

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Measurement Computing / ComputerBoards ISA-GPIB* at our website: *Click HERE*

CBI-488.2

SOFTWARE MANUAL

REV 2.0
APRIL 1997

(C) Copyright 1997 ComputerBoards, Inc.

ComputerBoards, Inc.
125 High Street
Mansfield, MA 02040
(508) 261-1123
FAX: (508) 261-1094
www.computerboards.com
E-mail: info@computerboards.com

No part of this manual may be reproduced without written permission.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1 INTRODUCTION 4
1.1 DEFINITION OF TERMS 4
1.2 CONVENTIONS 4
2 GETTING STARTED 6
2.1 OVERVIEW 6
2.2 INSTALLING SOFTWARE 6
2.3 CBCONF - CONFIGURING THE GPIB SOFTWARE 7
2.3.1 Hardware Configuration 9
2.3.2 GPIB Devices, Device Names, and Device Addresses 9
2.3.3 Board Options 10
2.3.3.1 Board Name 10
2.3.3.2 Board is Installed/Not Installed 10
2.3.3.3 Board Type 10
2.3.3.4 Slot Number 11
2.3.3.5 Base Address 11
2.3.3.6 Interrupt Level 11
2.3.3.7 DMA Channel 12
2.3.3.8 Primary GPIB Address 12
2.3.3.9 Secondary GPIB Address 12
2.3.3.10 Timeout Setting 12
2.3.3.11 EOS Byte 12
2.3.3.12 Terminate Read On EOS 13
2.3.3.13 Set EOI With EOS on Write 13
2.3.3.14 Type of Compare on EOS 13
2.3.3.15 Set EOI on Last Byte of Write 13
2.3.3.16 Board is System Controller 13
2.3.3.17 Local Lockout On All Devices 13
2.3.3.18 Disable Auto Serial Polling 14
2.3.3.19 Bus Timing 14
2.3.3.20 Automatically Assert REN When SC 14
2.3.3.21 Parallel Poll Timeout 14
2.3.4 Device Options 14
2.3.4.1 Name Of Device 14
2.3.4.2 Primary GPIB Address 15
2.3.4.3 Secondary GPIB Address 15
2.3.4.5 EOS Byte 15
2.3.4.6 Terminate Read On EOS 15
2.3.4.7 Set EOI With EOS on Write 15
2.3.4.8 Type of Compare on EOS 16
2.3.4.9 Set EOI on Last Byte of Write 16
2.3.4.10 Serial Poll Timeout 16
2.4 CBTEST - TESTING GPIB BOARD 16
2.5 CBIC - TESTING COMMUNICATION WITH GPIB DEVICES 16
2.5.1 Before You Start 16
2.5.2 Running CBIC 17

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.5.3 Testing with 488.2 Commands 17
2.5.3.1 Finding the Listeners on The Bus 17
2.5.3.2 Sending A Command 18
2.5.3.3 On-line Help 19
2.5.3.3 Receiving Data From Device 19
2.5.4 Testing with 488.1 Commands 20
2.5.4.1 Sending A Command with ibwrt 20
2.5.4.1 Receiving Data From Device 20
3 PROGRAMMING WITH THE DRVR-PC2A LIBRARY 22
3.1 GENERAL 22
3.1.1 General Concepts 22
3.1.2 Device vs. Board I/O 22
3.1.3 Device Handles 23
3.1.4 Global Variables 23
3.1.4.1 ibsta - The Status Word 24
3.1.4.2 iberr - The Error Variable 24
3.1.4.3 ibcnt and ibcntl - The Count Variables 24
3.2 EXAMPLES 24
3.2.1 Simple I/O With 488.2 Library 24
3.2.2 Device I/O with Original GPIB Library 25
3.2.3 Board I/O with Original GPIB Library 26
3.2.4 Error Checking 27
3.3 Programming Language Specific Information 29
3.3.1 Developing Programs for Visual Basic for Windows 29
3.3.2 Developing Programs for Visual C++ For Windows 30
3.3.3 Developing Programs in QuickBASIC/BASIC (DOS) 32
3.3.4 Developing GPIB Programs with HP VEE 33
3.3.5 Developing Programs in Professional Basic (DOS) 34
3.3.6 Developing Programs in Microsoft C/Turbo C/Borland 36
4 REFERENCE FOR THE ORIGINAL GPIB LIBRARY 38
(Alphabetical list of functions **ibask - ibwrtia**)
4.1 GENERAL 38
4.1.1 Library Routines 40
5 THE CIO-488.2 ROUTINES 108
(Alphabetical list of functions **AllSpoll - WaitSRQ**)
6 APPENDIX A - MULTILINE INTERFACE MESSAGES 139
7 APPENDIX B - IBSTA 141
8 APPENDIX C - IBERR 144

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1 INTRODUCTION

1.1 DEFINITION OF TERMS

Some of the GPIB terms used within this manual include:

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 an 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/unaddressed by the Active Controller.

Listener — A listener is any device which is able to received 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 any 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.

1.2 CONVENTIONS

The following typographical conventions are used within this manual.

[key] — Used to denote a key on the keyboard which must be pressed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Italics	Used to denote a filename or to denote *notes*, *warnings*, and *cautions*.
underline	Used for emphasis.
bold monospace	Used to denote text or characters which are to be input from keyboard, sections of code, programming examples, and syntax examples.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2 GETTING STARTED

2.1 OVERVIEW

The ComputerBoards GPIB Software consists of:

- A set of programming libraries for all of the most popular PC programming languages including:

  Visual BASIC (Windows)
  MicroSoft Visual C/C++ (Windows)
  Borland C/C++ (Windows)
  QuickBASIC (DOS)
  Visual BASIC (DOS)
  Professional BASIC (DOS)
  MicroSoft C (DOS)
  Borland C (DOS)
  Turbo Pascal, (DOS)

- A set of utility programs, including:

  WCBTEST.EXE - Hardware test program (Windows)
  WINCBIC.EXE - Interactive control program (Windows)
  CBCONF.EXE - Software configuration program
  CBIC.EXE - Interactive control program (DOS)
  CBTEST.EXE - Hardware test program (DOS)

2.2 INSTALLING SOFTWARE

- Windows Installation

  1. Insert GPIB Software Disk #1 into an unused disk drive.

  2. Select Run... from the Start Menu (Win 95)
     OR
     Select Run... from the Program Manager's File menu (Win 3.x)

  3. Enter the following command in the Dialog box:
     x:\setup
     where 'x' is the letter of the drive that the disk was inserted in (a or b)

  4. Follow the instructions on the screen.

- DOS Installation

  1. Insert the GPIB Software Disk into an unused disk drive.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2. Type:
   x:\install (Enter)
   where 'x' is the letter of the drive that the disk was inserted in (a or b)
   where '(Enter)' is the Enter key.

3. Follow the instructions on the screen.

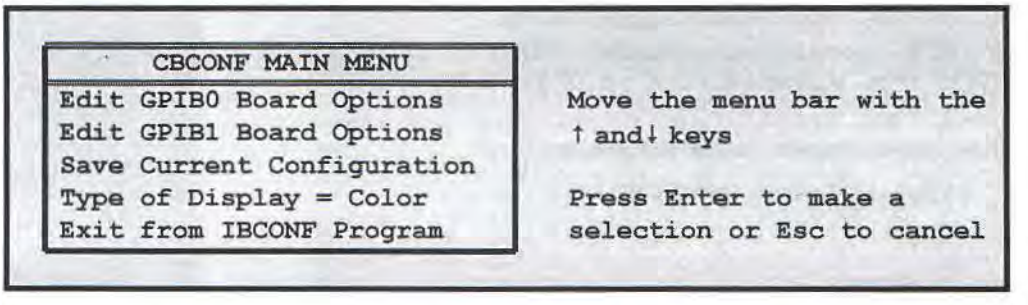
4. Reboot the computer.

2.3 CBCONF - CONFIGURING THE GPIB SOFTWARE

If you are using the DRVR-PC2A routines, you may need to run the *CBCONF* program. This program allows you to change the default configuration parameters for the GPIB Interface Board(s) and the GPIB devices connected to it.

The program reads in the contents of the default configuration file (*GPIB.CFG*) and then displays them. You can then change any of the parameters to meet the needs of your application and save them to the configuration file. This configuration file is read in by the library interface file when your application program is run. If your application program cannot find the *GPIB.CFG* (It looks for it by checking the directory specified in the GPIB.Dir environment variable), it will display an error message.

To run *CBCONF*, click on its icon or type `CBCONF` from the DOS command line prompt. The Main Menu will be displayed:



Use the [↑] and [↓] to highlight your selection. Then, press [Enter]. To cancel your selection, press [Esc].

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you select **GPIB0 Board Options** or **GPIB1 Board Options** from the Main Menu, the appropriate Board Menu will be displayed. For example, if you selected GPIB0, this menu would be displayed:

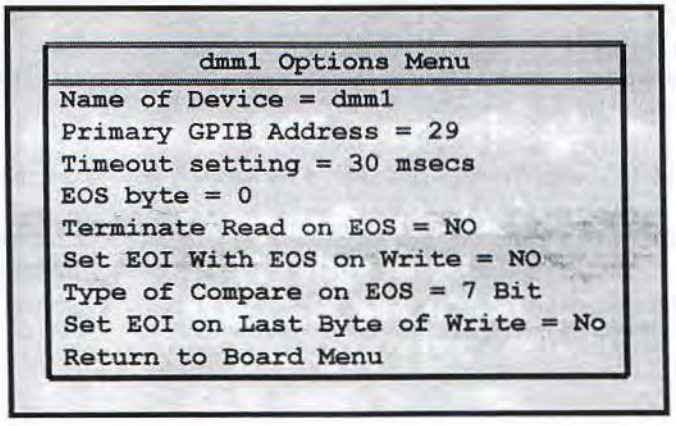


Use the [↑] and [↓] to highlight the device to be modified. Then, press [Enter]. To cancel your selection, press [Esc].

Other keys are used as follows:

[Home]	Move the highlight to the first device in the list.
[End]	Move the highlight to the last device in the list.
[PgUp] [PgDn]	To change between device list pages.

When you select a device to be modified, its menu will be displayed. For example, if you selected DEV1, this menu would be displayed:



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.3.1 Hardware Configuration

Refer to The Installation Manual that came with your GPIB board for instructions about installing the board and setting any jumpers and switches on the board.

2.3.2 GPIB Devices, Device Names, and Device Addresses

The ComputerBoards GPIB library actually contains two quite different and complete GPIB libraries. Each of these libraries is modeled on a corresponding National Instruments library.

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. If you plan to only use this library, you can ignore the section below about Setting Device Names and Addresses.

Original GPIB Library - The original library consists of all of the functions 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 is different.

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

Setting Device Names and Addressees - The CBCONF program lets you assign a name to each device on your GPIB bus. The initial configuration lists 16 devices (DEV1 - DEV16) connected to each GPIB board. Each of these names are associated with a unique GPIB address. "DEV1" is initially set for address 1, "DEV2" = address2, etc. If you like you can leave these names just as they are and your programs can refer to "DEV1" or "DEV2". In that case, you are in effect still referring to the devices by addresses.

A preferable approach is to use CBCONF to give each device a meaningful name that is independent of it's address. For example, imagine that you had two devices on your bus - a voltmeter at address 4 and a signal generator at address 7. You would go to CBCONF's Main Menu and select "Edit GPIB0 Board Options". The existing list of device names (Dev1 - Dev16) appear at the bottom of the GPIB0 Menu. Select "Dev1 Device Options". When the Dev1 Options Menu appears, select "Name of Device" and change it to "VoltMeter". Select "Primary GPIB Address" and change it to 4.

Likewise, you could select Dev2 Device Options and change the name to SigGen and the address to 7.

Once the system is configured with these names you can write a program that communicates with devices called "VoltMeter" and "SigGen". The program will make no reference to any GPIB addresses. If you later change the address of either device you will not have to change the program - you would just rerun CBCONF and change the address that is associated with the device name.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.3.3 Board Options

There are many board options that can be set from the GPIB0 Menu and options associated with each device on the Device Options Menus. In most cases, you will not need to modify any of these options to begin communicating with your GPIB devices.

If you are installing the GPIB board for the first time, we recommend that you check the five Board Options listed below:

Board Type
Base I/O Address
Interrupt Level
DMA Channel

To view or change these options "Edit GPIB0 Board Options" from CBCONF's Main Menu, then select "GPIB0 Board Options"

Depending on which GPIB board you are installing you may have to set one or more of these options. Refer to the board's Installation Manual for details about which of these configuration item(s) must be set for your board.

Each of these Board Options is described in the next section. After checking these five options we recommend that you skip the rest of the Board Options section and proceed directly to section 2.3 CBTEST - Testing GPIB Boards. Once you have finished testing the board and the communication with each of the GPIB devices on your bus, you may want to return to this section to see what other configurable options may be set with CBCONF

2.3.3.1 Board Name

This is the name associated with the GPIB board. This name will be passed to the ibfind routine (board = ibfind ("GPIB0")) whenever you want to do board level I/O. The default names for the two possible GPIB board are GPIB0 and GPIB1. In most cases, there is no advantage to changing the name.

If the system has more than one board installed it may be helpful to change these names to something more meaningful to you.

2.3.3.2 Board is Installed/Not Installed

If you install a second board in your system you must set this option to "Installed" before using the board.

2.3.3.3 Board Type

The ComputerBoards GPIB library supports a number of different types of GPIB boards. The choice selected here must match the board that you have installed. If not, the library will generate an ECFG error (error code = 252).

Refer to the cover of the Installation Manual that came with your board to find out which type of board you have.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you have a PCIIA board there are two possible choices on the Board Type menu "PCIIA (488.1)" and "PCIIA (488.2)". If you have just purchased the board, select "PCIIA (488.2)". If you are updating your software for a PCIIA board that was purchased before 1997 then select "PCIIA (488.1)".

Don't worry if you are not 100% sure of your board type. Make a guess, and the CBTEST program will detect if it is not correct.

2.3.3.4 Slot Number

Some GPIB boards (PCI-GPIB and PCMCIA-GPIB) are configured based on the slot number that they are plugged into. For these boards, this menu item will display the slot number that the board is currently plugged into. This menu item is set automatically by the configuration program. This information can be useful if you have more than one GPIB board installed. It provides a method of differentiating the boards.

2.3.3.5 Base Address

Specifies the base I/O address that the board is configured for. The board must be set for an address that does not conflict with any other board that is installed in the system. If you have more than one GPIB board installed they must be set to different base addresses.

On some boards (ISA, GPIB, ISA-GPIB/LC, PC2A) the base address is set with switches on the board. Refer to the board's Installation Manual for details. Anytime you change those address switches, you must also change this software configuration. The boards come from the factory with these switches in known positions - set to the default base address. In typical installations this default address can be used. In some cases though, you may have to change the address to prevent conflicts with other boards in your system.

> *IMPORTANT*: For proper operation, the switch settings for base address on the 488.2 board must always match the base address setting in CBCONF.

On other boards (PCI-GPIB and PCM-GPIB) the address is chosen automatically by the system and there are no switch settings.

2.3.3.6 Interrupt Level

Specifies the hardware interrupt (IRQ) line that the board is configured for. The board should be configured for an interrupt level that does not conflict with any other board that is installed in the system. If you have more than one GPIB board installed they must be set to different interrupt levels.

