll configuration of the board |
| IbaREADDR | dev | 0 = Forced re-addressing is disabled<br><br>non zero = Forced re-addressing is enabled. |
| IbaAUTOPOLL | bd | 0 = automatic at end of write<br><br>non zero = automatic serial poll is disabled |
| IbaCICPROT | bd | 0 = CIC protocol is disabled<br><br>non zero = CIC protocol is enabled |
| IbaSC | bd | 0 = board is not system controller<br><br>non zero = board is system controller |
| IbaSRE | bd | 0 = do not automatically assert REN line when system controller<br><br>non zero = automatically assert REN line when system controller |

**Table 3-2.** `ibask` Options (Continued)

| Option | Valid for | Information returned |
|---|---|---|
| `IbaEOSrd` | `bd/dev` | 0 = ignore EOS char during reads<br><br>non zero = terminate read when EOS char is received |
| `IbaEOSwrt` | `bd/dev` | 0 = don't assert EOI line when EOS char is sent<br><br>non zero = assert EOI line whenever EOS char is sent |
| `IbaEOScmp` | `bd/dev` | 0 = 7 bit compare is used when checking for EOS char<br><br>non zero = 8 bit compare is used when checking for EOS char |
| `IbaEOSchar` | `bd/dev` | 0 = The current EOS char of board or device |
| `IbaPP2` | `bd` | 0 = board is in remote parallel poll configuration<br><br>non zero = board is in local parallel poll configuration |
| `IbaTiming` | `bd` | Current T1 timing delay 1 = Normal (2 us), 2 = High Speed (500 ns), 3 = Very High Speed (350 ns) |
| `IbaDMA` | `bd` | 0 = The interface does not use DMA for GPIB transfers<br><br>non zero = The interface uses DMA for GPIB transfers |
| `IbaSendLLO` | `bd` | 0 = LLO command is not sent when device is put online<br><br>non zero = LLO command is sent |
| `IbaSPollTime` | `dev` | Length of time to wait for parallel poll response before timing out |
| `IbaPPollTime` | `bd` | Length of time to wait for parallel port response |
| `IbaEndBitIsNormal` | `bd` | 0 = The END bit of ibsta is not set when the EOS character is received without EOI.<br><br>non zero = The END bit of ibsta is set when the EOS character is received |
| `IbaUnAddr` | `dev` | 0 = The untalk and unlisten (UNT, UNL) are not sent after each device level read/write<br><br>non zero = The UNT, UNL commands are sent after each device lever read/write |
| `IbaIST` | `bd` | The individual status (ist) bit of the interface |
| `IbaBNA` | `dev` | Device's access board |

**Returns**              `ibsta` will contain a 16-bit status word; refer to Appendix B,
                         *IBSTA*.
                         `iberr` will contain an error code if an error occurred
                         `value` will contain the current value of selected configuration
                         item

**Usage notes**          Some options apply to boards, some to devices and some apply to
                         both boards and devices.
                         A program may modify many of these configuration items via
                         library routines (for example, `ibtmo`, `ibeos`, etc.). In that case,
                         `ibask` returns the modified version.

**Example**              Returns the `ist` bit of a device at PAD 3.


C                        ```
                         int dev, istbit;
                         dev = ibdev (0,3,0,13,1,0);
                         ibask (dev, IbaIST, &istbit);
                         ```

# IBCAC

Makes the specified board the Active Controller.

## Syntax

```
C                  ibcac (int board, int sync)
```

## Parameters

`board` is an integer containing the board handle

`sync` specifies if the GPIB board is to take control synchronously or asynchronously. If sync is 0, the GPIB board takes control asynchronously. Otherwise, it takes control synchronously (immediately).

When the board takes control, the GPIB interface board asserts the ATN line. When taking control synchronously, the board waits for any data transfer to be completed and then takes control. Note that if synchronous take control is specified while an `ibrd` or `ibwrt` operation is in progress, the synchronous take control may not occur if a timeout or other error occurs during the `ibrd`/`ibwrt`.

In comparison, if the board is to take control asynchronously, it takes control immediately, without waiting for any data transfers to be completed.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. In particular, the ECIC error occurs if the specified GPIB board cannot become an Active Controller.

## Usage Notes

This routine is only used when doing board level I/O.

## Example

GPIB board 1 takes control asynchronously.

```
C                  ibcac (brd1, 0);
```

# IBCLR

Clears a specified device.

## Syntax

```
C                       ibclr (int device)
```

## Parameters

`device` is an integer containing the device handle.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the GPIB Interface Board (which is currently the CIC) sends its talk address over the GPIB. This makes it the active talker. It then unlistens all devices and addresses the specified device as a listener. Finally, the GPIB board clears the device, by sending the `Selected Device Clear` message.

## Example

This example uses `ibdev` to return the unit descriptor for a device at PAD 5, a DMM, into the variable dmm. The DMM is then cleared.

```
C                   int dmm;
                    dmm = ibdev(0,5,0,13,1,0);
                        /*open instrument*/
                    ibclr (dmm);
                        /* clear it */
```

# IBCMD

Sends GPIB commands.

## Syntax

```
C                       ibcmd (int board, char cmnd[], long bytecount)
```

## Parameters

`board` is an integer containing the board handle.

`cmnd` is the command string to be sent. This string is comprised of GPIB multiline commands. These commands are listed in Appendix A, *Multiline Interface Messages*.

`bytecount` is the number of command bytes to be transferred.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

`ibcnt`, `ibcntl` will contain the number of bytes that were transferred. `ibcnt` is a 16-bit integer. `ibcntl` is a 32-bit integer. If the requested count was greater than 64 K, use `ibcntl` instead of `ibcnt`.

## Usage Notes

This routine passes only GPIB commands. It cannot be used to transmit programming instructions (data) to devices. Use the `ibrd` and `ibwrt` routines for this purpose.

This routine terminates when any one of the following takes place:

- Commands transfer is successfully completed.
- An error occurs
- A timeout occurs
- A Take Control (TCT) command is sent
- The system controller asserts the IFC (Interface Clear) line.

## Example

This example prepares the board to talk and addresses three devices (at addresses 8, 9, and 10) to listen.

```
C                 char *command;
                  command = "\0x3f\0x5f\0x40\0x28\0x29\0x2a"
                  ibcmd (board, command, 6);
```

# IBCMDA

Transfers GPIB commands asynchronously from a string.

## Syntax

```
C                    ibcmda (int board, char cmnd[], long bytecount)
```

## Parameters

board is an integer containing the board handle.

cmnd is the command string to be sent. This string is comprised of GPIB multiline commands. These commands are listed in Appendix A, *Multiline Interface Messages*.

bytecount is the number of command bytes to be transferred. Note that in C, although this parameter is of type long, integer values and variables can also be passed.

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred. An ECIC error is generated if the GPIB Interface Board specified is not the Active Controller. If no listening devices are found, the ENOL error is returned.

ibcnt, ibcntl will contain the number of bytes that were transferred. ibcnt is a 16-bit integer. ibcntl is a 32-bit integer. If the requested count was greater than 64 K, use ibcntl instead of ibcnt.

## Usage Notes

This routine passes only commands. It is not used to transmit programming instructions (data) to devices. Use the ibrd/ibwrt routines for this purpose.

Asynchronous I/O is not explicitly supported and will be treated as synchronous.

## Example

This example prepares the board to talk and addresses three devices (at addresses 8, 9, and 10) to listen. ibcmda executes in the background and the program continues into the WHILE loop. This loop calls ibwait to update ibsta and checks ibsta to see if ibcmda has completed or an error has occurred. The program may do anything within the WHILE loop except make other GPIB I/O calls.

```
C                    char command[] = "\0x3f\0x5f\0x40\0x28\0x29\0x2a"
                     ibcmda (board, command, 6);
                     while ( (ibsta & CMPL+ERR) == 0)
                         ibwait (board, 0);
```

# IBCONFIG

Changes configuration parameters.

## Syntax

```
C                 ibconfig (int boarddev, unsigned int option,
                  unsigned int value)
```

## Parameters

`boarddev` is an integer containing either a board handle or device handle.

`option` is a number which represents the configuration option to be changed. See Table 3-3.

`value` is the new configuration option value. Allowed values differ according to the option being programmed.

**Table 3-3.** `ibconfig` Options

| Option | Valid for | Description |
|---|---|---|
| IbcPAD | bd or dev | New Primary Address. Available primary addresses range from 0 to 30. `value` can be from 0 to 30 decimal. (See `ibpad`.) |
| IbcSAD | bd or dev | New Secondary Address. There are 31 secondary addresses available. `value` can be 0 or from 96 to 126 decimal. (See `ibsad`.) |
| IbcTMO | bd or dev | Timeout Value. The approximate time that I/O functions take before a timeout occurs. `value` is a number from 0 to 15 which corresponds to timeout values ranging from 10 usec to 100 sec. (See `ibtmo`) |
| IbcEOT | bd or dev | Enable/disable END message. If this option is enabled, the EOI line is asserted when the last byte of data is sent. If `value` = 0, then the EOI line is not asserted. If `value` is non zero, the EOI line is asserted. |
| IbcPPC | bd | Parallel Poll Configure. Redefines the parallel poll configuration bytes. `value` can be 0, or from 96 to 126 decimal. |
| IbcREADDR | dev | Forced re-addressing. If `value` = 0, forced re-addressing is disabled<br><br>non zero = Forced re-addressing is enabled. |

**Table 3-3.** `ibconfig` Options (Continued)

| Option | Valid for | Description |
|--------|-----------|-------------|
| IbcAUTOPOLL | bd | Enable/Disable Automatic Serial Polling. If `value` is 0, then Automatic Serial Polling is disabled. Otherwise, it is enabled. |
| IbcCICPROT | bd | CIC Protocol. If value is 0, then CIC Protocol is not used. Otherwise, CIC Protocol is used. |
| IbcIRQ | bd | Enable/Disable Hardware Interrupts. If `value` is 0, then hardware interrupts are disabled, otherwise `value` specifies the IRQ level the board uses to generate interrupts. |
| IbcSC | bd | Request/Release System Control. If `value` is 0, the board is not able to support routines requiring system controller capability. |
| | | If `value` is non-zero, the board can support routines requiring system controller capability. |
| IbcSRE | bd | Assert/Unassert REN. If `value` is 0, the REN line is unasserted. Otherwise, the REN line is asserted. |
| IbcEOSrd | bd or dev | Recognize EOS. If `value` is non-zero, a read is terminated when the End-Of-String (EOS) character is detected. Otherwise, EOS detection is disabled. |
| IbcEOSwrt | bd or dev | Assert EOI. If `value` is non-zero, then EOI is asserted when the EOS character is sent. Otherwise, EOI is not asserted. |
| IbcEOScmp | bd or dev | 7/8-bit Comparison. If `value` is zero, compare the low-order 7 bits of the EOS character. Otherwise, compare 8-bits. |
| IbcEOSchar | bd or dev | End-Of-String (EOS) Character. `value` is the new EOS character. `value` can be any 8-bit value. |
| IbcPP2 | | Parallel Poll Remote/Local. If `value` is zero, then the GPIB Interface Board is remotely configured for a parallel poll by an external Controller. Otherwise, the GPIB interface board accepts parallel poll configuration commands from your application program. |

**Table 3-3.** `ibconfig` Options (Continued)

| Option | Valid for | Description |
|--------|-----------|-------------|
| `IbcTIMING` | `bd` | Handshake Timing. If `value` is 1, normal timing (> 2 *sec.) is used. If `value` is 2, high-speed timing (> 500 nsec.) is used. If `value` is 3, very high-speed timing (> 350 nsec.) is used. |
| `IbcDMA` | `bd` | Enable/Disable DMA. If `value` is zero, DMA transfers are disabled, otherwise `value` specifies the DMA channel that the board uses. |
| `IbcSendLLO` | `bd` | Send Local Lockout. If `value` is 0, LLO command is not sent when device is put online; non zero = LLO command is sent |
| `IbcSPollTime` | `bd or dev` | Serial Poll Timeout. `value` ranges from 0 to 17 specify timeouts of 10 msecs to 1000 secs. Refer to Table 3-6, *Timeout Codes*. |
| `IbcEndBitIsNormal` | `bd or dev` | If set, causes END status to be set on receipt of EOS. |
| `IbcPPollTime` | `bd` | Parallel Poll Timeout. `value` ranges from 0 to 17 specify timeouts of 10 msecs to 1000 secs. Refer to Table 3-6, *Timeout Codes*. |
| `IbcUnAddr` | `dev` | If `value` is 0, the untalk and unlisten (UNT, UNL) are not sent after each device level read/write; non zero = the UNT, UNL commands are sent after each device lever read/write |

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

None.