On some ComputerBoards GPIB boards the interrupt level is controlled entirely by this selection on CBCONF's Board Options menu.

> *IMPORTANT*: Some boards (PCIIA) have switches and jumpers on the boards that must also be changed, any time you change the Interrupt Level.

Refer to your board's Installation Manual for details about setting the Interrupt Level of your board.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.3.3.7 DMA Channel

Specifies the DMA channel that the board is configured for. The board should be configured for a DMA channel that does not conflict with any other board that is installed in the system. If you have more than one GPIB board installed they must be set to different DMA channel.

Some GPIB Boards do not use DMA. When those Board Types are selected, the DMA Channel option is disabled.

Other GPIB boards (PCIIA and ISA-GPIB/LC) can use a DMA channel for high speed operation. The software comes from the factory set for DMA Channel 3. If you know that another board in your system already uses this DMA Channel then select a different channel or set the DMA Channel=NONE to disable DMA operation.

> *IMPORTANT*: Some boards (PCIIA) have jumpers on the boards that must also be changed, any time you change the DMA Channel.

Refer to your board's Installation Manual for details about setting the DMA Channel of your board.

2.3.3.8 Primary GPIB Address

Each GPIB device must have a unique address. This choice let's you set the board's address (0 - 30). The GPIB board must be set to an address that does not conflict with any other device on the GPIB bus.

2.3.3.9 Secondary GPIB Address

GPIB devices can use extended addressing to increase the number of addresses that can be supported on a single bus. This choice let's you set the board's secondary address. The default setting is NONE, which disables extended addressing for the board.

2.3.3.10 Timeout Setting

The Timeout Setting is the amount of time that each I/O function will wait for data before giving up and returning an EABO error. The default timeout is 10 seconds. In cases where it is expected that an operation will take longer (transfer of a very large mount of data, for example) you may have to increase the Timeout Setting.

2.3.3.11 EOS Byte

There are two methods for devices to indicate the last byte in a transfer of multiple bytes. The most common method is for the sending device to assert the EOI line before sending the last byte. This method is automatically supported by the GPIB Library.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The second method involves sending an extra End-Of-String (EOS) character to indicate the end of data. A common EOS Byte is the linefeed character (10). In most cases, you will not need to set the EOS Byte since most devices rely on the EOI line instead. The default value for the EOS Byte is 0.

2.3.3.12 Terminate Read On EOS

Some devices signal the last byte of a transfer by sending an EOS byte. If you set this option to YES, all read operations will automatically terminate whenever the EOS byte is received. If you set this option to YES, be certain to also set the EOS Byte (described above). The default value for this choice is NO. In most cases, it will never need to be set to YES.

2.3.3.13 Set EOI With EOS on Write

If you set this option to YES, all write operations will automatically assert the EOI line whenever the EOS byte is transmitted. If you set this option to YES, be certain to also set the EOS Byte (described above). The default value for this choice is NO. In most cases, it will never need to be set to YES.

2.3.3.14 Type of Compare on EOS

This option controls whether all checks for the EOS byte use a 7 or 8 bit compare. This option has no affect unless Terminate Read On EOS or Set EOI With EOS on Write is set to YES.

2.3.3.15 Set EOI on Last Byte of Write

When this option is set to YES, the library will automatically assert the EOI line before sending the last byte in a series of bytes. This is the most common method of signaling the end of a transfer. The default for this option is YES and should not be set to NO unless all devices that the board will communicate with are going to use an EOS byte instead.

2.3.3.16 Board is System Controller

There should always by one (and no more than one) System Controller in a GPIB system. Ordinarily the board will be the System Controller, so the default value for this option is YES. If there are multiple boards or multiple computers attached to the GPIB system, than one of them should be configured as System Controller and the others should be set to NO.

2.3.3.17 Local Lockout On All Devices

This option determines whether all GPIB devices will automatically be locked out of local operation whenever they're being programmed remotely by this board. The default setting is NO

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.3.3.18 Disable Auto Serial Polling

This option Enables/Disables automatic serial polls whenever any device asserts the SRQ line. The default choice is YES, which disables the auto serial poll.

2.3.3.19 Bus Timing

This option sets the T1 handshaking delay that the board will use when sending data over the bus. This delay is part of the IEEE 488.2 specification. The default value is 500 ns. If the total length of all GPIB cables in your system is less than 15 meters, than the T1 delay can be set to 350 ns (except on PCIIA boards).

2.3.3.20 Automatically Assert REN When SC

This option controls whether or not the Remote Enable (REN) line is asserted whenever the board is in use. The default choice is YES. If it is set to NO, than the only time REN will be asserted is when it is explicitly set with the ibsre routine.

2.3.3.21 Parallel Poll Timeout

This option controls how long the library will wait when doing a parallel poll before reading the data from the bus. The default setting is 2 usec. In some cases, if you are using a bus extender it may be necessary to increase the delay to 10 usec.

2.3.4 Device Options

There are many device options that can be set from the Device Options Menu. Each of these options is associated with a single device on the bus. These options will only be in affect when doing device level (rather than board level) I/O.

For example:

```
board = ibfind ("GPIB0");
dev = ibfind ("VoltMeter");

ibwrt (board,"ABC",3);      --- Board level options in affect
ibwrt (dev,"ABC",3);        --- VoltMeter devices options in affect
```

2.3.4.1 Name Of Device

This is the name that you wish to associate with a device at a particular address. This name will be passed to the ibfind routine (dev = ibfind ("VoltMeter")) whenever you want to communicate with that device.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.3.4.2 Primary GPIB Address

Each GPIB device must have a unique address. Typically the address of a GPIB device is set by switches on the device. This option lets you specify the address that the device is set for, so that the board can communicate with it.

2.3.4.3 Secondary GPIB Address

Some GPIB devices use extended addressing to increase the number of addresses that can be supported on a single bus. This choice let's you set the device's secondary address. The default setting is NONE, which indicates that the device is not using extended addressing.

2.2.4.4 Timeout Setting

The Timeout Setting is the amount of time that each I/O function will wait for data when communicating with this device. If the device does not respond within the specified period, it gives up and returns an EABO error. The default timeout is 10 seconds. In cases where it is expected that an operation will take longer (transfer of a very large mount of data, for example) you may have to increase the Timeout Setting.

2.3.4.5 EOS Byte

There are two methods for devices to indicate the last byte in a transfer of multiple bytes. The most common method is for the sending device to assert the EOI line before sending the last byte. This method is automatically supported by the GPIB Library.

The second method involves sending an extra End-Of-String (EOS) character to indicate the end of data. A common EOS Byte is the linefeed character (10). In most cases, you will not need to set the EOS Byte since most devices rely on the EOI line instead. The default value for the EOS Byte is 0.

2.3.4.6 Terminate Read On EOS

Some devices signal the last byte of a transfer by sending an EOS byte. If you set this option to YES, all read operations with this device will automatically terminate whenever the EOS byte is received. If you set this option to YES, be certain to also set the EOS Byte (described above). The default value for this choice is NO. In most cases, it will never need to be set to YES.

2.3.4.7 Set EOI With EOS on Write

If you set this option to YES, all write operations with this device will automatically assert the EOI line whenever the EOS byte is transmitted. If you set this option to YES, be certain to also set the EOS Byte (described above). The default value for this choice is NO. In most cases, it will never need to be set to YES.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.3.4.8 Type of Compare on EOS

This option controls whether all checks for the EOS byte use a 7 or 8 bit compare. This option has no affect unless Terminate Read On EOS or Set EOI With EOS on Write is set to YES.

2.3.4.9 Set EOI on Last Byte of Write

When this option is set to YES, the library will automatically assert the EOI line before sending the last byte in a series of bytes. This is the most common method of signaling the end of a transfer. The default for this option is YES and should not be set to NO unless this devices uses an EOS byte instead.

2.3.4.10 Serial Poll Timeout

This option controls how long the library will wait when doing a serial poll before it gives up. The default setting is 1 sec. If the serial poll with this device returns an EABO error, try increasing the Serial Poll Timeout value.

2.4 CBTEST - TESTING GPIB BOARD

CBTEST is a quick test program that verifies that the GPIB board is installed and configured correctly. Run CBTEST and it will display the current hardware configuration (board type, base address, interrupt level, DMA channel) that you selected with the CBCONF program.

CBTEST will check that the software and hardware configurations match and that the hardware is working. If it reports any problems, go back and verify that the switch and jumpers settings on the board (if it has any) are correct and that they match those shown in the CBCONF program.

2.5 CBIC - TESTING COMMUNICATION WITH GPIB DEVICES

CBIC is an interactive program that lets you type GPIB commands and execute them immediately rather than writing a program.

2.5.1 Before You Start

CBIC is the best place to begin before trying to write your first GPIB program. It will let you verify the device addressing, cabling, and proper commands for the instruments that you are using.

Find/Set Device Addresses - By this point, you should know what GPIB address each of your GPIB devices are configured for. If not, check the manuals for those instruments to find out how to set the GPIB address. Set each instrument to a different address. The default address for the GPIB board is 0 so be sure not to set any of the other devices to address 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Plug in GPIB cables - Connect the GPIB board to the device with a standard GPIB cable. Be sure to screw the cables in at either end. If you are connecting more than one device, than "daisy-chain" the cable from the first device to the next one, etc.

Find List of Device's GPIB Commands - Each GPIB device understands its own set of commands. So for example on a Fluke 45 Voltmeter you would send the command "vac" to select the Volts AC range. On a different voltmeter you may have to send a different command to perform the same function. Find the section in the device's User Manual that lists all of its GPIB commands.

2.5.2 Running CBIC

If you are using Windows, just click on the CBIC icon. In DOS, run CBIC by typing

cbic <Enter>

at the DOS command prompt.

In either case, the program will display a copyright message followed by a ":" prompt.

Now you can execute all of the routines in the GPIB Library interactively be typing them at the prompt. Refer to the reference section in the back of this manual for the details on each of the routines in the library.

The next few sections below are a simple tutorial that will get you started. They will show you how to use the GPIB Library and the CBIC program. Along the way they may also identify a few of the common installation/configuration problems.

Please take the few minutes necessary to run through these quick tests.

2.5.3 Testing with 488.2 Commands

At the ":" prompt, type the following command

: ibfind gpib0

The prompt should change to "GPIB0"

2.5.3.1 Finding the Listeners on The Bus

If you are using a 488.1 (older PCIIA board) skip to the next section.

Enter the FindLstn command at the GPIB0 prompt. This routine will search the GPIB bus for all listeners that are present. It should print a list containing the addresses of all devices that you have attached to the GPIB board.

```
GPIB0:  FindLstn
```

the program will respond with the following prompt

Type each address followed by Enter
Press Enter again at end of list

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Enter each GPIB address at which you expect to find a device. For example, if you have three instruments attached to the GPIB bus and you beleive that they are at addresses 3,4, and 5 you would respond:

```
Enter address list: 3
Enter address list: 4
Enter address list: 5
Enter address list:
```

The FindLstn function will then check the bus and see if any instruments responds at the specified addresses.

Responding Devices:

```
Device at address = 3
Device at address = 5
```

In this example, two devices were found - one at GPIB address 3 and one at address 5.

When you type FindLstn, check the addresses that are returned and make sure that they match the addresses that you believe your devices are set for. If not, check the address settings on those devices.

If the FindLstn returned an ECAP error, it means that you are using a PCIIA board which does not support the FindLstn routine, so proceed to the next section below.

2.5.3.2 Sending A Command

Refer to the list of GPIB commands that your device understands. Choose a command that should cause some visible response on the front panel of the device. That will vary depending on the instrument. Oftentimes a command that selects a range will have a visible indicator.

Use the GPIB Library's Send routine to send the command to the device. The Send routine in the GPIB library has four arguments. When calling any of the library routines from within CBIC, the program always sends the first argument (board or dev) automatically. The three other arguments for Send are address, data, and eotmode.

address should be set to the device's GPIB address .
data is the command string you wish to send
eotmode should be set to DABend

There are two ways to execute a GPIB Library routine within CBIC. You can type in the name of the routine and CBIC will prompt you for each of the arguments. You can also type them in all on one line, separated by spaces.

Here's an example of each method:

```
GPIB0:  Send
     Enter address of device :  7
     Enter string to be written : vac
     Enter EOTMode (DABend, NULLend, or NLend) :  DABend

GPIB0: Send 7 vac DABend
```

Using either method, the command "vac" will be sent to the device. On a Fluke45 voltmeter the "vac" command will select the Volts AC range and the front panel indicator on the voltmeter will switch to "VAC"

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.5.3.3 On-line Help

If you forget the arguments for a command you can use CBIC's help function. Rather than typing a command at the prompt, type "help" and the name of the GPIB Library routine. For example:

```
GPIB0: help send
```

For more general help just type the word "help" by itself. You can then choose from a variety of topics.

2.5.3.3 Receiving Data From Device

Refer to the list of GPIB commands that your device understands. Find a command that will return data. It will vary depending on the instrument. On many instruments you send a command to the instrument (with the Send routine) and it will send back a response which you can read with the GPIB Library's Receive routine.

For example, if you send the Fluke 45 Voltmeter the command "VAL?" it will take a measurement and than send the result back over the GPIB bus. This sequence would be executed via CBIC like this:

```
GPIB0: Send 7 val? DABend
GPIB0:  Receive 7 100 STOPend
Data:  -1.345e1
```

In the example the arguments for the Receive routine are:

7 - the address of the meter
100 - maximum number of characters to read
STOPend - causes Receive to stop reading when it receives the End (EOI) signal

-1.345e1 - is the reading returned by the meter.

Some GPIB instruments work a bit differently. They always return the most recent measurement whenever you call the Receive routine, and don't require that you ask for each measurement by Send'ing a command.

For example, the Fluke 8840 meter works this way.

```
GPIB0:  Receive 7 100 STOPend
Data:  -1.345e1
```

To read it again, you could type the same Receive command or better yet, use CBIC's "!" command. The "!" command executes the most recent command again.

```
GPIB0:  !
Data:  -1.347e1

GPIB0:  !
Data:  -1.422e1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.5.4 Testing with 488.1 Commands

In the examples above, you were using the 488.2 GPIB Library routines. The 488.2 Library always passes a GPIB address or a list of addresses as an argument to the routines.

The 488.1 routines can be a bit simpler to use because they use the device information that was configured with the CBCONF program. So for example if you configured a device called voltmeter with the CBCONF program, you would begin communicating with the device like this.

```
:  ibfind voltmeter
voltmeter:
```

The prompt would change to "voltmeter:" to indicate that it is now the current device.

The most common type of error that can occur at this point is that the device could not be found and the following message would be printed on the screen:

Cannot find a configured device named voltmeter.

There are two possible reasons. The simplest is that you typed the name incorrectly. The other possibility is that you have not yet configured a device called "voltmeter" with the CBCONF program. Run CBCONF and check the name that you gave the device. See section 2.2.2 for details.

2.5.4.1 Sending A Command with ibwrt

The ibwrt routine in the 488.1 library performs the same function as the Send routine in the 488.2 library. They both send a command to a GPIB device.

ibwrt is simpler to use because it has fewer arguments. Two of Send's arguments - the device's GPIB address (address), and the termination method (EOTmode) are not required when calling ibwrt. Instead they are automatically read from "voltmeter" configuration information that was set in the CBCONF program.

Here's how you would send the command "vac" to the device called "voltmeter".

```
voltmeter:  ibwrt vac
```

2.5.4.1 Receiving Data From Device

Receiving data with the ibrd routine is also a bit simpler than with the 488.2 library's Receive routine. Once again, fewer arguments need to be passed because it uses the configuration information in GPIB.CFG that was set up by the CBCONF program.

Here's how you would read a measurement from a Fluke 45 meter:

```
GPIB0: ibwrt val?
GPIB0:  ibrd 100
Data:  -1.345e1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

At this point, you can begin exploring the operation of your GPIB devices. With CBIC you can quickly experiment with each of your devices commands using ibrd and ibwrt (or Send and Receive) and get a good idea of what they do. You can also explore all of the other routines in both the Original Library and the 488.2 Library just by typing them in at the CBIC prompt.

As your proceed, refer to Chapter 4 or CBIC's on-line help, for additional details about the GPIB Library routines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3 PROGRAMMING WITH THE DRVR-PC2A LIBRARY

3.1 GENERAL

The DRVR-PC2A library is comprised of many different routines. These routines allow you to control all of the operations of the GPIB bus in a very specific manner. You may find the number of routines included in the DRVR-PC2A library intimidating. In most applications, you will need to use only a small subset of these routines.

The routinues are divided into two distinct libraries. All routinues which begin with IB are part of the "Original GPIB Library." All other routinues 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. Refer to section 2.2.2 and 2.4.2 for more info about choosing which library to use. If you choose the original GPIB Library then there is another choice to be made. You can use the orginal library routinues to do either Board Level or Device Level operations.

The remaining sections of this chapter explain the differences between routines which use device I/O and board I/O, device and board handles, and explain the global variables used within the interface. Other sections give an example program and provide language-specific information.

3.1.1 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 DRVR-PC2A routines.

3.1.2 Device vs. Board I/O

The most typical GPIB operations are sending commands to a device attached to the bus and reading back responses. In order to accomplish this, you must program the CIO-PCIIA board to execute the following 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. Send the GPIB Unlisten (UNL) message.
6. 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. All of these routines are described in detail in Chapter 4. You can program the board using either the provided Board I/O routines or Device I/O routines to perform the sequence of operations outlined above.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Device I/O

In general, it is almost always easier to use the Device I/O routines. Device I/O is very simple to use and understand. These routines are higher-level routines which conceal most of the underlying complexity of GPIB operations. These commands make use of the device names you assigned using the CBCONF program. 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 six steps outlines above, you need to use only two routines:

1. **IBFIND** --- To open the device.
2. **IBWRT** --- To send the command.

Board I/O

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

For example, to perform the same task as the six steps outlined above (send a command to a device), you will need to know the codes for the various forms of addressing and the codes for the GPIB Unlisten and Untalk commands. Then, you can use the routines in this sequence:

1. **IBFIND** --- To open the board.
2. **IBCMD** --- To send the address of the talker and listener.
3. **IBWRT** --- To send the command to the device.
4. **ICBMD** --- To send the Unlisten (UNL) and Untalk (UNT) commands.

3.1.3 Device Handles

All of the routines in the CIO-488 library have a device handle as the first argument. The first GPIB call in your program is usually an **IBFIND**. This routine "opens" a board or device and returns a GPIB board or device handle. If you pass the name of a board (as assigned by CBCONF), then it will return a board handle. Likewise, if a device name is passed, a device handle is returned. Some of the library routines work only with device handles, some only with board handles, and some with both.

A note regarding board/device names. These names are assigned by the configuration program, CBCONF. This program is described in detail in Chapter 2. It is very important that you run this program and make necessary changes prior to writing your application program.

3.1.4 Global Variables

The following global variables are used in all programming languages:

- `ibsta` - Status Word
- `iberr` - Error Codes
- `ibcnt,ibcntl` - Count Variables

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

This section describes these variables briefly. Appendices A and B provide detailed descriptions of ibsta and iberr. More specific information regarding the count variables are found in the descriptions of the routines which return them.

3.1.4.1 `ibsta` - The Status Word

Every routine returns a status word (**`ibsta`**) which indicates the results of the last operation. Your application program can test each bit of this word and then take appropriate action. The **`ibsta`** variable is discussed further in Appendix B.

3.1.4.2 `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 given in Appendix A.

3.1.4.3 `ibcnt` and `ibcntl` - The Count Variables

These variables will 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).

3.2 EXAMPLES

This section provides three example programs. The first example uses the 488.2 Library. The following two examples illustrate Device Level I/O and Board Level I/O with the orginal library. One uses Device I/O and the other uses Board I/O. You may also find it useful to consult the example programs provided on the release disks and/or to run the *CBIC* program provided with the CIO-PC2A driver software. This program allows you to execute specific DRVR-PC2A routines using your system configuration. This will help you anticipate what types of responses you will obtain.

3.2.1 Simple I/O With 488.2 Library

This example illustrates a simple GPIB application. It sends command strings to a Fluke45 Multimeter to initialize it, select a measurement range and take a measurement. It reads the measurement back from the meter and prints it on the screen. It uses the following commands which are specific to the Fluke 45.

* RST - Reset the meter
VDC - Select Volts DC Range
VAL? - Take a measurement and send it over the GPIB bus

The program assumes that the Fluke45 is set for GPIB address 3.

BASIC

```
const  METER_ADR = 3
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
const BOARD_NUM = 0

buffer$ = space$(100)
CALL                          ' Open the GPIB board
DevClear (BOARD_NUM%, METER_ADR)
Send (BOARD_NUM%, METER_ADR, "*RST", DABend)
Send (BOARD_NUM%, METER_ADR, "VDC",DABend)
Send (BOARD_NUM%, METER_ADR, "VAL?", DABend)
Receive (BOARD_NUM%, METER_ADR, buffer$, 100, STOPend)
PRINT "Current voltage = ";buffer$
```

C

```
#define METER_ADR  3
#define BOARD_NUM 0

 int Board;
char buffer[100];

                                        // Open the board
DevClear (BOARD_NUM, METER_ADR);        // Clear the meter
Send (BOARD_NUM, METER_ADR, "*RST", DABend);
Send (BOARD_NUM, METER_ADR, "VDC", DABend);
Send (BOARD_NUM, METER_ADR, "VAL?", DABend);
Receive (BOARD_NUM, METER_ADR, buffer, STOPend);
printf ("Current voltage = %s\n", buffer);
```

3.2.2 Device I/O with Original GPIB Library

This example performs the same operations as the example above. Instead of using the 488.2 Library it uses the original GPIB Library.

The following examples show a simplified program (no error checking) for reading a DC voltage measurement from the volt meter. The program assumes that the name "VoltMeter" has been assigned to the Fluke 45 (via the CBCONF.EXE program).

BASIC

```
buffer$ = space$(100)
CALL ibfind ("VoltMeter", device%)              ' Open device
CALL ibclr (device%)                            ' Clear the device
CALL ibwrt (device%, "*RST")                    ' Reset meter
CALL ibwrt (device%, "VDC")                     ' Select volts DC range
CALL ibwrt (device%, "VAL?")                ' Ask for current value
CALL ibrd  (device%, buffer$) ' Read current value
PRINT "current voltage = "; buffer$;
```

C

```
int  device;
char buffer[100];
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
device = ibfind ("Volt Meter");  /* Open device */
ibclr (device);                  /* Clear the device
ibwrt (device, "*RST");          /* Reset meter */
ibwrt (device, "VDC");           /* Select volts DC range */
ibwrt (device, "VAL?");          /* Ask for current value */ ibrd  (de-
vice, buffer,100);      /* Read current value */
printf ("current voltage = %s\n" buffer);
```

3.2.3 Board I/O with Original GPIB Library

Board I/O requires a more detailed understanding of the GPIB bus and commands. Your program must explicitly send all of the GPIB command codes to explicitly set up each GPIB operation. For example, you must set up the addressing for talker and listener etc.

The example below performs the same operations as the Device I/O example, using Board I/O routines. Note that not only do you need to be familiar with the device-specific commands, but also the GPIB Multiline Interface Messages (e.g., MLA, MTA etc.). The program assumes that the Fluke 45 meter is at GPIB address 13.

BASIC

```
CALL ibfind ("GPIB", board%)              ' Open board
CALL ibsic (board%)                       ' Send IFC message
CALL ibsre (board%,1)                     ' Turn on REN
command$ = chr$(&H11) + "-" + chr$(&H14) +"@"
CALL ibcmd (board%,command$)              ' LL0, MLA13, DCL, MTA0
CALL ibwrt (board%, "*RST")               ' Reset meter
CALL ibwrt (board%, "VDC")                ' Select volts DC range CALL ibwrt
CALL ibwrt(board%, "VAL?")               ' Ask for current value
command$ = "?" + "_" + "m" + " "
CALL ibcmd (board%, command$)             ' UNL, UNT, MTA13, MLA0
buffer$ = space$(100)
CALL ibrd  (board%, buffer$)             ' Read current value
PRINT "current voltage = "; buffer$;
```

C

```
int  board;
char buffer[100], *command;
board = ibfind ("GPIB");                 ' Open board
ibsic (board%);                          ' Send IFC message
ibsre (board%,1);                        ' Turn on REN
command = "\0x11\0x2d\0x14\0x40";
ibcmd (board%, command, 4);              ' LL0, MLA13, DCL, MTA0
ibwrt (board%, "*RST", 4);               ' Reset meter
ibwrt (board%, "VDC", 3);                ' Select volts DC range
ibwrt (board%, "VAL?",4);                ' Ask for current value
command = "\0x3f\0x5f\0x4d\0x20";
ibcmd (board%, command, 4);              ' UNL, UNT, MTA13, MLA0
ibrd  (board%, buffer);                  ' Read current value
printf "current voltage = %s\n" buffer);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.2.4 Error Checking

The above examples do not include any error checking. In a real program you should always check for error after each call to the library. The easiest way to do this is to write one error checking/handling routine and call it after every call to the library.

The error reporting mechanism involves three global variables - **ibsta, iberr** and **ibcnt**. If an error occurred during a call to the library then the EERR bit in **ibsta** will be set (to 1) and iberr will contain an error code. If the error code indicates that a DOS error occurred then ibcnt will contain the DOS error code.

The examples below illustrate the use of error handling routines in different languages.

BASIC

```
CALL ibwrt (device%, "*RST")      ' After each call to GPIB
Call CheckGPIBError               ' check for errors

SUB CheckGPIBError ()
   IF (ibsta% AND EERR) <> 0 THEN
      PRINT "*** ERROR -";
      SELECT CASE iberr%
         CASE EDVR
             PRINT "EDVR=DOS Error"
         CASE ECIC
             PRINT "ECIC=Board is not Controller In Charge"
         CASE ENOL
             PRINT "ENOL=No listener"
         CASE EADR
             PRINT "EADR=Address Error"
         CASE EARG
             PRINT "EARG=Invalid argument passed to library"
         CASE ESAC
             PRINT "ESAC=Board is not system controller"
         CASE EABO
             PRINT "EABO=I/O Operation terminated"
         CASE ENEB
             PRINT "ENEB=No GPIB board installed"
         CASE EOIP
             PRINT "EOIP=Background I/O already in progress"
         CASE ECAP
             PRINT "ECAP=Board missing required capability"
         CASE EFSO
             PRINT "EFSO=File system error"
         CASE EBUS
             PRINT "EBUS=Command error"
         CASE ESTB
             PRINT "ESTB=Status byte lost"
         CASE ESRQ
             PRINT "ESRQ=SRQ line is stuck on"
         CASE ETAB
             PRINT "ETAB=Table overflow"
      END SELECT
      IF iberr% == EDVR THEN PRINT "DOS Error Code=";ibcnt%
      IF ibsta% AND TIMO THEN PRINT "GPIB device timed out"
   END IF
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
END SUB
```

C

```
ibwrt (device, "*RST", 4);    /* After each call to GPIB */
CheckGPIBError();             /* check for errors */

void
CheckGPIBError (void)
{
   if (ibsta & EERR)
      {
      printf ("*** ERROR -");
      switch (iberr)
         {
         case EDVR
            printf ("EDVR=DOS Error\n");
         case ECIC
            printf ("ECIC=Board is not Controller In Charge\n");
         case ENOL
            printf ("ENOL=No listener\n");
         case EADR
            printf ("EADR=Address Error\n");
         case EARG
            printf ("EARG=Invalid argument passed to library\n");
         case ESAC
            printf ("ESAC=Board is not system controller\n");
         case EABO
            printf ("EABO=I/O Operation terminated\n");
         case ENEB
            printf ("ENEB=No GPIB board installed\n");
         case EOIP
            printf ("EOIP=Background I/O already in progress\n");
         case ECAP
            printf ("ECAP=Board missing required capability\n");
         case EFSO
            printf ("EFSO=File system error\n");
         case EBUS
            printf ("EBUS=Command error\n");
         case ESTB
            printf ("ESTB=Status byte lost\n");
         case ESRQ
            printf ("ESRQ=SRQ line is stuck on\n");
         case ETAB
            printf ("ETAB=Table overflow\n"););
         }
      if (iberr == EDVR)
         printf ("DOS Error Code=%d\n", ibcnt);
      if (ibsta% & TIMO)
         printf ("GPIB device timed out\n");
      }
   return;
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.3 Programming Language Specific Information

The GPIB Library is nearly identical in each of the supported languages. Each of the functions is described in detail in Chapter 4. The descriptions there apply to all supported programming languages. This section deals with the small amount of details that are specific to each programming language.

3.3.1 Developing Programs for Visual Basic for Windows

The Visual BASIC interface combines support for both the 16 bit and 32 bit versions of Visual BASIC. If you are running Windows 3.x than you are using the 16 bit VB. If you are running Windows 95 then you may be running either the 16 or 32 bit version. Use VB's "Help","About Microsft Visual BASIC" menu choice to find out which it is.

The files that make up the Visual BASIC for Windows interface are:

- *GPIB.DLL* - The GPIB Library
- *GPIB-16.DLL* - 16 bit "thunk" layer - only used by Visual BASIC (32 bit)
- *GPIB-32.DLL* - 32 bit "thunk" layer - only used by Visual BASIC (32 bit)
- *GPIB-16.BAS* - The Visual BASIC (16 bit version) header file
- *GPIB-32.BAS* - Visual BASIC (32 bit version) header file - compatible with National Instruments NI-488.2 Library
- *GPIB-32M.BAS* - Visual BASIC (32 bit version) header file - compatible with National Instruments NI-488.2M Library
- *GPIBVXD.386* - Device driver, is loaded by \windows\system.ini file

Sample Programs

The example programs are in the \cb_gpib\win_vb directory
vb16demo.mak - for 16 bit version of VB
vb32demo.mak - for 32 bit version of VB

Versions Supported

Visual BASIC 3.0 and higher

Header File

All VB programs that use the GPIB library must also include either GPIB-16.BAS, GPIB-32.BAS, or GPIB-32M.BAS file in the project. To add the file to a project, select "File","Add File" from VB's menus.

You should select the appropriate header file depending on which version of VB you're using, and which version of the National Instruments GPIB library that you wish to be compatible with. Refer to the list above.

National Instruments has two different versions of their GPIB Library (NI 488.2 and NI-488.2M) that have slight incompatibilities between them. We have included two different header files (GPIB-32.BAS and GPIB-32M.BAS) that compatible with each version of the National Instruments Library.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Special Programming Notes

Calling GPIB Library Via Windows Timers - The GPIB Library can only execute one function call at a time. Due to the nature of Windows and VB it is quite easy to write a program that will attempt to execute more then library call at a time, if you use VB's timers.