## Example

This example illustrates how to change the timeout value for GPIB Interface Board 1 to 300 msec.

```
C                 int device;
                  device = ibfind ("gpib1");
                  ibconfig (device, IbcTMO, 10);
```

## IBDEV

Obtains a device handle for a device whose name is unknown. It opens and initializes the device with the configuration given.

### Syntax

```
C               device = ibdev (int boardindex, int pad, int sad,
                int timeout, int eot, int eos)
```

### Parameters

`boardindex` identifies the GPIB Interface Board with which the device descriptor is associated. It is an index in the range 0 to (total number of boards - 1).

`pad` is the primary address of the device. Available addresses range from 0 to 30.

`sad` is the secondary address of the device. There are 31 secondary addresses available. value can be 0, or from 96 to 126 decimal; see Appendix A, *Multiline Interface Messages*. If 0 is selected, the driver will not expect the device to require a secondary address.

`timeout` is the timeout of the device. This is a value from 0 to 17 which corresponds to timeout value ranging from 10 usec to 1000 sec. See Table 3-6, *Timeout Codes*, for a list of timeouts and corresponding values.

`eot` when writing to a device, eot specifies whether or not to assert EOI with the last data byte. If eot is non-zero, EOI is asserted. If eot is 0, EOI is not asserted.

`eos` specifies the End-Of-String termination mode to be used when communicating with the device. See Table 3-4, *Selecting EOS*, for a description of special formatting features of this argument.

### Returns

`device` will contain the assigned descriptor or a negative number. If device is a negative number, then an error occurred. Two types of errors can occur:

• An `EDVR` or `ENEB` error is returned if a device is not available or the board index specifies a non-existent board.

• An `EARG` error is returned if illegal values are given for `pad`, `sad`, `timeout`, `eot`, `eos`.

`iberr` will contain an error code, if an error occurred.

### Usage Notes

This routine returns the device handle of the first available user-configurable device it finds in the device list.

## Example

This example opens an available device, associates it with GPIB interface board 1, and assigns it the following device configuration parameters.

- primary address = 3

- secondary address = 19 (115 decimal, 73 hex)

- timeout = 10 sec

- Assert EOI

- EOS Disabled

- The new device handle is returned.

```
C                      int device;
                       device = ibdev(1, 3, 0x73, 13, 1, 0);
```

# IBDMA

Enables/Disables DMA.

## Syntax

C                          ibdma (int board, int dma)

## Parameters

board is an integer containing the board handle.

dma is an integer which indicates whether DMA is to be enabled or disabled for the specified GPIB board. If dma is non-zero, all read and write operations between the GPIB board and memory are performed using DMA. Otherwise, programmed I/O is used.

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred. An ECAP error results if you tried to enable DMA operations for a board which does not support DMA operation. If no error occured, the previous value of dma is stored in iberr.

## Usage Notes

The GPIB Interface Board must have been configured for DMA operations in order for this routine to be executed successfully. This routine is useful for alternating between programmed I/O and DMA operations. This call remains in effect until one of the following occurs:

• Another ibdma call is made.

• ibonl or ibfind is called.

• The program is re-started.

• The maximum DMA transfer length in Windows is 64 K bytes.

## Example

This example enables DMA transfers for GPIB Interface Board 1. It assumes that the DMA channel was previously selected in your configuration program.

```
C                  int board, ibsta;
                   board = ibfind ("gpib1");
                   ibsta = ibdma (board, 1);
```

# IBEOS

Changes or disables End-Of-String termination mode.

## Syntax

```
C                    ibeos (int boarddev, int eos)
```

## Parameters

`boarddev` is an integer containing the board/device handle.

`eos` is an integer that defines which termination mode and what EOS character are to be used, as shown in Table 3-4, *Selecting EOS*.

**Table 3-4.**  Selecting EOS

| Method | Description | eos | |
| | | High Byte | Low Byte |
|---|---|---|---|
| A | Terminate read when EOS is detected. Can be used alone or in combination with Method C. (Constant = REOS) | 00000100 | EOS character |
| B | Set EOI with EOS on write function. Can be used alone or in combination with Method C. (Constant = XEOS) | 00001000 | EOS character |
| C | Compare all 8 bits of EOS byte rather than low 7 bits for all read and write functions. (Constant = BIN) | 00010000 | EOS character |

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. If an error does not occur the previous value of `eos` is stored in `iberr`.

## Usage Notes

This call only defines an EOS byte for a board or device. It does not cause the handler to automatically send that byte when performing writes. Your application must include the EOS byte in the data string it defines.

If this call defines an EOS for a *device*, then the defined EOS is used for all reads and writes involving that device. Likewise, if the call defines an EOS for a *board*, then all reads and writes involving that board will use that EOS.

This call remains in effect until one of the following occurs:

- Another ibeos call is made.

- ibonl or ibfind is called.

- The system is re-started.

See also ibeot

## Example

This example configures the GPIB system to send the END message whenever the line feed character is sent to a particular device. Method B described in Table 3-4, *Selecting EOS*, is used (XEOS).

```
C                   int device;
                    ibeos (device, XEOS + '\n');
```

# IBEOT

Enables/Disables assertion of EOI on write operations.

## Syntax

```
C                       ibeot (int boarddev, int eot)
```

## Parameters

boarddev is an integer containing the board or device handle. Here it represents a GPIB Interface Board or a device. This value is obtained by calling the ibfind routine.

eot is an integer which defines whether or not EOI is to be asserted. If eot is non-zero then EOI is asserted automatically when the last byte of the message is sent. If eot is 0, then EOI is not asserted.

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred.

## Usage Notes

This call is used to temporarily change the default EOT setting.

It is useful to automatically send EOI with the last data byte in situations where variable length data is being sent. When EOI is enabled, you do not need to send an EOS character.

If this call specifies a device, then EOI is asserted/unasserted on all writes to that device. Likewise, if the call specifies a board, then EOI is asserted/unasserted on all writes from that board. To assert EOI with an EOS, use the ibeos routine. This call remains in effect until one of the following occurs:

*   Another ibeot call is made.

*   ibonl or ibfind is called.

*   The system is re-started.

See also ibeos

## Example

Assert EOI with last byte of all write operations from GPIB board 1.

```
C                       int device;
                        device = ibfind ("gpib1");
                        ibeot (device,1);
```

# IBFIND

Opens a board or device and returns the handle associated with a given name.

## Syntax

C                            boarddev = ibfind (char name[])

## Parameters

name is the string specifying the board or device name.

## Returns

boarddev will contain the device handle associated with the given name. If a negative number is returned, this indicates that the call has failed. This most often happens when the specified name is does not match the default/configured board or device name.

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred.

## Usage Notes

This call is also opens the device/board and initializes the software parameters to their default configuration settings. See ibonl.

Using ibfind to obtain device descriptors is useful only for compatibility with existing applications. New applications should use ibdev instead of ibfind.

## Example

This example returns the device handle associated with the device named "DEV5" to the variable dmm. If the device name is not found, the program will jump to an error routine.

C                       Board = ibfind("GPIB0");
                        if (Board < 0) error();

# IBGTS

Puts an Active Controller in Standby mode.

## Syntax

```
C                     ibgts (int board, int handshake)
```

## Parameters

`board` is an integer containing the board handle.

`handshake` determines whether or not the shadow handshake option is to be activated. If `handshake` is non-zero, then the GPIB shadow handshake option is activated. This means that the GPIB board shadow handshakes the data transfer as an acceptor and when the END message is detected, the GPIB board enters a `Not Ready For Data` (NRFD) handshake hold-off state on the GPIB. Thus, the GPIB board participates in the data handshake as an Acceptor without actually reading the data. It monitors the transfers for the END message and holds off subsequent transfers. Using this mechanism, the GPIB board can take control synchronously on a subsequent operation like `ibcmd` or `ibrpp`.

If `handshake` is 0, then no shadow handshake or holdoff is done.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. The ECIC error occurs if the board is not an Active Controller.

## Usage Notes

This call makes the GPIB board go to Controller Standby state and unasserts the ATN line if it is initially the Active Controller. This allows transfers between GPIB devices to occur without the GPIB board's intervention.

Before performing an `ibgts` with a shadow handshake, use the `ibeos` function to define/disable EOS character detection.

## Example

This example uses the `ibcmd` routine to instruct GPIB board 1 to unlisten all devices (ASCII ?, hex 3F), and then to address a Talker at MTA26 (ASCII Z, hex 5) and a Listener at MLA11 (ASCII +, hex 2B). `ibgts` is then called to unassert the ATN line and place the GPIB board in Standby mode. This action allows the Talker to send messages to the Listener. Note that the GPIB commands/addresses are coded using printable ASCII characters, for example, "?Z+".

```
C                       int gpib1;
                        gpib1 = ibfind ("GPIB1");
                        ibcmd (gpib1, "?Z+", 3);
                        ibgts (gpib1, 1);
```

# IBIST

Sets/Clears the IST (Individual Status) Bit of the GPIB board for parallel polls.

## Syntax

```
C                       ibist (int board, int statusbit)
```

## Parameters

board is an integer containing the board handle.

statusbit indicates whether the IST bit is to be cleared or set. If statusbit is non-zero, then the IST bit is set. Otherwise, if statusbit = 0, the IST bit is cleared.

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred. If an error does not occur, the previous IST value is stored in iberr.

## Usage Notes

This routine is used when the GPIB Interface is not the Active Controller.

IST should be SET to indicate to the controller that service is required.

## Example

This example clears GPIB Board 1's IST bit.

```
C                   int gpib1;
                    gpib1 = ibfind ("GPIB1");
                    ibist (gpib1, 0);
```

# IBLINES

Returns the status of the GPIB control lines.

## Syntax

```
C                   iblines (int board, short *clines)
```

## Parameters

`board` is an integer containing the board handle.

## Returns

`clines` contains a valid mask and GPIB control line state data. Low-order bytes (bits 0 through 7) contain the mask indicating the capability of the GPIB interface board to sense the status of each GPIB control line. Upper bytes (bits 8 through 15) contain the GPIB control line state information. The pattern of each byte is as follows:

| High | | | | | | | |
|------|------|------|------|------|------|------|------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| EOI | ATN | SRQ | REN | IFC | NRFD | NDAC | DAV |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Low (Mask) | | | | | | | |

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

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

Handshake Information:

- NRFD = Not Ready for Data
- NDAC = Not Data Accepted
- DAV = Data Valid

Interface Management:

- ATN = Attention

- IFC = Interface Clear

- REN = Remote Enable

- SRQ = Service Request

- EOI = End or Identify

## Usage Notes

In order for this call to function properly, all devices attached to the GPIB bus must adhere to
IEEE-488 specification.