For example, imagine that you have a timer routine that reads a meter every second by sending an IBWRT followed by an IBRD and you also have say a button on a panel that sends a command to the meter - say a range change command.

When the timer code calls a GPIB function, the library may yield to Windows while its waiting for the GPIB bus. At that point, if the user presses the front panel button, another call into the GPIB library will occur before the first one has completed. This would cause all sorts of hard to diagnose problems since bytes from the two different commands would get intermixed on the GPIB bus. To prevent this problem, the library will stop the second call from executing, and return an ESML error.

To prevent these ESML error from ocurring you must prevent simultaneous calls from being executed. The easiest way to do this is with a "semaphore variable". The general idea is that you have a Global variable that your program sets whenever it is about to call a GPIB function. After it returns from the GPIB Library it clears the variable. All code that use the GPIB Library must check the variable before calling the library and only call it if it is clear.

Refer to the VBxxDEMO program for an example . Look at the use of the GPIBCallInProgress variable in the program.

When programming with BASIC, there are two versions of all Original Library routines - A subroutine and a function. The subroutines all begin with the letters "ib" (e.g. ibclr) and the functions begin with the letters "il" (e.g. ilclr). The only difference between them is how they are called from a BASIC program.

```
status% = ilclr (0)
OR
CALL ibclr 0
```

Compatability With National Instruments Library

The ComputerBoards Library is fully source code compatible with the National Instruments Library. To use existing National Instruments programs with the ComputerBoards Library just swap the header files.

All VB programs that run with the National Instruments Library include two header files - NIGLOBAL.BAS and VBIB-32.BAS. These files should be deleted from the project and the appropriate ComputerBoards header file (GPIB-16.BAS, GPIB-32.BAS or GPIB-32M.BAS) must be added.

3.3.2 Developing Programs for Visual C++ For Windows

The Visual C++ interface combines support for both the 16 bit and 32 bit versions of Visual C++.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The files are used by the Visual C++ interface are:

- *GPIB.DLL* - The GPIB Library (DLL)
- *GPIB.LIB* - The GPIB Library's 16 bit import library
- *GPIB-16.DLL* - 16 bit "thunk" layer - only used by Visual BASIC (32 bit)
- *GPIB-32.DLL* - 32 bit "thunk" layer - only used by Visual BASIC (32 bit)
- *GPIB-32.LIB* - The GPIB Library's 32 bit import library
- *WINDECL.BAS* - The Visual C++ header file
- *GPIBVXD.386* - Device driver, is loaded by \windows\system.ini file

Sample Programs

An example program is in the \cb_gpib\win_c directory

wincdemo.mak

Versions Supported

Visual C++ for Windows - all versions

Header File

All C or C++ programs that use the GPIB library must include the WINDECL.H file. Add the following line to the start of your program.

#include "windecl.h"

Adding Library to Project

Each program that uses the ComputerBoards GPIB library must be linked to the appropriate Import Library file.
There are two import libraries provided.
GPIB.LIB - for 16 bit Visual C++
GPIB-32.LIB - for 32 bit Visual C++

The details of how to do this are different for different versions of Visual C++.

Visual C++ 4.0 - First open or create a Visual C++ Project Workspace. Then from the VC++ menus select "View", then "Project Workspace". On the left side of the main Window you will see Project Workspace info. Click on the "FileView" tab at the bottom of that pane. Now the top of that pane should show "projectname files". Next, from the main menu select "Insert", "File into Project", go to the \cbi_gpib\win_c directory and select the GPIB-32.LIB.

Visual C++ 1.5x - First open or create a Visual C++ Project. Then from the VC++ menus select "Project", then "Edit Project". Change the "List Files of Type" box to "Library *.lib". Goto the \cbi_gpib\win_c directory and select the GPIB.LIB file.

Compatibility with National Instruments Library

The ComputerBoards Library is fully source code compatible with the National Instruments Library. To use existing National Instruments programs with the ComputerBoards Library just swap the header files and libraries.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

All VC++ programs that run with the National Instruments Library include a header file - DECL-16.H or DECL-32.H. The "#include DECL-xx.H" in the C source code should be replaced with an include of the ComputerBoards header file as described above.

All VC++ programs that run with the National Instruments NI-488.2M Library also include an OBJ file (GPIB-32.OBJ) in their project. Follow the instructions above for adding the ComputerBoards Library but before adding it, delete the National Instruments OBJ file from the project.

3.3.3 Developing Programs in QuickBASIC/BASIC (DOS)

These files form the QuickBASIC/BASIC Interface:

- *GPIBQB.QLB* A library of CIO-488 routines to be used within a QuickBASIC development environment.
- *GPIBQB.LIB* A library file which is linked to compiled programs to create stand-alone executable files.
- *DECL.BAS* BASIC header file.

Versions Supported

QuickBASIC 4.0 and Higher

Preparing the Environment

Before you begin to write your programs, copy the following files from the CIO-PC2A Driver Disk 2 to your working directory.

GPIBQB.QLB
GPIBQB.LIB
GPIB.CFG
DECL.BAS

Header File

Be sure to include the following line within your program:

```
'$INCLUDE: 'decl.bas'
```

The inclusion of this file allows QuickBASIC/BASIC to check that the correct number and type of parameters are specified for each routine called.

Using the GPIB Library from Within QuickBASIC

To Load the DRVR-PC2A Library
Be sure that the file *GPIBQB.QLB* is located where QuickBASIC can find it. Then, invoke QuickBASIC by typing:

```
qb /Lgpibqb yourprog
```

where ***yourprog*** is the name of your program.

Once you have entered the QuickBASIC environment, compile the program as you normally would.

Compliling with the

This process compiles the BASIC source code and links it to the BASIC and

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Command Line Compiler

CIO-PC2A library files. Compile the code by typing:

```
bc /o /d yourprog.bas
link yourprog,,,bcom45+gpibqb;
```

where:

`yourprog` is the name of your program.

`bcom45` is the QuickBASIC Runtime library name.

`gpbiqb` is the linkable BASIC library file.

Refer also to the *MAKEQB.BAT* file in the \QBASIC directory on Disk 2 of the CIO-PC2A Driver Disks.

Special Programming Notes

1. Parameters can be passed as either variables or constants. For example:

```
DevName$ = "VoltMeter"
CALL IBFIND (DevName$, Dev%)
```

is the same as:

```
CALL IBFIND ("VoltMeter", Dev%)
```

2. When programming with BASIC, there are two versions of all Original Library routines - A subroutine and a function. The subroutines all begin with the letters "ib" (e.g. ibclr) and the functions begin with the letters "il" (e.g. ilclr). The only difference between them is how they are called from a BASIC program.

```
status% = ilclr (0)
OR
CALL ibclr 0
```

Sample Program

SAMPLE.BAS illustrates the use of several of the DRVR-PC2A routines.

3.3.4 Developing GPIB Programs with HP VEE

HP VEE is a graphic programming environment from Hewlett Packard that will work with the ComputerBoards GPIB Library. HP Vee comes with 100's of Instrument drivers that greatly simplify the programming of most common GPIB instruments.

Enabling ComputerBoards GPIB Support Within HP VEE

HP VEE automatically recognizes that the ComputerBoards GPIB library has been installed on your system. However, if you also have a National Instruments GPIB library on your system, HP VEE may recognize the wrong library. In that case, you should rename or remove the following National Instruments files from your system.

```
\windows\system\gpib.dll
\windows\system\gpib-16.dll
\windows\system\gpib-32.dll
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Earlier than HP Ver 4.0

If you are using a version of HP VEE that is less than 4.0 the GPIB support will automatically be enabled. The GPIB board will be installed as GPIB interface 14. All GPIB devices within HP VEE have a 4 digit address property - 14xx, where xx is the GPIB address of the device. For example if you have a signal generator set to GPIB address 7, then the address field for this device will be set to 1407. This address (1407) should be used throughout VEE whenever you want to access that device.

Ver 4.00

If you are using a version 4.0 (or greater) of HP VEE than you must first copy the CompuetrBoards GPIB DLL files from the \cbi_gpib directory into the \windows\system directory.

Run HP Vee. Select I/O from VEE's main menu bar, then select Instrument Manager. The Instrument Manager dialog box will be displayed. Press the "refresh" button on the right. The panel on the left will be updated with a new driver called "HP-IB0". This is the ComputerBoards GPIB Library.

Select "HP-IB0" on the left panel, then press the "Edit.." button. A small "Interface Configuration" dialog box will appear. Change the address field to 14, then selct OK to close the dialog.

The "HP-IB0" in the left pane will have changed to "HP-IB14". Select "HP-IB14" and press the refresh button again. All devices that are connect to the GPIB boards should appear beneath the "HP-IB14" icon in the left pane. Each of the device icons will have a "?" superimposed on it. This indicates that the system does not yet know what type of device it is.

Select one of the unknown devices and press "refresh" again. A small dialog box will appear that says "OK to Send '*IDN?' to NewDevice. If you select OK, the system will send this command to the instrument. "*IDN?" is a standard (SCPI) command which instructs the device to send back an identification string. Many (most?) GPIB instruments respond to this command and the system will automatically recognize which type of device it is.

Select one of the devices and press the "Edit" button. The address field should be set to a 4 digit number - 14xx, where xx is the GPIB address of the device. For example if you have a signal generator set to GPIB address 7, then the address field for this device will be set to 1407. This address (1407) should be used throughout VEE whenever you want to access that device.

3.3.5 Developing Programs in Professional Basic (DOS)

These files form the Professional BASIC Interface:

- *GPIBPB.QLB* — A library of CIO-488 routines to be used within a Professional BASIC development environment.
- *GPIBPB.LIB* — A library file which is linked to compiled programs to create stand-alone executable files.
- *DECL.BAS* — BASIC header file.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Versions Supported

Professional BASIC 4.0 and higher

Preparing the Environment

Before you begin to write your programs, copy the following files from the CIO-PC2A Driver Disk 2 to your working directory:

GPIBQB.QLB
GPIBQB.LIB
GPIB.CFG
DECL.BAS

Header File

Be sure to include the following line within your program:

```
'$INCLUDE:'decl.bas'
```

The inclusion of this file allows Professional BASIC to check that the correct number and type of parameters are specified for each routine called.

Compliling Your Program

This process compiles the BASIC source code and links it to the BASIC and CIO-PC2A library files. Compile the code by typing:

```
bc /Fs yourprog.bas
link yourprog,,,gpibpb;
```

where:

`yourprog` is the name of your program.
`gpbiqb` is the linkable BASIC library file.

Refer also to the *MAKEPB.BAT* file in the \PBASIC directory on Disk 2 of the CIO-PC2A Driver Disks.

Special Programming Notes

1. Parameters can be passed as either variables or constants. For example:

```
DevName$ = "VoltMeter"
CALL IBFIND (DevName$, Dev%)
```

is the same as:

```
CALL IBFIND ("VoltMeter", Dev%)
```

2. When programming with BASIC, there are two versions of all Original Library routines - A subroutine and a function. The subroutines all begin with the letters "ib" (e.g. ibclr) and the functions begin with the letters "il" (e.g. ilclr). The only difference between them is how they are called from a BASIC program.

   status% = ilclr (0)
   OR
   CALL ibclr 0

Sample Program

SAMPLE.BAS illustrates the use of several of the CIO-PC2A routines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.3.6 Developing Programs in Microsoft C/Turbo C/Borland C/C++ (DOS)

There are several files which comprise this interface:

- *GPIBCL.LIB* Library File for Microsoft C Large Model.
- *GPIBCM.LIB* Library File for Microsoft C Medium Model.
- *GPIBCS.LIB* Library File for Microsoft C Small Model.
- *DECL.H* C Header file.

Versions Supported

Microsoft C version 3.0 and later

Preparing the Environment

Before you begin to write your programs, copy the following files to your working directory:

GPIBCS.LIB
GPIBCM.LIB
GPIBCL.LIB
DECL.H
GPIB.CFG

Header File

When you write your program, make sure to include the line:

```
#include <decl.h>
```

The DECL.H file is a header file which defines all of the DRVR-PC2A Routines. In particular, it defines the type and number of parameters associated with each routine.

Compliling Your Program

When the source code is compiled, it is linked to the appropriate *GPIB.LIB* file. This process forms a stand-alone executable (.exe) file.

Compile your program in the normal manner, being sure to link it with the correct library. For example, at the DOS prompt, you might type either:

```
cl yourprog.c /link gpibcs.lib
```

or

```
cl /c yourprog.c;
link yourprog,,,gpibcs/lib;
```

Refer also to the *MAKEMC.BAT* (Microsoft C) and *MAKETC.BAT* (Turbo C) files .

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Special Programming Notes

1. Parameters which are not pointers to int's or unsigned's may be passed as constants rather than variables.

2. It is very important that the number of bytes allocated for storage within a character array is at least one greater than the maximum byte count passed into the routine. This extra byte is necessary so to accommodate the NULL which marks the end of the received data. If a routine attempts to receive more bytes than have been allocated for storage into that variable, your program may crash.

3. When programming with BASIC, there are two versions of all Original Library routines - A subroutine and a function. The subroutines all begin with the letters "ib" (e.g. ibclr) and the functions begin with the letters "il" (e.g. ilclr). The only difference between them is how they are called from a BASIC program.

   ```
   status% = ilclr (0)
   OR
   CALL ibclr 0
   ```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4 REFERENCE FOR THE ORIGINAL GPIB LIBRARY

4.1 GENERAL

This chapter describes each of the original GPIB library routines. Each description provides a short description of the routine, its syntax, parameters, and any values which are returned. Any special usage notes and an example are also given. The routines are listed in alphabetical order. Table 4-1 lists all of the original GPIB library routines.

NAME	DESCRIPTION
IBASK	Returns software configuration information
IBBNA	Change access board of device
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
IBEVENT	Returns oldest recorded event
IBFIND	Open a device and return its unit descriptor
IBGTS	Go from Active Controller to standby
IBINIT	Reinitializes library, reloads software configuration
IBIST	Define IST bit
IBLINES	Return status of GPIB bus lines
IBLN	Check fpr presence of device on bus
IBLOC	Got to Local
IBONL	Place device online/offline
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
IBRDI	Read data to integer array
IBRDIA	Read data asynchronously to integer array
IBRPP	Conduct parallel poll
IBRSC	Request/release system control
IBRSP	Return serial poll byte
IRSRQ	Install an SRQ interrupt routine
IBRSV	Request service
IBSAD	Define secondary address
IBSIC	Send IFC
IBSRE	Set/clear REN line
IBSTOP	Stop asynchronous I/O operation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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
IBWRTI	Write data from integer array
IBWRTIA	Write data asynchronously

Table 4-1. The DRVR-PC2A Routines

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4.1.1 Library Routines

IBASK

Purpose Returns software configuration information.

Syntax

BASIC
```
CALL ibask (boarddev%, option%, value%)
```

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

PASCAL
```
ibask (boarddev:integer; option:integer; var value: integer)
```

Parameters

`boarddev` a board handle or device handle

`option` specifies which configuration item to return (see table below)

`value` current value of specified item returned here

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 list of possible vlaues
IbaEOT	bd/dev	0 = EOI asserted at end of write non zero = EOI is not asserted at end of write
IbaAutoPoll	bd	0 = automatic at end of write non zero = automatic serial poll is disabled
IbaIRQ	bd	IRQ used for interrupts or 0 if disabled
IbaSC	bd	0 = board is not system controller non zero = board is system controller
IbaSRE	bd	0 = do not automatically assert REN line when system controller non zero = automatically assert REN line when system controller
IbaEOSrd	bd/dev	0 = ignore EOS char during reads non zero = terminate read when EOS char is received
IbaEOSwrt	bd/dev	0 = don't assert EOI line when EOS char is sent non zero = assert EOI line whenever EOS char is sent
IbaEOScmp	bd/dev	0 = 7 bit compare is used when checking for EOS char non zero = 8 bit compare is usede when checking for EOS char
IbaEOSchar	bd/dev	0 = The current EOS char of board or device
IbaTiming	bd	Current T1 timing delay 1=Normal (2us), 2=High Speed (500ns), 3=Very High Speed (350 ns)
IbaDMA	bd	Low byte = DMA channel, hex 8000 bit indicates DMA enabled Examples: 0x003 = DMA disabled, set for DMA chan 3 0x8001 = DMA enabled, set for DMA chan 1
IbaSPollTime	dev	Length of time to wait for parallel poll response before timing out

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IbaPPollTime	bd	Length of time to wait for parallel port response
IbaBNA	dev	Device's access board
IbaBaseAddr	bd	Base I/O address of GPIB board
IbaBoardType	bd	Type of GPIB board installed 1 = PCIIA board with NEC 7210 chip 2 = PCIIA board with CBI 488.2 chip 3 = ISA-GPIB/lc Board 4 = ISA-GPIB Board 5 = PCM-GPIB Board 6 = PCI-GPIB Board 7 = PC104-GPIB Board
IbaChipType	bd	Type of GPIB Controller Chip on board 0 = NEC 7210 1 = CBI - 7210.2
IbaSlotNum	bd	Slot number that board is installed in
IbaPCIIndex	bd	PCI bus index for board
IbaBaseAdr2	bd	Address of data register (PCI-GPIB only)

Returns

ibsta will contain 16 bit status word described in Appendix B

iberr will contatin an error code if an error occurred

value will contain the current value of selected configuration item

Usage Notes

1. Some options apply to boards and some apply to both boards and devs.

2. The current value of each configuration item is initially controlled by the CBCONF.EXE program. A program may modify many of these configuration items via library routines (e.g. ibtmo, ibeos, etc.). In that case, ibask() will return the modified version. To restore the original configuration information as specified by CBCONF.EXE, call the ibinit() routine.

Example

This example returns the primary address of the device named "VoltMeter"...

BASIC

```
CALL ibfind ("VoltMeter", dev%)
CALL ibask (dev%, IbaPAD, address%)
```

C

```
int dev, address;
dev = ibfind ("Voltmeter");
ibask (dev, IbaPAD, &address);
```

PASCAL

```
var
   dev := integer;
   address := integer;
begin
   dev := ibfind ("Voltmeter");
   ibask (dev, IbaPAD, address);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBBNA

Purpose

Allows you to re-assign a device to another GPIB interface board

Syntax

BASIC
```
CALL ibbna (device%, boardname$)
```

C
```
ibbna (int device, char boardname[])
```

PASCAL
```
ibbna (device:integer; varboardname:string)
```

Parameters

`device` is an integer containing the device handle

`boardname` is the name of the new board to be assigned to the device

Returns

`ibsta` will contain a 16-bit status word as described in Appendix B

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

Usage Notes

1. This routine changes the board assignment made during the configuration process. The assigned board will be used for all subsequent operations involving the specified device until:

   - **IBBNA** is called again
   - **IBONL** or **IBFIND** is called
   - OR the sytem is restarted

Example

This example uses **IBFIND** to return the unit descriptor 5 ("DEV 5"), a DMM, into the variable `dmm`. The DMM is then affiliated with the GPIB Interface Board "GPIB1."

BASIC
```
CALL ibfind("DEV5"), dmn%)
CALL ibbna (dmn%, "GPIB1")
```

C
```
int dmn;
dmm = ibfind ("DEV5");
ibbna (dmm, "GPIB1");
```

PASCAL
```
var
 dmm := integer;
begin
 dmm :=ibfind('DEV5');
 ibbna (dmm, 'GPIB1');
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBCAC

Purpose

Makes the specified board the Active Controller.

Syntax

BASIC
```
CALL ibcac (board%, sync%)
```

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

PASCAL
```
ibcac (board:integer; sync:integer)
```

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 will take control synchronously. Otherwise, it will take control asynchronously (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.

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

Usage Notes

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

Example

In this example, GPIB board 1 takes control asynchronously.

BASIC
```
CALL ibcac (brd1%, 0)
```

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

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBCLR

Purpose

Clears a specified device.

Syntax

BASIC
```
CALL ibclr (device%)
```

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

PASCAL
```
ibclr (device: integer)
```

Parameters

`device` is an integer containing the device handle.

Returns

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

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

Usage Notes

1. 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 **IBFIND** to return the unit descriptor for device 5 ("DEV5"), a DMM, into the variable `dmm`. The DMM is then cleared.

BASIC
```
CALL iBfind ("dev5",dmm%)
CALL ibclr(dmm%)
```

C
```
int dmm;
dmm = ibfind("dev5");
  /*open instrument*/
ibclr (dmm);
  /* clear it */
```

PASCAL
```
var
  dmm:integer;
begin
  dmm := ibfind ('dev5');
  ibclr (dmm);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBCMD

Purpose

Sends GPIB commands.

Syntax

BASIC
```
CALL ibcmd (board%, cmnd$, bytecount%)
```

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

PASCAL
```
ibcmd (board:integer; command:string; bytecount:
word)
```

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.

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

Returns

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

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

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

Usage Notes

1. 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.

2. 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.

3. If your application must perform other functions while processing the GPIB command(s), use IBCMDA.

Example

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

BASIC
```
command$ = "?_@()*" 'UNL UNT MTA0 MLA8 MLA9 MLA10
CALL ibcmd (board%, command$)
```

C
```
char  *command;
command = "\x3f\x5f\x40\x28\x20\x2a"
ibcmd (board, command,6);

var
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
PASCAL      board:integer;
            command: string;
            bytecount:word;
          begin
            command := '\x3f\x5f\x40\x28\x20\x2a'
            ibcmd (board, command,6);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBCMDA

Purpose

Transfers GPIB commands asynchronously from a string.

Syntax

BASIC `CALL ibcmda (board%, cmnd$)`

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

PASCAL `ibcmda (board:integer; command: string; bytecount:word)`

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.

`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.

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

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

Usage Notes

1. This routine is seldom used in typical GPIB applications. The only reason to use asynchronous commands is to free up the machine during lengthy operations. GPIB commands are rarely longer then a half dozen bytes; thus, in most cases, you will be able to use **IBCMD** instead of **IBCMDA**.

2. This routine passes only commands. It cannot be used to transmit programming instructions (data) to devices. Use the **IBRDA** and **IBWRTA** routines for this purpose.

3. 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.

4. The specified GPIB Interface Board will remain the Active Controller of the GPIB bus after this routine has completed, even if it was not the Active Controller before the routine was called.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5. Remember that once an asynchronous I/O call has begun, any further calls to a DRVR-PC2A routine (except for **IBWAIT** and those functions which do not specify a device or GPIB Interface Board)cannot be executed until the I/O has finished and the GPIB driver and the application have been re-synchronized. This is accomplished 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 has been reset, and the driver and application are synchronized.

Re-synchronization is successful if a CMPL is returned to **ibsta**. See Appendix ? for further information.

Example

This example prepares the board to talk and addresses three devices (at addresses 8, 9, and 10) to listen. IBCMDA will execute in the background and the program will continue 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 else within the WHILE loop except make other GPIB I/O calls.

BASIC

```
command$ = "?_@()*"     'UNL UNT MTA0 MLA8 MLA9
MLA10
CALL ibcmda (board%, command$)
WHILE (ibsta% AND CMPL+EERR) = 0
   ' Check for complete or error
   CALL ibwait (board%, 0)
   ' This updates ibsta
WEND.
```

C

```
char  *command;
command = "\x3f\x5f\x40\x28\x20\x2a"
ibcmda (board, command,6);
   while ( (ibsta & CMPL+EERR) == 0)
   ibwait (board, 0)
```

PASCAL

```
var
   command:string;
begin
   command '\x3f\x5f\x40\x28\x20\x2a'
   ibcmda (board, command,6);
       while ( (ibsta & CMPL+EERR) == 0)
                ibwait (board, 0)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBCONFIG

Purpose Changes configuration parameters.

Syntax

BASIC
```
CALL ibconfig (boarddev%, option%, value%)
```

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

PASCAL
```
ibconfig (boarddev:integer; option:integer;
value:integer)
```

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.

`value` is the new configuration option value. Allowed values differ according to the option being programmed. See Table 4-2.

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 from 0, or from 96 to 126 decimal, (See IBSAD.)
IbcTMO	bd or dev	**Timeout Value**. This is the approximate length of time that I/O functions can take before a timeout occurs. `value` is a number from 0 to 15 which corresponds to timeout value 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 will be asserted when the last byte of data is sent. If `value` is 0, then the EOI line will not be asserted. Otherwise, it will be.
IbcPPC		**Parallel Poll Configure.** Redefines the parallel poll configuration bytes. `value` can be from 0, or from 96 to 126 decimal, (See IBPPC) See **IBBPC** for more information.
IbcAUTOPOLL	bd	**Enable/Disable Automatic Serial Polling**. If `value` is 0, then Automatic Serial Polling is disabled. Otherwise, it will be enabled.
IbcCICPROT		**CIC Protocol**. If `value` is 0, then CIC Protocol will not be used. Otherwise, it will be.
IbcIRQ	bd	**Enable/Disable Hardware Interrupts**. If `value` is 0, then hardware interrupts will not be disabled, otherwise value specifies the IRQ level the board will use to generate interrupts.
IbcSC	bd	**Request/Release System Control.** If `value` is 0, the board will not be 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 will be asserted.
IbcEOSrd	bd or dev	**Recognize EOS.** If `value` is non-zero, a read will be 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 will be asserted when the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

		EOS character is sent. Otherwise, EOI will not be 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
IbcPPL		**Parallel Poll Remote/Local.** If **value** is zero, then the GPIB Interface Board will be remotely configured for a parallel poll by an external Controller. Otherwise, the GPIB Interface Board will accept parallel poll configuration commands from your application program.
IbcTIMING	bd	**Handshake Timing.** If **value** is 1, normal timing (> 2 *sec.) will be used. If **value** is 2, high-speed timing (> 500 nsec.) will be used. If **value** is 3, very high-speed timing (> 350 nsec.) will be used.
IbcDMA	bd	**Enable/Disable DMA**. If **value** is zero, DMA transfers are disabled, otherwise value specifies the DMA channel that the board will use.
IbcBaseAddress	bd	Base address that board is configured for
Ibc Board Type	bd	Type of GPIB Board being used. Value = 1 PCIIA with NEC 7210 2 PCIIA board with CBI 488.2 chip 3 ISA-GPIB/lc 4 ISA-GPIB 5 PCM-GPIB 6 PCI-GPIB 7 PC104-GPIB
IbcSPollTime	bd or dev	**Serial Poll Timeout.** Values from D-15 specify timeouts of 100μsecs to 100secs. Refer to Table 4.4 in IBTMO description.
IbcPPollTime	bd	**Parallel Poll Timeout.** Values from D-15 specify timeouts of 100μsecs to 100secs. Refer to Table 4.4 in IBTMO description.

Table 4-2. Configuration Options

Returns

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

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.

BASIC

```
CALL ibfind ("gpib1", device%)
CALL ibconfig (device%, IbcTMO, 10)
```

C

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

PASCAL

```
var
 device:integer;
begin
  device := ibfind ('gpib1');
  ibconfig (device, IbcTMO, 10);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBDEV

Purpose

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

Syntax

BASIC

```
CALL ibdev (board.index%, pad%, sad%, timeout%,
eot%, eos%, device%)
```

C

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

PASCAL

```
device := Integer:
(boardindex:Integer;pad:Integer;sad:Integer,
timeout:Integer: eot:Integer;eos:Integer)
```

Parameters

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

`pad` is the primary address of the device. Available primary addresses range from 0 to 30. `value` can be from 0 to 30 decimal. (See Appendix A.)

`sad` is the secondary address of the device. There are 31 secondary addresses available. `value` can be from 0, 96 to 126 decimal, See Appendix A. Note that if 09 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 15 which corresponds to timeout value ranging from 10 *sec to 100 sec. See Table 4-4 in the IBTMO description, or a listing of timeouts and corresponding values.

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

`eos` specifies the End-Of-String termination mode to be used when communicating with the device.

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:

1. An EDVR or ENEB error will be returned if a device is not available or the board index specifies a non-existant board.

2. If illegal values are given for `pad, sad, timeout, eot, eos`

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

Usage Notes

1. This routine returns the device handle of the first available user-configurable device it finds in the device list. (See the IBCONF program description for more information.) Be sure to open all of your devices (with **IBFIND**) you plan to use before calling

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBDEV. This will guarantee that **IBDEV** will not open a device that you plan open with **IBFIND**.

2. It is very important that you disable the device descriptor (use **IBONL**) when you have finished all operations associated with that device. Otherwise, the device descriptor will not be available for further use with **IBDEV**.

Example

This example opens an available device, associates it with GPIB Interface Board 1, and assigns it various device configuration parameters. These are:

- primary address = 3
- secondary address = 19 (115 decimal, 73 hex)
- timeout = 10 sec
- Assert EOI.
- EOS Disabled

The new device handle is returned.

BASIC

```
CALL ibdev(1,3,&H73,13,1,0, device%)
```

C

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

PASCAL

```
var
 device: integer;
begin
 device := ibdev(1,ibdev(1,3,0x73,13,1,0);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBDMA

Purpose

Enables/Disables DMA.

Syntax

BASIC
```
CALL ibdma (board%, dma%)
```

C
```
ibdma (int board, int dma)
```

PASCAL
```
ibdma (board:integer; dma:integer)
```

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, then all read and write operations between the GPIB board and memory will be performed using DMA. Otherwise, programmed I/O will be used.

Returns

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

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

Usage Notes

1. The GPIB Interface Board must have been configured for DMA operations in order for this routine to executed successfully. (Refer to the *IBCONF* program.)

2. This routine is useful for alternating between programmed I/O and DMA operations.

3. 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.

Example

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

BASIC
```
CALL ibfind ("gpib1", board%)
CALL ibdma (board%, 1)
```

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

PASCAL
```
var
 board:integer;
begin
 board := ibfind ('gpib1');
 ibsta := ibdma(device, 1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBEOS

Purpose

Changes or disables End-Of-String termination mode.

Syntax

BASIC `CALL ibeos (boarddev%, eos%)`

C `ibeos (int boarddev, int eos)`

PASCAL `ibeos (boarddev:Integer; eos:Integer)`

Parameters

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

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

		eos	
Method	**Description**	**High Byte**	**Low Byte**
A	Terminate read when EOS is detected. Can be used alone or in combination with Method C.	0000100	EOS
B	Set EOI with EOS on write function. Can be used alone or in combination with Method C.	00001000	EOS
C	Compare all 8 bits of EOS byte rather that low 7 bits (all read and write functions.)	00010000	EOS

Table 4-3. Selecting EOS

Returns

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

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

Usage Notes

1. 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.

2. 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 used that EOS.

3. This call remains in effect until one of the following occurs:
   - Another **IBDMA** call is made.
   - **IBONL** or **IBFIND** is called.
   - The system is re-started.

4. See also **IBEOT**.

5. To set the EOS byte permanently, use the CBCONF program. Then, you need only call **IBEOS** when you want to change the default EOS byte.

Example

This example configures the GPIB system to send the END message whenever the line feed character is sent to a particular device.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC

```
CALL ibeos (device%, XEOS + chr$(&H0A) )
```

C

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

PASCAL

```
var
 device:integer;
begin
 ibeos (device, XEOS + '\n');
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBEOT

Purpose

Enables/Disables assertion of EOI on write operations.