## Example

This example tests the state of the ATN line.

```
C               #define ATNLINE = 0x40
                int lines;
                iblines (board, &lines);
                if (lines & ATNLINE == 0)
                    printf "ATN line can not be monitored by this
                GPIB board";
                else ( (lines >> 8) & ATNLINE ) == 0)
                    printf "ATN line is not asserted";
```

## IBLN

Check that a device is present on the bus.

### Syntax

```
C                    ibln (int board, int pad, int sad, short* listen)
```

### Parameters

`board` is the board or device handle.

`pad` is the primary address of the GPIB device (0-30).

`sad` is the secondary address of the GPIB device (96-126 or 0x60-0x7e) or one of the constant values NO_SAD or ALL_SAD.

`listen` is the variable that the result is returned to.

### Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code if an error occurred.

`listen` will contain 0 if no listener is found. Contains non-zero if a listener is found.

### Usage Notes

Set `sad` = NO_SAD (0) if the device does not have a secondary address.

Set `sad` = ALL_SAD (-1) if you do not know the device's secondary address and you want all possible secondary addresses to be tested.

### Example

This example tests for the presence of a device with a GPIB address of 4.

```
C                    int board, listen;
                     board = ibfind ("GPIB0");
                     ibln (board, 4, NO_SAD, &listen);
```

# IBLOC

Forces the specified board/device to go to local program mode.

## Syntax

```
C                      ibloc (int boarddev)
```

## Parameters

`boarddev` is an integer containing the device or board handle.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine is used to place boards or devices temporarily in local mode. If this routine specifies a *device*, the following GPIB commands are sent:

- Talk address of the access board
- Secondary address of the access board
- Unlisten (UNL)
- Listen address of the device
- Secondary address of the device (as necessary)
- Go to Local (GTL)

If this routine specifies a *board*, the board is placed in a local state by sending the Return to Local (RTL) message, if it is not locked in remote mode. The LOK bit of the status word indicates whether the board is in a lockout state. The IBLOC function is used to simulate a front panel RTL switch if the computer is used as an instrument.

## Example

Return GPIB board 1 to local state.

```
C                      int gpib1;
                       gpib1 = ibfind("GPIB1");
                       ibloc (gpib1);
```

# IBONL

Enables/Disables a device/interface board for operation.

## Syntax

```
C                    ibonl (int boarddev, int online)
```

## Parameters

`boarddev` is an integer containing the device/board handle.

`online` defines whether the device/board is to be enabled/disabled. If `online` is non-zero, the device/board is enabled for operation (placed on-line). This restores the board/device to its default settings. Otherwise, the board/device is placed off-line.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When a device is placed off-line, it is "closed". This means that in order to perform any other operations with this device, you will need to re-open it by calling the `ibfind` or `ibdev` routine.

## Example

This example restores the configuration of a device at PAD 1.

```
C                    int Dev;
                     Dev = ibdev (0,1,0,13,1,0);
                     ibonl (Dev, 1);
```

# IBPAD

Changes the primary address assigned to a device or interface board.

## Syntax

```
C                         ibpad (int boarddev, int address)
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`address` specifies the new primary GPIB address. Valid primary addresses range from 0 to 30 (0 to 1E hex).

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code if an error occurred. Contains the previous primary address if no error occurred. `EARG` error occurs if address is out of range.

## Usage Notes

This routine temporarily changes the configuration setting. It remains in effect until `ibonl` or `ibfind` is called, `ibpad` is called again, or the system is re-started.

If a *device* is specified, its talk and listen addresses are assigned on the basis of `address`. Its Listen Address equals `address` + 20 hex. Its Talk Address equals `address` + 40 hex. Thus, if a primary address of 0D hex was specified, the corresponding Listen Address would be 2D hex (MLA 13) and Talk Address would be 4D hex (MTA 13). If a *board* is specified, the board is assigned the primary address defined by `address`. Refer also to `ibsad` and `IBONL`.

Be sure that the address specified agrees with the GPIB address of the device. (Set with hardware switches or by a software program. Refer to the device's documentation for more information.)

## Example

This example changes the primary GPIB address associated with a DVM at PAD 4 to 1C hex.

```
C                   int dvm;
                    dvm = ibdev (0,4,0,13,1,0);
                    ibpad (dvm, 0x1C);
```

# IBPCT

Passes control to another device.

## Syntax

```
C                    ibpct (int device)
```

## Parameters

`device` an integer containing the device handle.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This makes the specified device the Controller-In-Charge (CIC). The GPIB board goes to the Controller Idle state and releases the ATN line.

The device that control is passed to must have Controller capability.

## Example

This example makes a device at PAD 1 the Controller-In-Charge.

```
C                    int Dev;
                     Dev = ibdev (0,1,0,13,1,0);
                     ibpct(Dev);
```

# IBPPC

Enables/Disables parallel polling of the specified device.

## Syntax

```
C                       ibppc (int boarddev, int command)
```

## Parameters

boarddev is an integer containing the board or device handle. This value is obtained by calling the ibfind routine.

command is a valid parallel poll enable/disable message or 0. If command represents a *PPE message*, then the device will use that message to respond to a parallel poll. Valid PPE messages range from 60 to 6F hex. The PPE specifies the GPIB data line (DIO1 through DIO8) on which the device is to respond and whether that line is to be asserted or unasserted.

The PPE byte is of the format:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | SENSE | P2 | P1 | P0 |

Where SENSE indicates the condition under which the data line is to be asserted. The device compares the value of the sense bit to its IST (individual status) bit and responds appropriately. For example, if SENSE = 1, the device will drive the line TRUE if its IST = 1 or FALSE if IST = 0.

P2–P0 specify which GPIB data line should be used to respond to a parallel poll, as shown in Table 3-5.

**Table 3-5.**  Values for P2–P0

| P2 | P1 | P0 | GPIB Data Line |
|---|---|---|---|
| 1 | 1 | 1 | DIO8 |
| 1 | 1 | 0 | DIO7 |
| 1 | 0 | 1 | DIO6 |
| 1 | 0 | 0 | DIO5 |
| 0 | 1 | 1 | DIO4 |

**Table 3-5.** Values for P2–P0

| P2 | P1 | P0 | GPIB Data Line |
|---|---|---|---|
| 0 | 1 | 0 | DIO3 |
| 0 | 0 | 1 | DIO2 |
| 0 | 0 | 0 | DIO1 |

For example, if the PPE byte 01101011 (hex 6B) is sent, the device will drive DIO4 true if its IST bit = 1, or false if its IST bit = 0.

If command is 0 or represents a *PPD (Parallel Poll Disable) message*, the current PPE (Parallel Poll Enable) configuration is cancelled. Valid PPD messages range from 70 to 7F hex. The PPD is of a similar format to the PPE byte, for example:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | SENSE | P2 | P1 | P0 |

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred. Contains the previous value of command if no error occurs.

## Usage Notes

If boarddev specifies a *GPIB interface board*, this routine sets the board's Local Poll Enable (LPE) message to command.

If boarddev specifies a *device*, the GPIB Interface Board associated with the device addresses itself as a Talker, unlistens all devices (sends a UNL), addresses the specified device as a Listener, and sends the PPC command followed by a PPE or PPD command.

## Example

This example configures a device at PAD 2 to send DIO7 true if its IST bit = 1.
```
C                 int dev2;
                  dev2 = ibdev (0,2,0,13,1,0);
                  ibppc (dev2, 0x6B);
```

# IBRD

Reads data from a device/interface board into a string.

## Syntax

```
C                ibrd (int boarddev, char buf[], unsigned long
                 bytecount)
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`buf` is the storage buffer for the data. Up to 4 gigabytes-1 ($2^{32}$-1 bytes) can be stored. String size may be limited by the language you are using. Check documentation for your language.

`bytecount` specifies the maximum number of bytes to read.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR results if the specified GPIB Interface Board is an Active Controller but has not been addressed to listen. An EABO error results if a timeout occurs.

`ibcnt`, `ibcntl` will contain the number of bytes that were read. `ibcnt` is a 16-bit integer. `ibcntl` is a 32-bit integer. If the requested count was greater than 64 K, use `ibcntl` instead of `ibcnt`.

## Usage Notes

A read will terminate when one of the following occurs:

•    The allocated buffer becomes full.

•    An error is detected.

•    The time limit is exceeded.

•    A terminator (or EOI) is detected.

If `boarddev` specifies a device, the specified device is addressed to talk and its associated access board is addressed to listen.

If `boarddev` specifies a GPIB Interface board, you must have already addressed a device as a talker and the board as a listener. If the board is the Active Controller, it will unassert ATN in order to receive data. This routine leaves the board in that state.

## Example

This example reads 90 characters of data from a device at PAD 5.

```
C                   int dev5;
                    char rd [90];
                    dev5 = ibdev (0,5,0,13,1,0);
                    ibrd (dev5, rd, 90);
```

# IBRDA

Reads data asynchronously from a device/interface board into a string.

## Syntax

```
C                       ibrda (int boarddev, char buf[], unsigned long
                        bytecount)
```

## Parameters

`boarddev` is an integer containing the device/board handle.

`buf` is the storage buffer for the data. Up to 4 gigabytes-1 ($2^{32}$-1 bytes) can be stored. String size may be limited by the language you are using. Check documentation for your language.

`bytecount` specifies the maximum number of bytes to read.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code of the first error detected, if an error occurred. An `EADR` results if the specified GPIB board is an Active Controller but has not been addressed to listen. An `EABO` error results if a timeout occurs.

`ibcnt`, `ibcntl` will contain the number of bytes that were read. `ibcnt` is a 16-bit integer. `ibcntl` is a 32-bit integer. If the requested count was greater than 64 K, use `ibcntl` instead of `ibcnt`.

## Usage Notes

Asynchronous I/O is not explicitly supported and will be treated as synchronous.

## Example

In this example, `ibwrt` sends the command "DUMP" to a device. The device responds by sending back a large block of data. `ibrda` begins a transfer of 5000 bytes in the background and the program continues on into the WHILE loop. The WHILE loop calls `ibwait` with MASK set to 0 to update IBSTA. The WHILE loop checks IBSTA to see if `ibrda` has completed, or if an error has occurred. The program may do anything else within the WHILE loop except make other GPIB I/O calls.

```
C                   char *readbuffer[5000];
                    ibwrt (device, "DUMP");
                    ibrda (device, readbuffer, 5000);
                    while ( (ibsta & CMPL+ERR) == 0)
                        ibwait (device, 0)
```

# IBRDF

Reads data from the GPIB into a file.

## Syntax

```
C                    ibrdf (int boarddev, char filename [])
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`filename` is the name of the file (up to 250 characters, including drive/path) in which the data is to be stored. Be certain to specify a drive and path if necessary. This file is automatically opened as a binary file. It is created if it does not already exist.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. An `EFSO` error is generated if the file can not be opened, created, found, written to, or closed.

`ibcnt`, `ibcntl` will contain the number of bytes that were read. `ibcnt` is a 16-bit integer. `ibcntl` is a 32-bit integer. If the requested count was greater than 64 K, use `ibcntl` instead of `ibcnt`.

## Usage Notes

A read terminates when one of the following occurs:

• The allocated buffer becomes full.

• An error is detected.

• The time limit is exceeded.

• A terminator (or EOI) is detected.

• A DCL or SDC command is received from the Active Controller.

If `boarddev` specifies a device, the specified device is addressed to talk and its associated access board is addressed to listen.

If `boarddev` specifies a GPIB Interface board, you must have already addressed a device as a talker and the board as a listener. If the board is the Active Controller, it unasserts ATN in order to receive data. This routine leaves the board in that state.

## Example

This program sends the command "DUMP" to a device. The device responds by sending data back. ibrdf reads the incoming data and stores it in the file called gpib.dat on the C drive.

```C
            ibwrt (boarddev, "DUMP");
            ibrdf (boarddev, "c:gpib.dat");
```

## IBRPP

Initiates a parallel poll.

### Syntax

```
C                     ibrpp (int boarddev, char *command)
```

### Parameters

`boarddev` is an integer containing the device or board handle.

### Returns

`command` will contain the response to the parallel poll.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

### Usage Notes

If this routine is called specifying a *GPIB Interface Board*, the board parallel polls all previously configured devices. If the routine is called specifying a *device*, the GPIB Interface board associated with the device conducts the parallel poll. Note that if the GPIB Interface Board to conduct the parallel poll is not the Controller-In-Charge, an ECIC error is generated.

Before executing a parallel poll, the `ibppc` function should configure the connected devices with `ibppc` to specify how they should respond to the poll.

### Example

This program configures two devices for a parallel poll. It then conducts the poll. It is assumed that voltmeter and scope have already been set by opening the devices with an `ibfind` and board has been set by opening the board with an `ibfind`.

Both devices indicate that they want service by setting their first bit to 1. The first `ibppc` specifies that the first device (`voltmeter`) should drive the DIO1 line high when its first line goes high. The second `ibppc` specifies that the second device (`scope`) should drive the DIO2 line high when its first bit goes high. The `ibrpp` conducts the poll and checks DIO1 and DIO2 to see if either device is requesting service.

```
C                int voltmeter, scope, board, pollbyte;
                 ibppc (voltmeter, 0x68)
                 ibppc (scope, 0x69)
                 ibrpp (board, &pollbyte)
                 if (pollbyte & 1)
                     printf "Volt meter is requesting service"
                 if (pollbyte & 2)
                     printf "Oscilloscope is requesting service"
```

# IBRSC

Request/Release System Control.

## Syntax

```
C                 ibrsc (int board, int control)
```

## Parameters

`board` is an integer containing the board handle.

`control` indicates whether the GPIB Interface Board is to become the system controller or to relinquish system control capability. If `control` is non-zero, the specified board becomes the system controller on the GPIB. If `control` is 0, the board is not the system controller.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. If no error occurs, `iberr` equals 1 if the specified interface board was previously the system controller or 0 if it was not.

## Usage Notes

There may only be one system controller in a GPIB system.

## Example

This example makes GPIB board 1 the system controller.

```
C                 int gpib1;
                  gpib1 = ibfind ("gpib1");
                  ibrsc (gpib1, 3);
```

## IBRSP

Serial polls a device.

### Syntax

C                          ibrsp (int device, char *serialpollbyte)

### Parameters

device is an integer containing the device handle.

### Returns

serialpollbyte will contain the serial poll response byte of the device. The serial poll response byte is device-specific *with the exception of bit 6*. If bit 6 (hex 40) is set, then the device is requesting service. Consult the device's documentation for more information.

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred.

### Usage Notes

If the automatic serial polling feature is enabled, the specified device may have been automatically polled previously. If it has been polled and a positive response was obtained, the RQS bit of ibsta is set on that device. In this case ibrsp returns the previously acquired status byte. If the RQS bit of ibsta is not set during an automatic poll, it serial polls the device. This routine is used to serial poll one device, and obtain its status byte or to obtain a previously stored status byte. If bit 6 (the hex 40 bit) of the response is set, the device is requesting service.

When a serial poll occurs, the following sequence of events happens. The board sends an UNL (unlisten) command. It then addresses itself as a listener and sends a SPE (Serial Poll Enable) Byte. It then addresses a device as a talker. The board then reads the serial poll response byte from the device. The board then sends a serial poll disable (SPD) and untalks and unlistens all devices.

### Example

Returns the serial response byte (into serialpollbyte) of a device at PAD 1.

```
C                   int dev1;
                    char serialpollbyte;
                    dev1 = ibdev (0,1,0,13,1,0);
                    ibrsp (dev1, &serialpollbyte);
```

# IBRSV

Changes the serial poll response byte.

## Syntax

```
C                       ibrsv (int board, int statusbyte)
```

## Parameters

`board` is an integer containing the board handle.

`statusbyte` represents the serial poll response byte of the GPIB Interface Board. The serial poll response byte is system-specific, *with the exception of bit 6 (hex 40)*. If bit 6 (hex 40) is set, then the SRQ line is asserted to indicate to the Controller-In-Charge that the board is requesting service.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. If no error occurs, `iberr` will contain the previous value of `statusbyte`.

## Usage Notes

This routine is used when the specified GPIB Interface Board is not the Controller-In-Charge. It can be used to request service (set bit 6 of the serial response byte) from the Controller-In-Charge or to change the value of GPIB Interface Board's serial poll response byte.

## Example

This example sets the GPIB Interface Board 1 serial poll status byte to 41 hex (assert SRQ) which indicates that the board requires service.

```
C                   int gpib1;
                    gpib1 = ibfind ("gpib1");
                    ibrsv (gpib1, 0x41);
```

# IBSAD

Assigns/unassigns a secondary address to a board or device.

## Syntax

```
C                       ibsad (int boarddev, int address)
```

## Parameters

`boarddev` is an integer containing device or board handle.

`address` represents the secondary address. If `address` = 0 or `address` = 7F hex, secondary addressing is disabled. If `address` is a legal secondary address (60 to 7E hex), the new secondary address is temporarily assigned to the board/device. The new secondary address is used until it is either redefined by calling `ibsad` again, the device/board is re-initialized by calling `ibfind` or `ibonl`, or the program is restarted.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurs. Contains the previously assigned secondary address if no error occurs.

## Usage Notes

See also `ibpad`.

## Example

This example assigns the secondary address 7 (MSA7, hex 67) to a device at PAD 5.

```
C                   int dev5;
                    dev5 = ibdev (0,5,0,13,1,0);
                    ibsad (dev5, 0x67);
```

## IBSIC

Asserts IFC (Interface Clear) signal. This re-initializes the GPIB system.

## Syntax

```
C                    ibsic (int board)
```

## Parameters

`board` is an integer containing the board handle.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. The `ESAC` error is generated if the specified GPIB Interface Board is not the system controller.

## Usage Notes

*This routine can only be used if the specified GPIB board is the system controller.* When the routine is executed, the GPIB interface board asserts the `IFC` (Interface Clear) signal for at least 100 μsec. This action results in the system controller regaining control of the GPIB (for example, becoming the Controller-In-Charge). When `IFC` line is asserted, all GPIB interface functions of the bus devices are reset.

## Example

This example resets the GPIB bus associated with the specified GPIB Interface Board and makes that board Controller-In-Charge.

```
C                int gpib1;
                 gpib1 = ibfind("GPIB1");
                 ibsic (gpib1);
```

## IBSRE

Asserts/Unasserts the REN (Remote Enable) line.

### Syntax

```
C                    ibsre (int board, int ren)
```

### Parameters

`board` is an integer containing the board handle.

`ren` specifies whether the REN line is to be asserted or unasserted. If `ren` is zero, the REN line is unasserted. Otherwise, the REN line is asserted.

### Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. The ESAC error is generated if the specified GPIB interface board is not the system controller. Contains the previous REN state if no error occurs.

### Usage Notes

This routine can only be used if the specified GPIB interface board is the system controller.

Even though the REN line is asserted, the device(s) is not put into remote state until is addressed to listen by the Active Controller. When the REN line is unasserted, *all devices* return to local control.

### Example

This example puts the device at MLA 12 (2C hex, ASCII ,) and associated with GPIB Interface Board 1 in remote mode.

```
C                    int gpib1:
                     gpib1 = ibfind ("GPIB1");
                     ibsre (gpib1, 2);        /* Use any non-zero
                                                 value */
                     ibcmd (gpib1,",", 1);
```

## IBSTOP

Terminate an asynchronous operation.

### Syntax

```
C                    ibstop (int boarddev)
```

### Parameters

`boarddev` is an integer containing the device or board handle.

### Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*. If an operation is terminated, the `ERR` bit is set.

`iberr` will contain an error code, if an error occurred. If an operation is terminated, an `EABO` error is returned.

### Usage Notes

If a *device* is specified, all unfinished asynchronous operations (read, write, or command) associated with that device is stopped. If a *GPIB Interface Board* is specified, all unfinished asynchronous operations associated with that board is stopped. Once the operation(s) have been terminated, the application is resynchronized with the driver.

### Example

This example starts a background write command and then immediately stops it.

```
C                    int dev;
                     dev = ibdev (0,2,0,13,1,0);
                     ibwrta(dev, "datafile");
                     ibstop (dev);
```

# IBTMO

Changes timeout value.

## Syntax

```
C                  int ibtmo (int boarddev, int timeout)
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`timeout` specifies the timeout. The timeout value determines how long I/O routines wait for a response from a device. When the timeout period expires during an I/O operation, the I/O function returns an `EABO` error. Valid timeout codes are shown in Table 3-6.

**Table 3-6.**  Timeout Codes

| Code | Value | Minimum timeout | Code | Value | Minimum timeout |
|------|-------|-----------------|------|-------|-----------------|
| TNONE | 0 | Disabled | T100ms | 9 | 100 msec |
| T10us | 1 | 10 msec | T300ms | 10 | 300 msec |
| T30us | 2 | 30 msec | T1s | 11 | 1 sec |
| T100us | 3 | 100 msec | T3s | 12 | 3 sec |
| T300us | 4 | 300 msec | T10s | 13 | 10 sec |
| T1ms | 5 | 1 msec | T30s | 14 | 30 sec |
| T3ms | 6 | 3msec | T100s | 15 | 100 sec |
| T10ms | 7 | 10msec | T300s | 16 | 300 sec |
| T30ms | 8 | 30 msec | T1000s | 17 | 1000 sec |

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. Contains the previous timeout code if no error occurs.

## Usage Notes

This routine is used to temporarily change the default timeout value assigned to the device/GPIB Interface board.

The new timeout is used until it is redefined (by calling `ibtmo` again) the device/board is re-initialized (by calling `ibfind` or `ibonl`); or the system is restarted.

## Example

This example changes the timeout (to 30 sec) for all calls specifying the "plotter" device at PAD 3.

```
C                      int plotter;
                       plotter = ibdev (0,3,0,13,1,0);
                       ibtmo(plotter, T30us);
```

# IBTRG

Triggers the specified device.

## Syntax

```
C                        ibtrg (int device)
```

## Parameters

`device` is an integer containing device handle.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the GPIB Interface Board associated with the device is addressed to talk and all devices are unlistened. The specified device is then addressed to listen and a GET (Group Execute Trigger) command is sent.

## Example

This example triggers the specified device.

```
C                    int plotter;
                     plotter = ibdev (0,6,0,13,1,0);
                     ibtrg (plotter);
```

# IBWAIT

Forces application program to wait for a specified event(s) to occur.

## Syntax

```
C                    ibwait (int boarddev, int mask)
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`mask` specifies the events that `ibwait` will wait for. Each bit in mask represents a different event. These bits are the same as the bits in `ibsta` positions.

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| **Event** | ERR | TIMO | END | SRQI | RQS | | | CMPL |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Event** | LOK | REM | CIC | ATN | TACS | LACS | DTAS | DCAS |

For more information regarding `ibsta`, see Appendix B, *IBSTA*.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

Because the mnemonic for each bit of `ibsta` is defined as a constant within the header file, you can elect to use the mnemonic rather than the hex value. If `TIMO` is set, `ibwait` returns if the event does not occur within the timeout period of the device.

If a GPIB interface board is specified, the `RQS` bit is not applicable.

## Example

This example forces your program to wait indefinitely for the specified device to request service.