Syntax

BASIC
```
CALL ibeot (boarddev%, eot%)
```

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

PASCAL
```
ibeot (boarddev: integer, eot: integer)
```

Parameters

`boarddev` is an integer containing the unit descriptor. In this case 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 will be asserted automatically when the last byte of the message is sent. Otherwise, if `eot` is 0, then EOI is not asserted.

Returns

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

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

Usage Notes

1. This call is used to temporarily change the default EOT setting.
2. 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.
3. If this call specifies a device, then EOI will be asserted/unasserted on all writes to that device. Likewise, if the call specifies a board, then EOI will be asserted/unasserted on all writes from that board.
4. If you want to assert EOI with an EOS, use the **IBEOS** routine.
5. 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.
6. See also **IBEOS**.

Example

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

BASIC
```
CALL ibfind ("gpib1", device)
CALL ibeot (device%, 1)
```

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

PASCAL
```
var
 device:integer;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
begin
 device := ibfind ('gpib1');
 ibeot (device, 1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBEVENT

Purpose

Returns the oldest recorded event

Syntax

BASIC
```
CALL ibevent (board%, event%)
```

C
```
ibevent (short board, unsigned short *event)
```

Pascal
```
ibevent (board:integer; var event:integer)
```

Parameters

`board` a board handle

`event` the event is returned here

Returns

`ibsta` will contain 16 bit status word described in Appendix B

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

`ibcnt` will contain the number of events remaining in queue

`event` will contain the code for the oldest event in queue

Usage Notes

1. Usually you call ibevent() when the EVENT bit is set in ibsta

Example

This example returns the next event in the board's queue

BASIC
```
CALL ibfind ("GPIB0", board%)
IF (ibsta% AND EVENT) <> 0 THEN
     CALL ibevent (board%, event%)
END IF
```

C
```
int board, event;
board = ibfind ("GPIB0");
if (ibsta & EVENT)
     ibevent(board, &event);
```

PASCAL
```
var
     board := integer;
     event := integer;
begin
     board := ibfind ("GPIB0");
     if (ibsta and EVENT) <> 0 then
          ibevent (board, event);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBFIND

Purpose

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

Syntax

BASIC
```
CALL ibfind (name$, boarddev%)
```

C
```
boarddev := int ibfind (char name[])
```

PASCAL
```
ibfind (var name:nbuf):Integer
```

Parameters

name is a string specifying the board or device name. It must match the configured board or device name specified in CBCONF.

Returns

device 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.

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

Usage Notes

1. This call is also opens the device/board and initializes the software parameters to their default configuration settings. (See **IBONL**.)

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.

BASIC
```
dmm% = ilfind ("DEV5")
IF dmm% <0 GOTO 1000
```

C
```
dmm = ibfind("DEV5");
if (dmm<0) error();
```

PASCAL
```
dmm := ibfind('DEV5');
if (dmm<0) error();
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBGTS

Purpose

Puts an Active Controller in Standby mode.

Syntax

BASIC
```
CALL ibgts (board%, handshake%)
```

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

PASCAL
```
ibgts (board:integer; handshake : integer)
```

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.

`iberr` will contain an error code, if an error occurred. The ECIC error will occur, 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, i.e., "?Z+".

BASIC
```
CALL ibfind ("GPIB1", gpib1%)
CALL ibcmd (gpib1%, "?Z+", 3)
CALL ibgts(gpib1%, 1)
```

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

PASCAL
```
var
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
 gpib1: integer;
begin
 gpib1 := ibfind ('GPIB1');
 ibcmd (gpib1, '?Z+', 3);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBINIT

Purpose

Re-initialize board, reload configuration information

Syntax

BASIC
```
CALL ibinit (board%)
```

C
```
ibinit (short board)
```

PASCAL
```
ibinit (board:integer)
```

Parameters

`board` a board handle

Returns

`ibsta` will contain 16 bit status word described in Appendix B

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

`event` will contain the code for the oldest event in queue

Usage Notes

1. This routine is useful if you run CBCONF.EXE to change the configuration while a GPIB program is running. If the program calls ibinit () it will load the new configuration.

2. This routine can also work around a potentially confusing problem when using the library from Windows. The Windows library is a DLL which is shared between all programs that use it. When the DLL is first called, the system loads it into memory, and the DLL loads configuration information. Normally, when the GPIB program terminates the DLL is unloaded. If the program is run again the whole process is repeated. If more than one program is using the DLL, or when running from some programming environments, the DLL may not be unloaded when the program terminates. In that case, a call to ibinit () at the start of the program will force the configuration info to be reloaded.

Example

This example reloads the configuration information

BASIC
```
CALL ibfind ("GPIB 0", board%)
CALL ibinit (board%)
```

C
```
int board;
board = ibfind ("GPIB0");
ibinit (board);
```

PASCAL
```
var
     board := integer;
     listen:integer
begin
     board := ibfind ('GPIB0');
     ibln (board, 4, NO_SAD, listen);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Purpose

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

Syntax

BASIC
```
CALL ibist(board%,statusbit%)
```

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

PASCAL
```
ibist (board:integer; statusbit:integer)
```

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.

`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

1. This routine is used when the specified GPIB Interface is not the Active Controller.
2. IST should be SET to indicate to the controller that service is required.

Example

This example clears GPIB Board 1's IST bit.

BASIC
```
gpib1% = ilfind ("GPIB1")
ibsta% = ilist (gpib1%,0)
```

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

PASCAL
```
var
 gpib: integer;
begin
 gpib1 = ibfind ('GPIB1');
 ibist (gpib1, 0);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBLINES

Purpose

Returns the status of the GPIB control lines.

Syntax

BASIC `CALL iblines (board%, clines%)`

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

PASCAL `ibline (board:integer; clines:word)`

Parameters

`board` is an integer containing the board handle.

Returns

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

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

To determine if a GPIB control line is asserted, first check the appropriate bit in the lower byte to dtermine 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.

`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 Identtify

Usage Notes

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

2. Different ComputerBoards GPIB boards have more or less ability to monitor individual GPIB control lines. The mask bits in clines (bits 0-7) indicate which lines the installed board is capable of monitoring.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example This example tests the state of the ATN line.

BASIC

```
CONST ATNline% = &h40
CALL iblines (board%, lines%)
IF lines% AND ATNline% = 0 THEN
  PRINT "ATN line cannot be monitored by this
      GPIB board"
ELSE IF (lines% / 256) AND ATNline% = 0 THEN
  PRINT "ATN line is not asserted"
ELSE
  PRINT "ATN line is asserted"
END IF
```

C

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

PASCAL

```
const
      ATNLINE = $40
var
      lines:integer;
begin
      iblines(board,lines);
      if lines and ATNLINE = 0 then
            writeln ('ATN line can be monitored
            by this GPIB board');
         else if (lines/256) and ATNLINE=0 then
            writeln ('ATN line is not asserted');
         else
            writeln ('ATN line is asserted');
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBLN

Purpose

Check that a device is present on the bus.

Syntax

BASIC
```
CALL ibln (board%, pad%, sad%, listen%)
```

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

PASCAL
```
ibln (board:integer, pad:integer, sad:integer, var listen:integer)
```

Parameters

`board` a board or device handle

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

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

`listen` the variable that results is returned to

Returns

`ibsta` will contain 16 bit status word described in Appendix B

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

`listen` will contain 0 if no listener is found

will contain non-zero if listener is found

Usage Notes

1. Set sad = NO_SAD if the device does not have a secondary address
2. Set sad = ALL_SAD if you do not know the devices secondary address and you want all possible secondary address to be tested

Example

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

BASIC
```
CALL ibfind ("GPIB0", board%)
CALL ibln (board%, 4, NO_SAD, listen%)
```

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

PASCAL
```
var
board := integer;
listen:integer
begin
board := ibfind ('GPIB0');
ibln (board, 4, NO_SAD, listen);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBLOC

Purpose

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

Syntax

BASIC
```
CALL ibloc (boarddev%)
```

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

PASCAL
```
ibloc (boarddev:integer)
```

Parameters

`boarddev` is an integer containing the device handle.

Returns

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

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

Usage Notes

1. This routine is used to place boards or devices temporarily in local mode.

2. 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 local 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.

BASIC
```
CALL ibfind ("GPIB1",gpib1%)
CALL ibloc (gpib1%)
```

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

PASCAL
```
var
gpib : integer;
begin
gpib1 := ibfind('GPIB1');
ibloc (gpib1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBONL

Purpose

Enables/Disables a device/interface board for operation.

Syntax

BASIC
```
CALL ibonl(boarddev%, online%)
```

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

PASCAL
```
ibonl (boarddev:integer; online:integer)
```

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, then the device/board will be enabled for operation (placed on-line). This restores the board/device to its default settings. Otherwise, the board/device will be placed off-line.

Returns

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

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

Usage Notes

1. When a device is placed off-line, it is in essence "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 device 0.

BASIC
```
CALL ibfind ("DEV0", dev0%)
CALL ibonl (dev0%, 1)
```

C
```
int dev0;
dev0 = ibfind ("DEV0");
ibonl (dev0, 1);
```

PASCAL
```
var
dev0 : integer;
begin
dev0 := ibfind ('DEV0');
ibonl (dev0 , 1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBPAD

Purpose

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

Syntax

BASIC
```
CALL ibpad (boarddev%, address%)
```

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

PASCAL
```
ibpad(boarddev:Integer; address:Integer)
```

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.

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

Usage Notes

1. 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.

2. If a device is specified, its talk and listen addresses will be 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 will be assigned the primary addressed defined by `address`.

3. Refer also to **IBSAD** and **IBONL**.

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

Example

This example changes the primary GPIB address associated with DVM ("DEV4") to 1C hex.

BASIC
```
CALL ibfind ("DEV4", dvm%)
CALL ibpad (dvm%, &H1C)
```

C
```
int dvm;
dvm = ibfind ("dev4");
ibpad (dvm%, 0x1C);
```

PASCAL
```
var
 dvm : integer;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
begin
 dvm := ibfind ('dev4');
 ibpad (dvm, 0x1C);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBPCT

Purpose

Passes control to another device.

Syntax

BASIC
```
CALL ibpct (device%)
```

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

PASCAL
```
ibpct (device:integer)
```

Parameters

`device` is an integer containing the device handle.

Returns

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

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

Usage Notes

1. This makes the specified device the Controller-In-Charge (CIC). The GPIB board goes to the Controller Idle state and releases the ATW line.
2. The device that control is passed to must have Controller capability.

Example

This example makes Device ABCD the Controller-In-Charge.

BASIC
```
CALL ibfind ("ABCD", abcd%)
CALL ibpct(abcd%)
```

C
```
int abcd;
abcd = ibfind ("ABCD");
ibpct(abcd);
```

PASCAL
```
var
 abcd : integer;
begin
 abcd := ibfind ('ABCD');
 ibpct (abcd);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBPPC

Purpose Enables/Disables parallel polling of the specified device.

Syntax

BASIC `CALL ibppc (boarddev%, command%)`

C `ibppc (int boarddev, int command)`

PASCAL `ibppc(boarddev:integer; command:integer)`

Parameters `boarddev` is the device's 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 below:

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
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, i.e.:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

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

Usage Notes

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

2. 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 specifed device as a Listener, and sends the PPC command followed by a PPE or PPD command.

Example

This example configures device 29 to send DIO7 true if its IST bit = 1.

BASIC

```
dev29% = ilfind ("DEV29")
ibsta% = ilppc (dev29%, &H6B)
```

C

```
int dev29;
dev29 = ibfind ("DEV29");
ibppc (dev29, 0x6B);
```

PASCAL

```
var
 dev29 : integer;
begin
 dev29 := ibfind ('DEV29');
 ibppc (dev29, 0x6B);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRD

Purpose

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

Syntax

BASIC

```
CALL ibrd (boarddev%, buf$)
```

C

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

PASCAL

```
ibrd (boarddev:Integer, var Buf; ByteCount:word)
```

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.

Returns

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

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

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR will result if the specified GPIB Interface Board is an Active Controller but has not been addressed to listen. An EABO error will result 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 64K, use `ibcntl` instead of `ibcnt`.

Usage Notes

1. 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.

2. 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 DEV5.

BASIC

```
dev5% = ilfind ("DEV5")
rd$ = space$(90)
ibsta% = ilrd (DEV5%, rd$)
```

C

```
int dev5;
char rd [90];
dev5 = ibfind ("DEV5");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
ibrd (dev5, rd, 90);
```

PASCAL

```
var
 dev5: integer;

begin
 dev5 := ibfind ('DEV5');
 ibrd (dev5, rd, 90);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRDA

Purpose

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

Syntax

BASIC `CALL ibrda (boarddev%, buf$)`

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

PASCAL `ibrda (boarddev:Integer; var buf; byteCount:Word)`

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 langugage 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.

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR will result if the specified GPIB Interface Board is an Active Controller but has not been addressed to listen. An EABO error will result 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 64K, use ibcntl instead of `ibcnt`

Usage Notes

1. 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.

2. If `boarddev` specifies a device, the specified device is addressed to talk and its associateded 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.

3. Remember that once an asynchronous I/O call has begun, any further calls to a CIO-488 routine (except for IBWAIT and those functions which do not specify a device or GPIB Interface Board) cannot be executed until the I/O has finished and the GPIB driver and the application have been re-synchronized. This is accomplished 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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

	application are synchronized.
IBONL	The asynchronous I/O is canceled, the interface has been reset, and the driver and application are synchronized

Re-synchronization is successful if a CMPL is returned to **ibsta**. See Appendix C for further information.

This command is normally only used for very large data transfers. It allows the program to continue running in the foreground while the data transfer continues in the background. Use **IBRD** for small transfers.

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 an error has occurred. The program may do anything else within the WHILE loop except make other GPIB I/O calls.

BASIC

```
readbuffer = space(5000)
CALL ibwrt (device%, "DUMP")
CALL ibrda (device%, readbuffer$)
WHILE (ibsta% AND CMPL+EERR) = 0
 ' Check for complete or error
    ibwait (device%, 0)
    ' Other code may be inserted here
WEND
```

C

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

PASCAL

```
type
       buf = array[1...5000] of char;
var
       readbuffer:buf;
begin
       ibwrt(device, readbuffer);
       ibrda(device, readbuffer, 5000);
while ibsta adn (CMPL + EERR) = 0
do IBWAIT (device,0);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRDF

Purpose

Reads data from the GPIB into a file.

Syntax

BASIC
```
CALL ibrdf(boarddev%, filename$)
```

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

PASCAL
```
ibrdf (boarddev:Integer; var filename:flbuf)
```

Parameters

`boarddev` is an integer containing the device/board 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 will be created if it does not already exist.

Returns

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

`iberr` will contain an error code, if an error occurred. An EFSO error will be 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 64K, use ibcntl instead of `ibcnt`

Usage Notes

1. 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.
   - A DCL or SDC command is received from the Active Controller.

2. 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 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.

BASIC
```
CALL ibwrt (boarddev%, "DUMP")
CALL ibrdf (boarddev%, "c:gpib.dat")
```

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

PASCAL
```
var
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
  boarddev: integer;
begin
  ibwrt (boarddev,'c:gpib.dat');
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRDI (QuickBASIC, BASIC only)

Purpose

Reads data into an integer array.