```
C                     int plotter;
                      plotter = ibdev (0,1,0,13,1,0);
                      ibwait (plotter, RQS);
```

# IBWRT

Writes data from a string to the specified device or GPIB Interface Board.

## Syntax

C                    ibwrt (int boarddev, char buf[], unsigned long
                     bytecount)

## Parameters

boarddev is an integer containing the board or device handle.

buf is the string containing the data to be written. buf can contain up to 4 gigabytes-1 ($2^{32}$-1 bytes). String size may be limited by the language you are using. Check documentation for your language.

bytecount specifies the number of bytes to be written from the string.

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code of the first error detected, if an error occurred. An EADR results if boarddev specifies a board and the board has not already been addressed to talk.

ibcnt, ibcntl will contain the number of bytes that were written. ibcnt is a 16-bit integer. ibcntl is a 32-bit integer. If the requested count was greater than 64 K, use ibcntl instead of ibcnt.

## Usage Notes

This routine is used to send device-specific commands. A write terminates when one of the following occurs:

* All bytes are transferred.

* An error is detected.

* The time limit is exceeded.

* A DCL (Device Clear) or SDC (Selected DC) is received from the CIC.

* All data is sent.

If boarddev specifies a device, the specified device is addressed to listen and its associated access board is addressed to talk. If boarddev specifies a GPIB Interface Board, the Controller-In-Charge must have already addressed a device as a listener and the board as a talker. If the board is the Active Controller, it unasserts ATN in order to send data. This routine leaves the board in that state.

If you want to send an EOS character at the end of the data string, you must include it in the string.

## Example

This example sends five bytes terminated by a carriage return and line feed to the specified device.

```
C                     int ptr;
                      ptr = ibdev (0,7,0,13,1,0);
                      ibwrt (ptr,"IP2X5\r\n", 7);
```

# IBWRTA

Writes data asynchronously from a string to the specified device or GPIB interface board.

## Syntax

```
C                    ibwrta (int boarddev, char buf[], unsigned long
                     bytecount)
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`buf` is the storage buffer for the data. Up to 4 gigabytes-1 ($2^{32}$-1 bytes) can be stored. String size may be limited by the language you are using. Check documentation for your language.

`bytecount` specifies the number of bytes to be written.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code of the first error detected, if an error occurred. An `EADR` results if `boarddev` specifies a board and the board has not already been addressed to talk.

`ibcnt`, `ibcntl` will contain the number of bytes that were written. `ibcnt` is a 16-bit integer. `ibcntl` is a 32-bit integer. If the requested count was greater than 64 K, use `ibcntl` instead of `ibcnt`.

## Usage Notes

Asynchronous I/O is not explicitly supported and will be treated as synchronous.

## Example

In this example, `ibwrt` sends a command ("UPLOAD") to a device. The device expects a block of data to be sent immediately. `ibwrta` begins a transfer of 5000 bytes in the background and program continues on into the WHILE loop. The WHILE loop calls `ibwait` with MASK set to 0 to update `ibsta`. The WHILE loop checks `ibsta` to see if `ibwrta` has completed or any error have occurred. The program may do anything else within the WHILE loop except make other GPIB I/O calls.

```
C                    char writebuffer[5000];
                     ibwrt (device, "UPLOAD")
                     ibwrta (device, writebuffer, 5000);
                     while ( (ibsta & CMPL+ERR) == 0)
                         ibwait (device, 0)
```

# IBWRTF

Writes data from a file to the specified device or GPIB Interface Board.

## Syntax

```
C                       ibwrtf (int boarddev, char filename [])
```

## Parameters

`boarddev` is an integer containing the board or device handle.

`filename` is the name of the file (up to 50 characters, including drive/path) to store the data. Specify a drive and path if necessary. This file is automatically opened as a binary file.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. An `EFSO` error is generated if the file can not be found.

`ibcnt`, `ibcntl` will contain the number of bytes that were written. `ibcnt` is a 16-bit integer. `ibcntl` is a 32-bit integer. If the requested count was greater than 64 K, use `ibcntl` instead of `ibcnt`.

## Usage Notes

A write terminates when one of the following occurs:

• An error is detected.

• The time limit is exceeded.

• A DCL or SDC is received from the Active Controller.

• All data has been sent.

If `boarddev` specifies a *device*, the specified device is addressed to talk and its associated access board is addressed to listen. If `boarddev` specifies a *GPIB interface board*, you must have already addressed a device as a listener and the board as a talker. If the board is the CIC, it unasserts ATN in order to receive data. This routine leaves the board in that state.

## Example

This program sends the command "`UPLOAD`" to a device and prepares the device to receive a large amount of data. The program then sends the data from a file to the device.

```
C                       int device;
                        ibwrt (device, "UPLOAD");
                        ibwrtf (device, "c:gpib.dat");
```

# 4

# GPIB 488.2 Library Reference

This chapter describes each of the 488.2 GPIB library routines. A short description of the routine, its syntax, parameters, any values that are returned, any special usage notes, and an example are included for each routine. The routines are listed in alphabetical order. The following table lists all of the 488.2 GPIB library routines. A full description of each routine follows the table.

**Note** 488.2 addresses contain two bytes packed into a word – the low byte is the primary address and the high byte is the secondary address. If secondary addressing is not used, the high byte should be zero.

**Table 4-1.** 488.2 Address word

| HIGH BYTE | LOW BYTE |
|---|---|
| Secondary Address (0 or 96-126) | Primary Address (0-30) |

488.2 routines use a board index as the first argument (typically zero) – not a handle.

# AllSpoll

Performs a serial poll on specified devices.

## Syntax

```
C                    AllSpoll (int board, short addresslist[],
                     short resultlist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR. These addresses identify the devices to be serial polled.

## Returns

`resultlist` is an array which contains the results of the serial poll. Once a device has been serial polled, the results of the serial poll are stored in the corresponding element of `resultlist`.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. If a device times out, `iberr` contains Error 6 – EABO (see Appendix C, *IBERR*), and `ibcnt` contains the index of the timed-out device.

## Usage Notes

To poll only one GPIB device, use ReadStatusByte.

## Example

This example serial polls two devices (GPIB address 6 and 7) connected to GPIB board 0.

```
C                    short addresslist[3] = {6,7,NOADDR};
                     short resultlist[2];
                     AllSpoll (0, addresslist, resultlist);
```

# DevClear

Clears one device.

## Syntax

```
C                    DevClear(int board, short address)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`address` is the GPIB address of the device to clear.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine sends the GPIB Selected Device Clear (SDC) message to the specified device.

To clear multiple devices, use the `DevClearList` routine.

If `address` is set to `NOADDR`, then all connected devices on the GPIB is cleared through the Universal `Device Clear` (UDC) message.

## Example

This example clears the device at GPIB primary address 4, secondary address 30 connected to GPIB board 0.

```
C                    DevClear(0, MakeAddr (4,30));
                        /* Use MakeAddr macro (in GPIB.H) to pack
                        primary and secondary address */
```

# DevClearList

Clears specified devices.

## Syntax

C                    DevClearList(int board, short addresslist[])

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR. These addresses identify the devices to be cleared.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine sends the GPIB Selected Device Clear (SDC) to the devices specified by `addresslist`.

To clear only one device, use `DevClear`.

## Example

This clears the devices at GPIB addresses 6 and 7, connected to GPIB board 0.

```
C                    short addresslist[3] = {6,7,NOADDR};
                     DevClearList(0, addresslist);
```

# EnableLocal

Places specified devices in local mode (Can be "programmed" from front panel controls.).

## Syntax

```
C                    EnableLocal(int board, short addresslist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. These addresses identify the devices to enable locally.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the Controller addresses the specified GPIB devices as listeners and then sends the GPIB `Go To Local` (GTL) command.

To put all devices in local mode, use an array containing only the `NOADDR` value. This unasserts the GPIB `Remote Enable` (REN) line, thereby placing all GPIB devices in local mode.

## Example

Put the GPIB devices at addresses 6 and 7 (connected to board 0) in local mode.

```
C                    short addresslist[3] = {6,7,NOADDR};
                     EnableLocal(0, addresslist);
```

# EnableRemote

Allow remote programming (by sending messages over the GPIB line) of a device.

## Syntax

```
C                       EnableRemote(int board, short addresslist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. These addresses identify the devices to be put in remote programming mode.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the system controller asserts the `Remote Enable` (REN) line and the Controller addresses the specified devices as listeners.

## Example

Places devices at GPIB addresses 6 and 7 (connected to GPIB board) in remote mode.

```
C                       short addresslist[3] = {6,7,NOADDR};
                        EnableRemote(0, addresslist);
```

# FindLstn

Finds all listeners on the GPIB.

## Syntax

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

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR.

`limit` is an integer which specifies how many address entries can be placed into the `resultlist` array. Set to the size of the `resultlist` array.

## Returns

`resultlist` will contain the addresses of all detected listeners. This array must be large enough to hold all possible addresses.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. An ETAB (20) error indicates that more listeners are present on the GPIB bus than limit will allow to be placed in `resultlist`. In this case, `ibcnt` contains the number of addresses actually placed in `resultlist`.

## Usage Notes

The addresses specified by `addresslist` are tested to see if a listening device is present. If a listener is found at a primary address, its address is placed in `resultlist`. If no listeners are detected at a primary address, then all secondary addresses associated with that primary address are tested. If any listeners are detected, their addresses are placed in `resultlist`. You can use this routine to determine how many devices on the network are capable of listening. Once these devices are detected, they can be identified by their response to identification request messages.

## Example

This example verifies if listening devices are present at GPIB primary addresses 6 and 7 on Board 0.

```
C                   short addresslist[3] = {6,7,NOADDR};
                    short resultlist[4];
                    FindLstn(0, addresslist, resultlist, 4);
```

# FindRQS

Identify the device requesting service.

## Syntax

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

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. The devices located at these addresses are serial polled until the one asserting SRQ is located.

## Returns

`result` will contain the returned status byte of the device asserting SRQ.

`ibcnt` will contain the index (in `addresslist`) identifying the device's address.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred. `iberr` contains the error code `ETAB`, if no device is requesting service. In this case, `ibcnt` contains `NOADDR`'s index.

`iberr` will contain the error code `EABO` if a device times out while responding to its serial poll. In this case, `ibcnt` contains the index of the timed-out device.

## Usage Notes

None.

## Example

Identifies which of the devices at GPIB addresses 6 and 7 (connected to board 0) is requesting service.

```
C
                    short addresslist[3] = {6,7,NOADDR};
                    short result;
                    FindRQS (0, addresslist, &result);
```

# PassControl

Makes another device the Active Controller.

## Syntax

```
C                    PassControl(int board, short address)
```

## Parameters

board is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

address is an integer representing the GPIB address of the device that is to become the controller. The low byte of the integer contains the device's primary GPIB address. The high byte of the address contains the device's secondary GPIB address. If the device has no secondary address, the high byte of address is 0.

## Returns

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the GPIB Take Control (TCT) command is issued. This forces the Active Controller to pass control to the device at the specified address. This device must have Controller capability.

## Example

This example would make the device connected to Board 0 and whose GPIB address is 6 the Active Controller.

```
C                    PassControl(0, 6);
```

# PpoII

Performs a parallel poll.

## Syntax

C                        PPoll(int board, short *result)

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

## Returns

`result` will contain the eight-bit result of the parallel poll. Each bit of the poll result contains one bit of status information from each device which has been configured for parallel polls. The value of each bit is dependent on the latest parallel poll configuration sent to the devices through `PPollConfig` and the individual status of the devices.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage notes

None.

## Example

Parallel polls devices connected to board 0.

```
C                   short result;
                    PPoll(0, &result);
```

# PPollConfig

Configures a device for parallel polls.

## Syntax

```
C                 PPollConfig(int board, short address, int
                  dataline, int sense)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`address` is the address of the GPIB device to be configured for a parallel poll.

`dataline` specifies which data line (1-8) the device uses to respond to a parallel poll.

`sense` can be 1 or 0, specifying the condition under which the data line is to be asserted/unasserted. The device compares this value to its Individual Status Bit (IST) and then responds accordingly. For example, if `sense` = 0 and the device asserts the specified data line if its IST bit = 0 and unassert the data line if its IST bit = 1.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage notes

Remember that if a device is locally configured for a parallel poll, the Controller's parallel poll configuration instruction is ignored.

## Example

Configures the device connected to board 0 at address 6 to respond to parallel polls on line 7 with sense 1. The device asserts line 7 if its IST bit = 1, and unasserts line 7 if IST = 0.

```
C                 PPollConfig(0, 6, 7, 1)
```

# PPollUnconfig

Unconfigures devices for parallel polls.

## Syntax

```
C                       PPollUnconfig(int board, short addresslist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. These addresses identify the devices that do not respond to a parallel poll.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Example

Unconfigure the devices connected to board 0 and located at GPIB addresses 6 and 7.

```
C                       short addresslist[3] = {6, 7, NOADDR};
                        PPollUnconfig(0, addresslist);
```

# RcvRespMsg

Reads data from a previously addressed device.

## Syntax

```
C                 RcvRespMsg(int board, char data[], long count,
                  int termination)
```

## Parameters

board is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

count specifies the maximum number of data bytes which are to be read.

termination is the flag used to signal the end of data. If termination equals a value between 0 and 00FF hex, the corresponding ASCII character is the termination character. The read is stopped when this character is detected. If termination = STOPend (A constant defined in the header file), then the read is stopped when EOI is detected.

## Returns

data is the string that receives the data.

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred.

## Usage Notes

You must address the appropriate devices as Listeners/Talkers prior to calling this routine. The input data string is *not* terminated with a zero byte.

## Example

A previously addressed Listener receives 50 bytes of data from a previously addressed Talker. The transmission is terminated when EOI is detected.

```
C                 char data[50];
                  RcvRespMsg(0, data, 50, STOPend)
```

# ReadStatusByte

Serial poll a single device and read its status byte.

## Syntax

```
C                    ReadStatusByte(int board, int address, short
                     *result)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`address` is an integer representing the GPIB address of the device that is to be serial polled.

## Returns

`result` will contain the status byte. The high byte of result is always 0.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

None.

## Example

This example serial polls the device at address 2 and retrieves its status byte.

```
C                    short result;
                     ReadStatusByte (0, 2, &result);
```

# Receive

Reads data from a GPIB device.

## Syntax

```
C                    Receive(int board, int address,char data[],
                     unsigned long count, int termination)
```

## Parameters

board is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

address is an integer representing the GPIB address of the device that is to be read from.

count specifies the maximum number of data bytes which are to be read.

termination is the flag used to signal the end of data. If termination equals a value between 0 and 00FF hex, the corresponding ASCII character is the termination character. The read is stopped when this character is detected. If termination = STOPend (constant defined in the header file), then the read is stopped when EOI is detected.

## Returns

data is the string that receives the data.

ibsta will contain a 16-bit status word as described in Appendix B, *IBSTA*.

iberr will contain an error code, if an error occurred.

## Usage Notes

The input data string is *not* terminated with a zero byte.

## Example

Receive 50 bytes of data from the specified talker (device at address 2, connected to board). EOI signals the end of the message.

```
C                char data[50];
                 Receive (0, 2, data, 50, STOPend);
```

# ReceiveSetup

Address a GPIB Interface Board as a Listener and a GPIB device as a Talker, in preparation for data transmission.

## Syntax

```
C                     ReceiveSetup(int board, short address)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`address` is an integer representing the GPIB address of the device to send the data.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

In order to actually transfer any data, you must call a routine such as `RcvRespMsg` following this routine.

This routine is useful in instances where you need to transfer multiple blocks of data between devices. For example, you could initially address the devices using `ReceiveSetup`, then make multiple calls of `RcvRespMsg` to transfer the data.

For typical cases, `Receive` is simpler to use, since it takes care of both the setup and the data transfer.

## Example

This example instructs a GPIB device at address 5 to send data to GPIB Board 0. Up to 50 bytes of data is received and then stored in a string. The message is terminated with an EOI.

```
C                 char message[100];
                  ReceiveSetup(0, 5);
                  RcvRespMsg (0, message, 50, STOPend);
```

# ResetSys

Initializes GPIB System.

## Syntax

```
C                    ResetSys(int board, short addresslist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR. These addresses identify the devices on the system to be reset.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine initializes the GPIB bus and all specified devices. First, the system controller asserts the REN (Remote Enable) line and then the IFC (Interface Clear) line. This action unlistens and untalks all of the attached GPIB devices and causes the system controller to become the Controller-In-Charge (CIC).

The Device Clear (DCL) message is then sent to all of the connected devices. This forces the devices to return to their default states and ensures that they can receive the Reset (RST) message. A reset message (RST) is then sent to all of the devices specified by `addresslist`. This resets the devices to specific parameters.

## Example

This example resets the GPIB devices connected to GPIB board 0 and assigned GPIB bus addresses of 6 and 7.

```
C                    short addresslist[3] = {6, 7, NOADDR};
                     ResetSys(0, addresslist);
```

# Send

Sends data to one GPIB device.

## Syntax

```
C                    Send (int board, short address,char data[],
                     long count, int eotmode)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`address` is an integer representing the GPIB address of the device to receive the data.

`data` is the string of data which is sent to the device.

`count` specifies the maximum number of data bytes which are to be sent to the device.

`eotmode` is the flag used to signal the end of data.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the specified GPIB board is addressed as a Talker, the designated GPIB device is addressed as a Listener and the number of bytes (specified by `count`) in `data` is sent. Values for `eotmode` are:

- `NLend`—Send NL (Line Feed) with EOI after last data byte.
- `DABend`—Send EOI with the last data byte in the string.
- `NULLend`—Do not mark the end of the transfer.

These constants are defined in the header file.

## Example

In this example, GPIB board 0 sends an identification query to the GPIB device at address 3. End of data is signalled by an EOI.

```
C                    Send (0, 3, "*IDN?", DABend)
```

# SendCmds

Send GPIB commands.

## Syntax

```
C                    SendCmds (int board, char commands[],
                     unsigned long count)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`commands` is a string containing the GPIB command bytes to be sent.

`count` specifies the maximum number of command bytes which are to be sent.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine is useful in situations where specialized GPIB command sequences are called for.

## Example

The GPIB board (at 0) simultaneously triggers the GPIB devices at addresses 8 and 9 and quickly puts them in local mode.

```
C                    SendCmds0, "\x3F\x40\x28\x29\x04\x01",6);
```

# SendDataBytes

Sends data to previously addressed devices.

## Syntax

```
C                   SendDataBytes(int board, char data[], long count,
                    int eotmode)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`data` is the string that contains the data which is sent to the device.

`count` specifies the maximum number of data bytes which are to be sent to the device.

`eotmode` is the flag used to signal the end of data.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine assumes that the desired GPIB listeners have already been addressed (by using `SendSetup`, for example).

Values for `eotmode` are as follows:

- `NLend`—Send NL (Line Feed) with EOI after last data byte.
- `DABend`—Send EOI with the last data byte in the string.
- `NULLend`—Do not mark the end of the transfer.

These constants are defined in the header files.

## Example

In this example, GPIB board 0 sends an identification query to all previously addressed listeners. End of data is signaled by an EOI.

```
C                   SendDataBytes (0, "*IDN?", 5, DABend)
```

# SendIFC

Clears the GPIB bus by asserting the IFC (Interface Clear) line.

## Syntax

```
C                       SendIFC(int board)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine is used as part of the GPIB initialization procedure. When the system controller asserts the IFC line, it unlistens and untalks all GPIB devices, forcing them to an idle state. The system controller also becomes the Controller-In-Charge (CIC).

## Example

Clears the GPIB bus from Board 0.
```
C                       SendIFC(0);
```

# SendList

Sends data to multiple GPIB devices.

## Syntax

```
C                    SendList(int board, short addresslist[],
                     char data[], long count, int eotmode)
```

## Parameters

`board` is an integer which identifies the GPIB board to use for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. These addresses identify the devices on the system to reset.

`data` is the string containing the data to send.

`count` specifies the maximum number of data bytes to send to the device.

`eotmode` is the flag used to signal the end of data.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the specified GPIB board is addressed as a Talker and the designated GPIB devices as Listeners. The board then sends the given number of bytes of data from the data string to the listening GPIB devices.

- `NLend`—Send NL (Line Feed) with EOI after last data byte.
- `DABend`—Send EOI with the last data byte in the string.
- `NULLend`—Do not mark the end of the transfer.

These constants are defined in the header files.

## Example

In this example, GPIB board 0 sends an identification query to the GPIB devices at addresses 6 and 7. End of data is signalled by an EOI.

```
C                    short addresslist[3] = {6, 7, NOADDR};
                     SendList (0, addresslist, "*IDN?", 5, DABend)
```

# SendLLO

Sends Local Lockout (LLO) message to all GPIB devices.

## Syntax

```
C                        SendLLO(int board)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, the specified GPIB board sends the GPIB Local Lockout (LLO) message to all devices. This means that once they have been addressed as listeners, the devices respond only to messages sent over the GPIB by the Controller. (In other words, they can not be locally programmed from front panel controls.) Only the Controller can return them to a local programming state.

## Example

In this example, GPIB board 0 sends a Local Lockout to its connected GPIB devices.

```
C                    SendLLO (0);
```

# SendSetup

Addresses a GPIB board as a Talker and the specified GPIB devices as Listeners.

## Syntax

```
C                   SendSetup(int board, short addresslist [])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. These addresses identify the devices to address as Listeners.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

Following this routine, you should call a routine such as `SendDataBytes` to actually transfer the data.

## Example

This example prepares GPIB board 0 to send data to GPIB devices 6 and 7.

```
C                   short addresslist[3] = {6, 7, NOADDR};
                    SendSetup(0, addresslist);
```

# SetRWLS

Puts all devices in Remote state with Local Lockout and addresses specified devices as Listeners.

## Syntax

```
C                       SetRWLS(int board, short addresslist [])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR. These addresses identify the devices to be put in remote mode.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This routine puts the specified devices in remote mode with local lockout. The system controller asserts the REN (Remote Enable) line and addresses the specified devices as listeners. These devices can then be programmed by messages sent over the GPIB bus. (In other words, they can not be locally programmed from front panel controls.)

## Example

This example puts all devices controlled by GPIB board 0 into Remote mode. Devices 6 and 7 are then addressed as Listeners by the Controller.

```
C                       short addresslist[3] = {6, 7, NOADDR};
                        SetRWLS(0, addresslist);
```

# TestSRQ

Evaluate state of SRQ line.

## Syntax

```
C                       TestSRQ(int board, short *result)
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

## Returns

`result` is equal to 1 if the GPIB SRQ line is asserted. `result` = 0 if the GPIB SRQ line is unasserted.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

`TestSRQ` will not alter the state of the SRQ line.

## Example

This example tests to see if SRQ is asserted.

```
C                   Short result;
                    TestSRQ (0, &result);
                    if (result == 1)
                        { /* SRQ is asserted */}
                    else
                        { /* No SRQ at this time */}
```

# TestSys

Activate self-test procedures of specified devices.

## Syntax

```
C               SendSetup(int board, short addresslist [],
                short resultlist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to use for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR. These addresses identify the devices to perform self-tests.

## Returns

`resultlist` is an array which contains the results of each device's self-test procedure. According to the IEEE-488.2 standard, a result code of 0 indicates the device passed its test. Any other value indicates an error.