Syntax

BASIC

```
CALL ibrdi(boarddev%, iarr%, bytecount&)
```

C Not Supported.

PASCAL Not Supported.

Parameters

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

`iarr` is the integer array into which data is to be read.

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

Returns

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

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR will result if the specified GPIB Interface Board is and Active Controller but has not been addressed to listen. An EABO errror will result 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 64K, use ibcntl instead of `ibcnt`

Usage Notes

1. As data is read, each byte pair is treated as an integer.
2. In C and PASCAL, IBRD can be used to read both strings and integers.

Example

This example reads 90 bytes of data from a device address at MTA 15 (hex 4F, ASCII 0) to GPIB Board 1. The GPIB board's listen address is MLA 1. (hex 21, ASCII !).

BASIC

```
DIM iarr%(45)    'dimension array as 1/2 of
byte count
brd1% = ilfind("GPIB1")
ibsta% = ilcmd (brd1%, "?0!", 3)
ibsta% = ilrdi (brd1%, iarr%(), 90)
```

C Not Supported.

PASCAL Not Supported.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRDIA (QuickBASIC, BASIC only)

Purpose

Reads data asynchronously data into an integer array.

Syntax

BASIC `CALL ibrdia(boarddev%, iarr%, bytecount&)`

C Not Supported.

PASCAL Not Supported.

Parameters

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

`iarr` is the integer array into which data is to be read.

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

Returns

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

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR will result if the specified GPIB Interface Board is and Active Controller but has not been addressed to listen. An EABO errror will result 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 64K, use ibcntl instead of `ibcnt`

Usage Notes

1. As data is read, each byte pair is treated as an integer. Thus, the array size (dimension) should be half of the byte count.

2. Remember that once an asynchronous I/O call has begun, any further calls to a CIO-488 routine (except for **IBWAIT** and those functions which do not specify a device or GPIB Interface Board) cannot be executed until the I/O has finished and the GPIB driver and the application have been re-synchronized. This is accomplished 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 has been reset, and the driver and application are synchronized.

IBWAIT (mask contains CMPL)	The driver and application are synchronized.
IBSTOP	The asynchronous I/O is canceled, and the driver and applications are synchronized.
IBONL	The asynchronous I/O is canceled, the interface has been reset, and the driver and application are synchronized.

Re-synchronization is successful if a CMPL is returned to `ibsta`. See Appendix C for further information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example reads 90 bytes of data from a device address at MTA 15 (hex 4F, ASCII 0) to GPIB Board 1. The GPIB board's listen address is MLA 1. (hex 21, ASCII !).

BASIC

```
DIM iarr%(45)    'dimension array as 1/2 of byte
count
brd1% = ilfind("GPIB1")
ibsta% = ilcmd (brd1%, "?0!", 3)
ibsta% = ilrdia (brd1%, iarr%(), 90)
```

C Not Supported.

PASCAL Not Supported.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRPP

Purpose

Initiates a parallel poll.

Syntax

BASIC
```
CALL ibrpp(boarddev%, command%)
```

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

PASCAL
```
ibrpp (boarddev:Integer, command:Integer)
```

Parameters

`device` is an integer containing 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.

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

Usage Notes

1. 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 will conduct the parallel poll.

   Note that if the GPIB Interface Board to conduct the parallel poll is not the Controller-In-Charge, an ECIC error will be generated.

2. 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 to respond to 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 ist bit to 1. The first **IBPPC** specifies that the first device (voltmeter%) should drive the DIO1 line high when its ist line goes high. The second IBPPC specifies that the second device (scope%) should drive the DIO2 line high when its ist bit goes high. The **IBRPP** conducts the poll and checks DIO1 and DIO2 to see if either device is requesting service.

BASIC
```
CALL ibppc (voltmeter%, &H68)
CALL ibppc (scope%, &H69)
CALL ibrpp (board%, pollbyte%)
IF pollbyte% AND 1 <> 0 THEN
     PRINT "Volt meter is requesting service"
IF pollbyte% AND 2 <> 0 THEN
     PRINT "Oscilloscope is requesting service"
```

C
```
int  voltmeter, scope, board;
ibppc (voltmeter, 0x68)
ibppc (scope, 0x69)
ibrpp (board, &pollbyte)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
if pollbyte & 1
      printf "Volt meter is requesting service"
if pollbyte & 2
      printf "Oscilloscope is requesting service"
```

PASCAL

```
var
   voltmeter, scope, board, pollbyte: integer;
begin
   ibppc (voltmeter, 0 x 68);
   ibppc (scope, 0x69);
   ibrpp (board, board, pollbyte);;
   if pollbyte and 1 then
      writeln ('Volt meter is requesting service');
   if pollbyte and 2 then
    writeln ('Oscilloscope is requesting service');
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRSC

Purpose

Request/Releases System Controll.

Syntax

BASIC
```
CALL ibrsc (board%, control%)
```

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

PASCAL
```
ibrsc (board:integer; control:integer)
```

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.

`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

1. There may only be one System Controller in a GPIB system.

Example

This example makes GPIB board 1 the System Controller.

BASIC
```
CALL ibfind ("GPIB1", gpib1%)
CALL ibrsc (gpib1%, 3)
```

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

PASCAL
```
var
 gpib1 : integer;
begin
 gpib1 := ibfind ('gpib1');
 ibrsc (gpib1, 3);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRSP

Purpose

Serial polls a device.

Syntax

BASIC
```
CALL ibrsp(device%, serialpollbyte%)
```

C
```
ibrsp (int device, char *serialpollbyte)
```

PASCAL
```
ibrsp (device:Integer; var 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.

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

Usage Notes

1. 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.

2. 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.

3. 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

This example returns the serial response byte (into `serialpollbyte%`) of device 1.

BASIC
```
CALL ibfind ("DEVICE1",dev1%)
CALL ibrsp (dev1%, spr%)
```

C
```
int dev1;
char spbyte
dev1 = ibfind ("device1");
ibrsp (dev1, &spbyte);
```

PASCAL
```
var
 dev1 := integer;
 spbyte := char;
begin
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
dev1 := ibfind ('device1');
ibrsp (dev1, spbyte);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBRSV

Purpose

Changes the serial poll response byte of the specified GPIB board or device.

Syntax

BASIC
```
CALL ibrsv (boarddev%, statusbyte%)
```

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

PASCAL
```
ibrsv (boarddev:Integer; statusbyte:Integer)
```

Parameters

`boarddev` is an integer containing the board/device 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 will be 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.

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

Usage Notes

1. 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.

BASIC
```
CALL ibfind ("GPIB1",gpib1%)
CALL ibrsv (gpib1%, &H41)
```

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

PASCAL
```
var
 gpib1: integer;
begin
 gpib1 = ibfing ('gpbi1');
 ibrsv (gpib1, $41);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBSAD

Purpose

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

Syntax

BASIC
Subroutine
```
CALL ibsad (boarddev%, address%)
```
Function
```
ilsad (boarddev%, address%)
```

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

PASCAL
```
ibsad(boarddev:Integer; address:Integer)
```

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. Otherwise 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 will be used until it is redefined (by calling **IBSAD** again) the device/board is re-initialized (by callling **IBFIND** or **IBONL**); or the program is restarted.

Returns

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

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

Usage Notes

See IBPAD.

Example

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

BASIC
```
CALL ibfind ("DEVICE5",dev5%)
CALL ibsad (dev5%, &H67)
```

C
```
int dev5;
dev5 = ibfind ("device5");
ibsad (dev5, 0x67);
```

PASCAL
```
var
 dev5: integer;
begin
 dev5 := ibfind ('device5');
 ibsad (dev5, 0x67);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBSIC

Purpose

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

Syntax

BASIC
```
CALL ibsic (board%)
```

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

PASCAL
```
ibsic (board:integer)
```

Parameters

`board` is an integer containing the board handle.

Returns

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

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

Usage Notes

1. This routine can only be used if the specified GPIB Interface Board is the System Controller. When the routine is executed, the GPIB interface Board asserts the IFC (Interface Clear) signal for at least 100 *sec. This action results in the System Controller regaining control of the GPIB (i.e., becoming the Controller-In-Charge). When the 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.

BASIC
```
CALL ibfind ("GPIB1", gpib1%)
CALL ibsic (gpib1%)
```

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

PASCAL
```
var
 gpib1 : integer;
begin
 gpib1 := ibfind ('GPIB1');
 ibsic (gpib1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBSRE

Purpose

Asserts/Unasserts the REN (Remote Enable) line.

Syntax

BASIC
```
CALL ibsre (board%, ren%)
```

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

PASCAL
```
ibsre (board:integer; ren:integer)
```

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 will be asserted.

Returns

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

`iberr` will contain an error code, if an error occurred. The ESAC error will be generated if the specified GPIB Interface Board is not the System Controller. If no error occurs, `iberr` will contain the previous REN state.

Usage Notes

1. This routine can only be used if the specified GPIB Interface Board is the System Controller. Remember that even though the REN line is asserted, the device(s) will not be put into remote state until is addressed to listen by the Active Controller. When the REN line is unasserted, all devices will 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.

BASIC
```
CALL ibfind ("GPIB1", gpib%)
CALL ibsre (gpib1%, 2) ' You can use any non-zero
value
CALL ibcmd (gpib1%, ",")
```

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

PASCAL
```
var
 gpib1 : integer;
begin
 gpib1 := ibfind ('GPIB1');
 ibsre (gpib1, 2);
 ibcmd (gpib1, ",", 1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBSRQ (C only)

Purpose

Defines the user-written service routine to be executed upon detection of SRQ.

Syntax

BASIC

Subroutine Not Applicable. (Use ON PEN.)
Function Not Applicable. (Use ON PEN.)

C

```
ibsrq (void (far *func) (void) )
```

PASCAL Not Applicable

Parameters

`func` is the procedure which is to be called whenever the SRQI bit is set (= 1) in `ibsta`. If `func` = NULL, then SRQ servicing will be disabled.

Returns

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

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

Usage Notes

Automatic serial polling must be disabled in order to use this routine.

The SRQI bit in `ibsta` is checked after each call to the handler is made. If the SRQI bit = 1, then the `func` routine is called before control is returned to the application program.

Example

Define the routine srqroutine to be called when SRQI is set (=1).

BASIC Not Applicable

C

```
int plotter;
main ()
     {
     int gpib1 = ibfind ("gpib1");
     /*disable autopolling */
     ibconfig (gpib1, ibcAUTOPOLL, 0);
     plotter = ibfind ("DEVICE5");
     ibsrq (srqroutine);
     }
void far srqroutine ()
     {
     char spr;
     ibrsp (plotter, &spr);
     /* analyze spr here */
     }
```

PASCAL Not Applicable

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBSTOP

Purpose

Terminate an asynchronous operation.

Syntax

BASIC
```
CALL ibstop (boarddev%)
```

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

PASCAL
```
ibstop (boarddev:Integer)
```

Parameters

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

Returns

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

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

Usage Notes

1. If a device is specified, all unfinished asynchronous operations (read, write, or command) associated with that device will be stopped. Likewise, if a GPIB Interface Board is specified, all unfinished asynchronous operations associated with that board will be 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.

BASIC
```
CALL ibfind ("DEV2", dev%)
CALL ibwrta (dev%, "datafile")
CALL ibstop (dev%)
```

C
```
int dev;
dev = ibfind ("DEV2");
ibwrta(dev, "datafile");
ibstop (dev);
```

PASCAL
```
var
 dev: integer;
begin
 dev:= ibfind ('DEV2');
 ibwrta (dev, 'datafile');
ibstop (dev);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBTMO

Purpose

Changes timeout value.

Syntax

BASIC `CALL ibtmo (boarddev%, timeout%)`

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

PASCAL `ibtmo (boarddev:integer; timeout:integer)`

Parameters

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

`timeout` specifies the timeout. The timeout value determines how long all I/O routines will wait for a response from a device. When the timeout period expires during an I/O operation, the I/O function will return an EABO error. Valid timeout codes are shown in Table 4-4.

Code	Minimum Timeout	Code	Minimum Timeout
TNONE	Disabled	**T100ms**	100 msec
T10us	10μsec	**T300ms**	300 msec
T30us	30μsec	**T1s**	1 sec
T100us	100μsec	**T3s**	3 sec
T300us	300μsec	**T10s**	10 sec
T1ms	1 msec	**T30s**	30 sec
T3ms	3msec	**T100s**	100 sec
T10ms	10msec	**T300s**	300 sec
T30ms	30 msec	**T1000s**	1000 sec

Table 4-4. Timeout Codes

Returns

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

`iberr` will contain an error code, if an error occurred. If no error occured, `iberr` will contain the prevous timeout code.

Usage Notes

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

2. The new timeout will be 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.

BASIC

```
CALL ibfind("PLOTTER", plotter%)
CALL ibtmo (plotter%, T30)

int plotter;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C

```
plotter = ibfind ("PLOTTER");
ibtmo(plotter, T30);
```

PASCAL

```
var
 plotter : integer;
begin
 plotter := ibfind ('PLOTTER');
 ibtmo(plotter, T30);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBTRG

Purpose

Triggers the specified device.

Syntax

BASIC
```
CALL ibtrg (device%)
```

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

PASCAL
```
ibtrg (device:Integer)
```

Parameters

`device` is an integer containing device handle.

Returns

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

BASIC
```
CALL ibfind ("DEVICE6", plotter%)
CALL ibtrg (plotter%)
```

C
```
int plotter;
plotter = ibfind ("DEVICE6");
ibtrg (plotter);
```

PASCAL
```
var
 plotter: integer;
begin
 plotter := ibfind ('DEVICE6');
 ibtrg (plotter);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBWAIT

Purpose

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

Syntax

BASIC
```
CALL ibwait (boarddev%, mask%)
```

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

PASCAL
```
ibwait (boarddev:integer; mask:word)
```

Parameters

`boarddev` is an integer containing the device/board 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	7	6	5	4	3	2	1	0
	EERR	TIMO	EEND	SRQI	RQS	SPOL	EVENT	CMPL	LOK	REM	CIC	AATN	TACS	LACS	DTAS	DCAS

For more information regarding `ibsta`, see Appendix C.

Returns

`ibsta` will contain a 16-bit status word as described in Appendix B

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

Usage Notes

1. 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. For example, these two BASIC/QuickBASIC calls are equivalent:

```
IF IBSTA% AND TIMO THEN PRINT "TIMEOUT"
```

or

```
IF IBSTA% AND &H4000 THEN PRINT "TIMEOUT"
```

2. This routine also updates `ibsta`. If `mask` = 0 or `mask` = 8000 hex (the EERR bit), the routine returns immediately with a fresh update of ibsta.

3. If the TIMO bit is 0 or a previous **IBTMO** specifies a time limit of 0, timeouts are disabled. This should only be done when setting `mask` = 0 or when it is certain the selected event will occur. Otherwise, the processor may wait indefinitely for the event to occur.