`ibcnt` will contain the number of devices which failed their tests.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this routine is executed, all of the devices identified within the `addresslist` array are concurrently sent a message which directs them to perform their self-test procedures. Each device returns an integer code indicating the results of its tests. This code is placed into the corresponding element of the `resultlist` array.

## Example

This example tells the devices at addresses 6 and 7 (from Board 0) to perform thier self-test procedures.

```
C               short addresslist[3] = {6, 7, NOADDR};
                short resultlist[2];
                TestSys(0, addresslist, resultlist);
```

# Trigger

Triggers one device.

## Syntax

```
C                    Trigger(int board, short address)
```

## Parameters

`board` is an integer which identifies the GPIB board to used for this operation. In most applications, this value is 0.

`address` is an integer representing the GPIB address of the device to trigger. If `address = NOADDR` then all Listeners already addressed are triggered.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

When this call is executed, the GPIB GET (Group Execute Trigger) message is sent to the specified device.

To trigger several GPIB devices, use `TriggerList`.

## Example

This example triggers a device connected to board 0 whose primary GPIB address is 6 and secondary address is 12.

```
C                    Trigger (0, MakeAddr (6, 12));
```

# TriggerList

Triggers multiple GPIB devices

## Syntax

```
C                   void TriggerList(int board, int addresslist[])
```

## Parameters

`board` is an integer which identifies the GPIB board to be used for this operation. In most applications, this value is 0.

`addresslist` is an array of GPIB addresses, terminated by the value NOADDR. These addresses identify the devices to be triggered. If this array contains only NOADDR, all previously addressed listeners are triggered.

## Returns

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

Use `Trigger` to trigger only one device.

## Example

This example triggers two devices simultanously. The devices are connected to board 0 and are at GPIB addresses 6 and 7.

```
C                   short addresslist[3] = {6, 7, NOADDR};
                    TriggerList(0, addresslist);
```

# WaitSRQ

Wait until a device asserts SRQ.

## Syntax

```
C                      WaitSRQ(int board, short *result)
```

## Parameters

`board` is an integer which identifies the GPIB board to use for this operation. In most applications, this value is 0.

## Returns

`result` indicates whether or not an SRQ occurred. If an SRQ occurs before the timeout expires, `result` = 1. Otherwise, `result` = 0.

`ibsta` will contain a 16-bit status word as described in Appendix B, *IBSTA*.

`iberr` will contain an error code, if an error occurred.

## Usage Notes

This call suspends operation until a device requests service or a timeout occurs. Follow this call with a `FindRQS` call to determine which device needs service.

## Example

Wait for a GPIB device to request service and then ascertain if device 6 or 7 requires service.

```
C                      short addresslist[3] = {6,7,NOADDR};
                       short resultlist[3];
                       short result;
                       WaitSRQ (0,&result);
                       if (result == 1)
                            {FindSRQ (0, addresslist, resultlist)}
```

# A

# Multiline Interface Messages

Table A-1. Multiline Interface Messages (Hex 00-3F)

| HEX | DEC | ASCII | MSG | HEX | DEC | ASCII | MSG |
|-----|-----|-------|-----|-----|-----|-------|-----|
| 00 | 0 | NUL | | 20 | 32 | SP | MLA0 |
| 01 | 1 | SOH | GTL | 21 | 33 | ! | MLA1 |
| 02 | 2 | STX | | 22 | 34 | " | MLA2 |
| 03 | 3 | ETX | | 23 | 35 | # | MLA3 |
| 04 | 4 | EOT | SDC | 24 | 36 | $ | MLA4 |
| 05 | 5 | ENQ | PPC | 25 | 37 | % | MLA5 |
| 06 | 6 | ACK | | 26 | 38 | & | MLA6 |
| 07 | 7 | BEL | | 27 | 39 | ' | MLA7 |
| 08 | 8 | BS | GET | 28 | 40 | ( | MLA8 |
| 09 | 9 | HT | TCT | 29 | 41 | ) | MLA9 |
| 0A | 10 | LF | | 2A | 42 | * | MLA10 |
| 0B | 11 | VT | | 2B | 43 | + | MLA11 |
| 0C | 12 | FF | | 2C | 44 | ' | MLA12 |
| 0D | 13 | CR | | 2D | 45 | - | MLA13 |
| 0E | 14 | SO | | 2E | 46 | > | MLA14 |
| 0F | 15 | SI | | 2F | 47 | / | MLA15 |
| 10 | 16 | DLE | | 30 | 48 | 0 | MLA16 |
| 11 | 17 | DC1 | LLO | 31 | 49 | 1 | MLA17 |
| 12 | 18 | DC2 | | 32 | 50 | 2 | MLA18 |
| 13 | 19 | DC3 | | 33 | 51 | 3 | MLA19 |
| 14 | 20 | DC4 | DCL | 34 | 52 | 4 | MLA20 |

**Table A-1.** Multiline Interface Messages (Hex 00-3F) (Continued)

| HEX | DEC | ASCII | MSG | HEX | DEC | ASCII | MSG |
|-----|-----|-------|-----|-----|-----|-------|-----|
| 15 | 21 | NAK | PPU | 35 | 53 | 5 | MLA21 |
| 16 | 22 | SYN | | 36 | 54 | 6 | MLA22 |
| 17 | 23 | ETB | | 37 | 55 | 7 | MLA23 |
| 18 | 24 | CAN | SPE | 38 | 56 | 8 | MLA24 |
| 19 | 25 | EM | SPD39 | 39 | 57 | 9 | MLA25 |
| 1A | 26 | SUB | | 3A | 58 | : | MLA26 |
| 1B | 27 | ESC | | 3B | 59 | ; | MLA27 |
| 1C | 28 | FS | | 3C | 60 | < | MLA28 |
| 1D | 29 | GS | | 3D | 61 | = | MLA29 |
| 1E | 30 | RS | | 3E | 62 | > | MLA30 |
| 1F | 31 | US | | 3F | 63 | ? | UNL |

**Table A-2.** Multiline Interface Messages (Hex 40-7F)

| HEX | DEC | ASCII | MSG | HEX | DEC | ASCII | MSG |
|-----|-----|-------|-----|-----|-----|-------|-----|
| 40 | 64 | @ | MTA0 | 60 | 96 | ' | MSA0,PPE |
| 41 | 65 | A | MTA1 | 61 | 97 | a | MSA1,PPE |
| 42 | 66 | B | MTA2 | 62 | 98 | b | MSA2,PPE |
| 43 | 67 | C | MTA3 | 63 | 99 | c | MSA3,PPE |
| 44 | 68 | D | MTA4 | 64 | 100 | d | MSA4,PPE |
| 45 | 69 | E | MTA5 | 65 | 101 | e | MSA5,PPE |
| 46 | 70 | F | MTA6 | 66 | 102 | f | MSA6,PPE |

**Table A-2.**  Multiline Interface Messages (Hex 40-7F) (Continued)

| HEX | DEC | ASCII | MSG | HEX | DEC | ASCII | MSG |
|-----|-----|-------|------|-----|-----|-------|------|
| 47 | 71 | G | MTA7 | 67 | 103 | g | MSA7,PPE |
| 48 | 72 | H | MTA8 | 68 | 104 | h | MSA8,PPE |
| 49 | 73 | I | MTA9 | 69 | 105 | i | MSA9,PPE |
| 4A | 74 | J | MTA10 | 6A | 106 | j | MSA10,PPE |
| 4B | 75 | K | MTA11 | 6B | 107 | k | MSA11,PPE |
| 4C | 76 | L | MTA12 | 6C | 108 | l | MSA12,PPE |
| 4D | 77 | M | MTA13 | 6D | 109 | m | MSA13,PPE |
| 4E | 78 | N | MTA14 | 6E | 110 | n | MSA14,PPE |
| 4F | 79 | O | MTA15 | 6F | 111 | o | MSA15,PPE |
| 50 | 80 | P | MTA16 | 70 | 112 | p | MSA16,PPD |
| 51 | 81 | Q | MTA17 | 71 | 113 | q | MSA17,PPD |
| 52 | 82 | R | MTA18 | 72 | 114 | r | MSA18,PPD |
| 53 | 83 | S | MTA19 | 73 | 115 | s | MSA19,PPD |
| 54 | 84 | T | MTA20 | 74 | 116 | t | MSA20,PPD |
| 55 | 85 | U | MTA21 | 75 | 117 | u | MSA21,PPD |
| 56 | 86 | V | MTA22 | 76 | 118 | v | MSA22,PPD |

**Table A-2.** Multiline Interface Messages (Hex 40-7F) (Continued)

| HEX | DEC | ASCII | MSG | HEX | DEC | ASCII | MSG |
|-----|-----|-------|-----|-----|-----|-------|-----|
| 57 | 87 | W | MTA23 | 77 | 119 | w | MSA23, PPD |
| 58 | 88 | X | MTA24 | 78 | 120 | x | MSA24, PPD |
| 59 | 89 | Y | MTA25 | 79 | 121 | y | MSA25, PPD |
| 5A | 90 | Z | MTA26 | 7A | 122 | z | MSA26, PPD |
| 5B | 91 | [ | MTA27 | 7B | 123 | { | MSA27, PPD |
| 5C | 92 | \ | MTA28 | 7C | 124 | \| | MSA28, PPD |
| 5D | 93 | ] | MTA29 | 7D | 125 | } | MSA29, PPD |
| 5E | 94 | ^ | MTA30 | 7E | 126 | ~ | MSA30 |
| 5F | 95 | _ | UNT | 7F | 127 | DEL | MSA0 |

**Table A-3.** Message definitions (MSG column)

| DCL | Device Clear | MTA | My Talk Address | SPD | Serial Poll Disable |
|-----|-------------|-----|-----------------|-----|---------------------|
| GET | Group Execute Trigger | PPC | Parallel Port Configure | SPE | Serial Port Enable |
| GTL | Go To Local | PPD | Parallel Poll Disable | TCT | Take Control |
| LLO | Local Lockout | PPE | Parallel Port Enable | UNL | UnListen |
| MLA | My Listen Address *n* | PPU | Parallel Port Unconfigure | UNT | UnTalk |
| MSA | My Secondary Address | SDC | Selected Device Clear | | |

# B

# IBSTA

Every GPIB library routine returns a 16-bit status word to the variable `ibsta`. This status word describes the current condition of the GPIB bus lines and the GPIB Interface Board. By examining the status word, the programmer may determine what path the application program is to take.

Note that the mnemonics used to describe each bit of the status word are defined within the header file for each language.

The status word contains 16-bits. If a bit equals 1 (set), the corresponding condition is true. Likewise, if a bit equals 0, then the corresponding condition has not occurred. The status word is of the following format:

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
|  | ERR | TIMO | END | SRQI | RQS |  |  | CMPL |
| Hex Value | 8000 | 4000 | 2000 | 1000 | 800 |  |  | 100 |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
|  | LOK | REM | CIC | ATN | TACS | LACS | DTAS | DCAS |
| Hex Value | 80 | 40 | 20 | 10 | 8 | 4 | 2 | 1 |

**Table B-1.** Bit Definitions

| Bit | Description |
|---|---|
| ERR | **GPIB Error**. If this bit = 1, an error has occurred during the call. An error code describing the exact error is returned to the iberr variable. See Appendix C, *IBERR* for more information. ERR is cleared following any call that does not result in an error.<br><br>Check for errors after each call. If an undetected error occurs early in your program, it may not be apparent until later in the program, when it is more difficult to isolate. |
| TIMO | **Timeout Error**. If this bit = 1, a time-out has occurred. Some of the conditions which may result in this are:<br><br>A synchronous I/O function exceeds the programmed timeout.<br><br>An ibwait has exceeded the time limit value.<br><br>This bit is cleared (set to 0) during all other operations.<br><br>Different timeout periods may be set for each device. You can set the timeout values either by using the configuration utility or in your program by using the ibtmo function. |
| END (END) | **END or EOS Detected**. If this bit = 1, EOI has been asserted or an EOS byte has been detected. If the GPIB Interface board has been programmed through an ibgts to perform shadow handshaking, any other routine can set this bit to one.<br><br>This bit is cleared (set to 0) whenever an I/O operation is initiated. |
| SRQI | **SRQ Interrupt Received**. If this bit = 1, a device or Controller is requesting service. This bit is set upon the following conditions:<br><br>• The GPIB Interface Board is the Active Controller.<br>• The GPIB SRQ line is asserted.<br>• Automatic serial poll capability is disabled.<br><br>This bit is cleared (= 0) upon the following conditions:<br><br>• The GPIB Interface Board is no longer the Active Controller.<br>• The GPIB SRQ line is unasserted. |

**Table B-1.** Bit Definitions (Continued)

| Bit | Description |
|---|---|
| RQS | **Device Requesting Service**. If this bit = 1, a device is requesting service. RQS is set in the status word whenever the hex 40 bit is asserted in the serial poll status byte of the device. The serial poll that obtained the status byte may have been the result of an `ibrsp` or the poll may have been automatically performed if automatic serial polling is enabled. RQS is cleared when an `ibrsp` reads the serial poll status byte that caused the RQS. An `ibwait` on RQS should only be done on devices that respond to serial polls. |
| CMPL | **I/O Completed**. When set, this bit indicates that all I/O operations have been completed. CMPL is cleared while I/O is in progress. |
| LOK | **Lockout State**. Indicates whether the board is in a lockout state. While LOK is set, the `EnableLocal` routine or `ibloc` function is inoperative for that board. LOK is set whenever the GPIB board detects the Local Lockout (LLO) message has been sent either by the GPIB board or another Controller. LOK is cleared when the Remote Enable (REN) GPIB line becomes unasserted by the system controller. |
| REM | **Remote State**. If this bit = 1, the GPIB Interface Board is in the remote state (i.e, the REN line has been asserted and the Board has been addressed as a listener.). This bit is cleared when one of the following occurs: <br><br> • REN is unasserted. <br><br> • The GPIB Interface Board has been addressed as a Listener and receives a GTL (Go to Local) command. <br><br> • `IBLOC` is called while the LOK bit = 0. |
| CIC | **Controller-In-Charge**. If this bit = 1, it indicates that the GPIB Interface Board is the Active Controller. This bit is set upon the following conditions: <br><br> • If the board is the system controller and `ibsic` or `SendIFC` is executed. <br><br> • Control is passed to the GPIB Interface Board. <br><br> • This bit is cleared whenever the GPIB board detects Interface Clear (IFC) from the system controller, or when the GPIB board passes control to another device. |
| ATN | **Attention**. If this bit = 1, ATN is asserted. Likewise, if this bit = 0, the ATN line is unasserted. |

**Table B-1.** Bit Definitions (Continued)

| Bit | Description |
|-----|-------------|
| TACS | **Talker**. If this bit = 1, the GPIB Interface Board has been addressed as a talker. This bit is cleared when one of the following occurs:<br><br>• The UNT command is sent.<br>• The GPIB Interface Board is sent its listen address.<br>• Another talker is addressed.<br>• IFC is asserted. |
| LACS | **Listener**. If this bit = 1, the GPIB Interface Board has been addressed as a Listener. It can also be set if the GPIB Interface Board shadow handshakes (as a result of an ibgts). This bit is cleared whenever one of the following occurs:<br><br>• The GPIB Interface Board receives an UNL (Unlisten) command.<br>• The GPIB Interface Board is addressed as a talker.<br>• ibgts is called disabling shadow handshaking. |
| DTAS | **Device Trigger Status**. If this bit = 1, a GET (Group Execute Trigger) command has been detected. This bit is cleared in the status word on any call immediately following an ibwait if the DTAS bit is set in the ibwait mask parameter. |
| DCAS | **Device Clear State**. If this bit = 1, a DCL (Device Clear) or SDC (Selected Device Clear) command has been received by the GPIB Interface Board. The bit is cleared on any call immediately following an ibwait call if the DCAS bit was set in the ibwait mask parameter, or on any call immediately following a read or write.<br><br>Some of the status word bits can also be cleared under the following circumstances:<br><br>• If ibonl is called, the END, LOK, REM, CIC, TACS, LACS, DTAS, and DCAS bits are cleared.<br>• If an ENEB or EDVR error is returned (See Appendix C, *IBERR* for more information.), all of the status word bits *except ERR* are cleared. |

# C

# IBERR

If the ERR bit in the status word (ibsta) has been set (= 1), an error code describing the exact error is returned to the iberr variable. Table C-1 lists the possible GPIB error codes, the corresponding mnemonic, and a brief explanation. Note that the mnemonics are defined within the header files for each language. This allows you to use the mnemonic in place of the error code value. Detailed explanations of each error and possible solutions are listed after the table.

**Table C-1.** Error codes

| Error Code | | Description |
|---|---|---|
| **Decimal** | **Mnemonic** | |
| 0 | EDVR | System error |
| 1 | ECIC | Function requires GPIB board to be CIC |
| 2 | ENOL | Write function detected no Listeners |
| 3 | EADR | Interface board not addressed correctly |
| 4 | EARG | Invalid argument to function call |
| 5 | ESAC | Function requires GPIB board to be SAC |
| 6 | EABO | I/O operation aborted |
| 7 | ENEB | Non-existent interface board |
| 10 | EOIP | I/O operation started before previous operation completed |
| 11 | ECAP | No capability for intended operation |
| 12 | EFSO | File system operation error |
| 14 | EBUS | Command error during device call |
| 15 | ESTB | Serial poll status byte lost |
| 16 | ESRQ | SRQ remains asserted |

**Table C-1.** Error codes (Continued)

| Error Code | | Description |
|---|---|---|
| **Decimal** | **Mnemonic** | |
| 20 | ETAB | The return buffer is full. |
| 23 | EHDL | The input handle is invalid |

| EDVR—Driver error | |
|---|---|
| Cause: | A board or device is not installed or configured properly. This error is returned when a device or board that is passed to `ibfind` cannot be found, or when a board index passed to `ibdev` cannot be found. `ibcntl` will contain an error code to help further identify the problem. |
| Solution: | • Call `ibdev` to open a device without using its symbolic name.<br>• Configure each board and device.<br>• Include the unit descriptor that is returned from `ibfind` or `ibdev` as the first parameter in the function. Verify that the value of the variable before the failing function is not corrupted. |

| ECIC—Specified GPIB interface board is not CIC | |
|---|---|
| Cause: | Many routines require that the GPIB interface board is the Controller-In-Charge. These include: `ibcmd`, `ibln`, `ibrpp`, `Send`, `Receive`, for example, as well as any routines which manipulate the GPIB ATN, EOI, or REN lines. |
| Solution: | Make sure that the GPIB interface board is the Controller-In-Charge. |

| **ENOL**—N**o listening device(s)** | |
|---|---|
| Cause: | The routine detected no listeners. |
| Solution: | Verify the following:<br><br>• Make sure that a device has been addressed properly.<br><br>• Check that at least two-thirds of the devices on the GPIB bus are turned on, including the specified device.<br><br>• Make sure that all connections are secure.<br><br>• Verify the device(s) GPIB address.<br><br>• Make sure that the device was properly configured. |

| **EADR—GPIB interface board not addressed** | |
|---|---|
| Cause: | The GPIB interface board has not been addressed. |
| Solution: | Call ibcmd and address the board.<br><br>If you are calling ibgts, with shadow handshaking enabled, call ibcac. |

| **EARG—Invalid argument** | |
|---|---|
| Cause: | An invalid parameter has been provided to a routine. |
| Solution: | Check syntax and range of parameters. See Chapter 3, *GPIB 488.1 Library Reference*, and Chapter 4, *GPIB 488.2 Library Reference*. |

| **ESAC—Board is not the system controller** | |
|---|---|
| Cause: | Routine requires that the GPIB interface board is the system controller. |
| Solution: | Configure the board to be the system controller. |

| EABO—I/O operation aborted | |
|---|---|
| Cause: | I/O operation aborted. (time-out) |
| Solution: | Check that the device is powered on. |
| | Verify proper cable connections. |

| ENEB—Non-existent GPIB board | |
|---|---|
| Cause: | GPIB board is not recognized. |
| Solution: | Make sure that GPIB board settings and configuration setup parameters agree. |

| EOIP—Function not allowed while I/O is in progress | |
|---|---|
| Cause: | An illegal call is made during an asynchronous I/O operation. |
| Solution: | Only `ibstop`, `ibwait`, and `ibonl` calls are allowed during asynchronous I/O operations. The driver must be resynchronized by calling one of the following: |
| | • `ibwait` (mask contains `CMPL`)—The driver and application are synchronized. |
| | • `ibstop`—The asynchronous I/O is canceled, and the driver and application are synchronized. |
| | • `ibonl`—The asynchronous I/O is canceled, the interface is reset, and the driver and application are synchronized. |
| | Re-synchronization is successful if a CMPL is returned to `ibsta`. |

| ECAP—No capability for operation | |
|---|---|
| Cause: | A call attempts to use a capability which is not supported by the GPIB board. |
| Solution: | Don't use the call. |

| **EFSO—File system error** | |
|---|---|
| Cause: | A problem was encountered while performing a file operation. |
| Solution: | Verify the following:<br><br>• Make sure that you specified the correct filename.<br><br>• Check the spelling and the path.<br><br>• If you need more room on the disk, delete some files.<br><br>• Verify that you did not assign a name to a device which is already used by a file. |

| **EBUS—Command byte transfer error** | |
|---|---|
| Cause: | A GPIB bus error occurs during a device function. |
| Solution: | • Determine which device, if any, is abnormally slow to accept commands and fix the problem with the device.<br><br>• Lengthen the assigned time limit using `ibtmo` or `ibconfig`. |

| **ESTB—Serial poll status byte(s) lost** | |
|---|---|
| Cause: | One or more serial poll status bytes received during an automatic serial poll was lost. |
| Solution: | Call `ibrsp` more frequently to read the status bytes. Ignore the ESTB error.<br><br>Note: This error occurs only during `ibrsp` functions. |

| ESRQ—SRQ in "ON" position | |
|---|---|
| Cause: | A wait for an RQS can not occur because the GPIB SRQ line is ON. |
| Solution: | Verify the following:<br><br>• Ignore the ESRQ until all devices are found.<br><br>• Make sure `ibdev` is called to open all devices on the bus capable of asserting SRQ.<br><br>• Test that each device is unasserting SRQ.<br><br>• Check the cabling. Make sure that all devices are attached and the connectors are seated properly. |

| ETAB—Table problem | |
|---|---|
| Cause: | There was a problem with a table used in a `FindLstn` or `FindRQS`. |
| Solution: | • If the error occurs during a `FindLstn`, this is a warning and not an error. Ignore the warning or make the buffer larger.<br><br>• If the error occurs during a `FindRQS`, then none of the specified devices are requesting services. Verify that the device list contains the addresses of all online devices. |

| EHDL—Invalid handle | |
|---|---|
| Cause: | EHDL results when an invalid handle is passed to a function call. |
| Solution: | • Do not use a device descriptor in a board function or vice-versa.<br><br>• Make sure that the board index passed to the call is valid. |

# Index

supported languages (table), 1-1

## T

TestSRQ, 4-26
TestSys, 4-27
Trigger, 4-28
TriggerList, 4-29

## U

utility programs, 1-2

## W

WaitSRQ, 4-30