4. Remember that if a device is specified, only EERR, TIMO, EEND, RQS, and CMPL are applicable.

   Also, if automatic polling is enabled and RQS is specified by **IBWAIT**, then each time the SRQ line is asserted, the GPIB Interface Board associated with the specified device will serial poll all devices on its bus and save the responses. This continues

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

until the status byte returned by the device specified by **IBWAIT**, indicates that it was the device requesting service. If TIMO is set, **IBWAIT** returns if the event does not occur withn the timeout period of the device.

5. 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.

BASIC

```
CALL ibfind ("DEVICE7", plotter%)
CALL ibwait (plotter%, RQS)
```

C

```
int plotter;
plotter = ibfind ("DEVICE7");
ibwait (plotter, RQS);
```

PASCAL

```
var
 plotter:integer;
begin
 plotter := ibfind ('DEVICE7');
 ibwait (plotter, RQS);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBWRT

Purpose

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

Syntax

BASIC
```
CALL ibwrt (boarddev%, buf$)
```

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

PASCAL
```
ibwrt (boarddev:Integer; var buf; bytecount:Word)
```

Parameters

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

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR will result 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 64K, use `ibcntl` instead of `ibcnt`.

Usage Notes

1. This routine is used to send device-specific commands.

2. 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 Device Clear) is received from the Active Controller.
   - All data is sent

3. 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 will unassert ATN in order to send data. This routine leaves the board in that state.

4. 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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

specified device.

BASIC

```
CALL ibfind ("DEVICE7")
wrt$ = "IP2X5" + chr$(&H0D) + chr$(&H0A)
CALL ibwrt(ptr%, wrt$)
```

C

```
int ptr;
ptr = ibfind ("DEVICE7");
ibwrt (ptr,"IP2X5\r\n",7);
```

PASCAL

```
var
 ptr : integer;
begin
 ptr := ibfind ('DEVICE7');
 ibwrt (ptr, 'IP2X5#13#10',7);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBWRTA

Purpose

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

Syntax

BASIC
```
CALL ibwrta (boarddev%, buf$)
```

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

PASCAL
```
ibwrta (boarddev:Integer; var buf; bytecount:Word)
```

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.

Returns

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

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

`iberr` will contain an error code of the first error detected, if an error occurred. An EADR will result 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 64K, use `ibcntl` instead of `ibcnt`.

Usage Notes

1. A write will terminate when one of the following occurs:
   - An error is detected.
   - The time limit is exceeded.
   - A DCL or SDC is received from the Controller-In-Charge

2. 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 Active Controller, it will unassert ATN in order to send data. This routine leaves the board in that state.

3. This command is normally only used for very large data transfers. It allows the program to continue running in the foreground while the data transfer continues in the background. For small data transfers, use **IBWRT**.

   Small data transfers are very quick. In those cases it is simpler to use IBWRT() instead of IBWRTA().

Example

In this example, **IBWRT** sends a command ("UPLOAD") to a device. The device expects

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

BASIC

```
writebuffer$ = space(5000)
CALL ibwrt (device%, "UPLOAD")
CALL ibwrta (device%, writebuffer$)
WHILE (ibsta% AND CMPL+EERR) = 0
' Check for complete or error
    ibwait (device%, 0)
' Other code may be inserted here
WEND
```

C

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

PASCAL

```
type
    buf = array [1...5000] of char;
var
    wrtbuffer : buf;
begin
    ibwrt (device, 'UPLOAD');
    ibwrta (device, writebuffer, 5000);
    while ibsta and (CMPL+EERR) = 0 do
     ibwait (device, 0);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBWRTF

Purpose

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

Syntax

BASIC
```
CALL ibwrtf(boarddev%, filename$)
```

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

PASCAL
```
ibwrtf (boarddev:integer; var filename:flbuf)
```

Parameters

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

`filename` is the name of the file (up to 50 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.

Returns

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

`iberr` will contain an error code, if an error occurred. An EFSO error will be 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 64K, use `ibcntl` instead of `ibcnt`.

Usage Notes

1. A write will terminate when one of the following occurs:
   - An error is detected.
   - The time limit is exceeded.
   - A DCL or SDC command is received from the ActiveController.
   - All data has been sent

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

3. 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 Controller-In-Charge, it will unassert 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 recieve a large amount of data. The program then sends the data from a file to the device.

BASIC
```
CALL ibwrt (device%, "UPLOAD")
CALL ibwrtf (device%, "c:gpib.dat")
```

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

```
var
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
PASCAL    device : integer;
          begin
           ibwrt (device, 'UPLOAD');
           ibwrtf (device, 'c:gpib.dat');
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBWRTI (QuickBASIC, BASIC only)

Purpose

Writes data from an integer array.

Syntax

BASIC `CALL ibwrti(boarddev%, iarr%, bytecount&)`

C Not Supported.

PASCAL Not Supported.

Parameters

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

`iarr` is the integer array which will be written to the device.

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

Returns

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

`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 64K, use `ibcntl` instead of `ibcnt`.

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

Usage Notes

1. As data is written, each byte pair is treated as an integer.
2. C and PASCAL users should use IBWRT for both strings and integers.

Example

This example writes 90 bytes of data from a device address at MTA 15 (hex 4F, ASCII 0) to GPIB Board 1. The GPIB board's listen address is MLA 1. (hex 21, ASCII !).

BASIC

```
DIM iarr%(90)
CALL ibfind ("GPIB1", brd1%)
CALL ibcmd (brd1%, "?0!")
CALL ibwrti (brd1%, iarr%(), 90)
```

C Not Supported.

PASCAL Not Supported.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBWRTIA (QuickBASIC, BASIC only)

Purpose

Writes data asynchronously data from an integer array.

Syntax

BASIC `CALL ibwrtia(boarddev%, iarr%, bytecount&)`

C Not Supported

PASCAL Not Supported

Parameters

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

`iarr` is the integer array which will be written to the device. The array can be of type single, double, or long. `iarr` can not be a dynamic array.

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

Returns

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

`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 64K, use `ibcntl` instead of `ibcnt`.

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

Usage Notes

1. As data is written, each byte pair is treated as an integer.

2. Remember that once an asynchronous I/O call has begun, any further calls to a CIO-488 routine (except for IBWRT and those functions which do not specify a device or GPIB Interface Board) cannot be executed until the I/O has finished and the GPIB driver and the application have been re-synchronized. This is accomplished by calling one of the following:

IBWAIT (make contains CMPL)	The driver and applications 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 has been reset and the driver and application are synchronized.

Re-synchronization is successful if a CMPL is returned to `ibsta`. See Appendix C for further information.

3. C and Pascal users should use IBRDA.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example writes 90 bytes of data from a device address at MTA 15 (hex 4F, ASCII 0) to GPIB Board 1. The GPIB board's listen address is MLA 1. (hex 21, ASCII !).

BASIC

```
DIM iarr%(90)
CALL ibfind("GPIB1", brd1%)
CALL ibcmd (brd1%, "?0!")
CALL ibwrtia (brd1%, iarr%(), 90)
```

C

Not Supported.

PASCAL

Not Supported.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5 THE CIO-488.2 ROUTINES

This chapter describes each of the CIO-488.2 routines. Each description provides a short description of the routine, its syntax, parameters, and any values which are returned. Any special usage notes and an example of are also given. The routines are listed in alphabetical order. Table 5-1 lists all of the CIO-488.2 routines.

NAME	DESCRIPTION
AllSpoll	Serial polls all devices
DevClear	Clears a single device
DevClearList	Clears multiple devices
EnableLocal	Enables local programming of a device
EnableRemote	Enables remote programming of a device
FindLstn	Find all Listeners
FindRQS	Find device requesting service
PassControl	Make another device the Active Controller
PPoll	Conduct a parallel poll
PPollConfig	Configure a device for parallel polls
PPollUnconfig	Unconfigure devices for parallel polls
RcvRespMsg	Read data from a previuosly addressed GPIB device
ReadStatusByte	Serial poll a single device
Receive	Read data from a GPIB device
ReceiveSetup	Address a device as a talker and a GPIB Interface Board as a Listener
ResetSys	Initialize a GPIB system
Send	Send data to one device
SendCmds	Send GPIB commands
SendDataBytes	Send data to previously addressed GPIB devices
SendIFC	Assert the IFC line
SendList	Send data to multiple GPIB devices
SendLLO	Send Local Lockout (LLO) to all devices
SendSetup	Prepare devices to receive data
SetRWLS	Put devices in Remote with Lockout state
TestSRQ	Determine status of SRQ line
TestSys	Force devices to conduct self-tests
Trigger	Trigger a single device
TriggerList	Trigger multiple devices
WaitSRQ	Wait until a device asserts SRQ

Table 5-1. The CIO-488.2 Routines

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

AllSpoll

Purpose

Performs a serial poll on specified devices.

Syntax

BASIC

```
Call AllSpoll (board%, addresslist%(), result-
list%())
```

C

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

Parameters

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

`iberr` will contain an error code, if an error occurred. If a device times out, `iberr` will contain Error 6 - EABO (See Appendix C for more information.) and `ibcnt` will contain the index of the timed-out device.

Usage Notes

1. 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.

BASIC

```
DIM addresslist%(3)
DIM resultlist%(2)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL AllSpoll (0, addresslist% (), resultlist% ())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DevClear

Purpose

Clears one device.

Syntax

BASIC

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

C

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

Parameters

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

`address` is the GPIB address of the device to be cleared.

Returns

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

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

Usage Notes

1. This routine sends the GPIB Selected Device Clear (SDC) message to the specified device.
2. To clear multiple devices, use the **DevClearList** routine.
3. If `address` is set to `NOADDR`, then all connected devices on the GPIB will be cleared via the Universal Device Clear (UDC) message.

Example

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

BASIC

```
address% = 4 + 256*30
CALL DevClear (0, address%)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DevClearList

Purpose

Clears specified devices.

Syntax

BASIC
```
CALL DevClearList(board%, addresslist%())
```
C
```
void  DevClearList(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 will be 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.

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

Usage Notes

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

2. To clear all devices, in BASIC/QuickBASIC use an array containing only the `NOADDR` value. In C, pass a NULL value. This sends the GPIB Universal Device Clear (DCL) message.

3. To clear only one device, use DevClear.

Example

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

BASIC
```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL DevClearList(0, addresslist% ())
```
C
```
int addresslist[3] = {6,7,NOADDR};
DevClearList(0, addresslist);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EnableLocal

Purpose

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

Syntax

BASIC

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

C

```
void  EnableLocal(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 will be 0.

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

Returns

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

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

Usage Notes

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

2. To put all devices in local mode, in BASIC/QuickBASIC use an array containing only the `NOADDR` value. In C, pass a NULL 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.

BASIC

```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL EnableLocal (0, addresslist% ())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EnableRemote

Purpose

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

Syntax

BASIC

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

C

```
void  EnableRemote(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 will be 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.

`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.

To put all devices in remote mode, in BASIC/QuickBASIC use an array containing only the `NOADDR` value. In C, pass a NULL value. This asserts the GPIB Remote Enable (REN) line, thereby placing all GPIB devices in remote mode.

Example

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

BASIC

```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL EnableRemote (0, addresslist% ())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FindLstn

Purpose

Finds all listeners on the GPIB.

Syntax

BASIC

```
CALL FindLstn(board%, addresslist%(), result-
list%(), limit%)
```

C

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

Parameters

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

`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` will contain the number of addresses actually placed in `resultlist`.

Usage Notes

1. 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 `result-list`. 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`.

2. This routine can be used 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 any listening devices are present at GPIB primary addresses 6 and 7 on Board 0.

BASIC

```
DIM addresslist%(3)
DIM resultlist%(4)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
limit% = 4
CALL FindLstn (0, addresslist% (), resultlist% (),
limit%)
```

C

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FindRQS

Purpose

Identify the device requesting service.

Syntax

BASIC

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

C

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

Parameters

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

iberr will contain an error code, if an error occurred. **iberr** will contain the error code ETAB, if no device is requesting service. In this case, **ibcnt** will contain **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** will contain the index of the timed-out device.

Usage Notes

1. None.

Example

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

BASIC

```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL FindRQS(0, addresslist% (), result%)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PassControl

Purpose

Makes another device the Active Controller.

Syntax

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

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

Parameters

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

Returns

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

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

Usage Notes

1. 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. Note that 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.

BASIC
```
address% =6
CALL PassControl(0,address%)
```

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PPoll

Purpose

Performs a Parallel Poll.

Syntax

BASIC

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

C

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

Parameters

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

Returns

`result` will contain the eight-bit result of the parallel poll. Each bit of the poll result will contain 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 via PPollConfig and the individual status of the devices.

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

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

Usage Notes

1. None.

Example

Parallel polls devices connected to board 0.

BASIC

```
CALL PPoll(0, result%)
```

C

```
int result;
PPoll(0, &result);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PPollConfig

Purpose

Configures a device for parallel polls.

Syntax

BASIC

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

C

```
void  PPollConfig(int board, int addresslist, 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 will be 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 will use 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 will assert 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.

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

Usage Notes

1. Remember that if a device is locally configured for a parallel poll, the Controller's parallel poll configuration instruction will be 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 will assert line 7 if its IST bit = 1 and unassert line 7 if IST = 0).

BASIC

```
Call PpollConfig(0,6,7,1)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PPollUnconfig

Purpose

Unconfigures devices for parallel polls.

Syntax

BASIC

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

C

```
void  PPollUnconfig(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 will be 0.

`addresslist` is an array of GPIB addresses, terminated by the value `NOADDR`. These addresses identify the devices which are no longer to respond to a parallel poll.

Returns

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

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

Usage Notes

1. To unconfigure all devices, in BASIC/QuickBASIC use an array containing only the `NOADDR` value. In C, pass a NULL value. This sends the GPIB Parallel Poll Unconfigure (PPU) command.

Example

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

BASIC

```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL PPollUnconfig(0, addresslist% ())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RcvRespMsg

Purpose

Reads data from a previously addressed device.

Syntax

BASIC

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

C

```
void  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 will be 0.

`count` specifies the maximum number of data bytes which are to be read. This parameter is not required in BASIC, since `count` will be set to the length of the data string.

`data` is the string which will receive the data. In BASIC, it is important that you allocate a large enough string to accomodate all of the data.

`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 will be stopped when this character is detected. If `termination` = `STOPend` (A constant defined in the header file.), then the read will be stopped when EOI is detected.

Returns

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

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

Usage Notes

1. You must address the appropriate devices as Listeners/Talkers prior to calling this routine.

Example

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

BASIC

```
DIM data AS STRING*50
CALL RcvMsgResp(0,data$,STOPend)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ReadStatusByte

Purpose

Serial poll a single device and read its status byte.

Syntax

BASIC

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

C

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

Parameters

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

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

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

Usage Notes

1. None.

Example

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

BASIC

```
CALL ReadStatusByte (0, 2, result% )
```

C

```
int result;
AllSpoll (0, 2, &result);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Receive

Purpose

Reads data from a GPIB device.

Syntax

BASIC

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

C

```
void 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 will be 0.

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

`data` is the string which will receive the data. In BASIC, it is important that you allocate a large enough string to accomodate all of the data.

`count` specifies the maximum number of data bytes which are to be read. This parameter is not required in BASIC, since `count` will be set to the string length of the data.

`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 will be stopped when this character is detected. If `termination` = `STOPend` (constant defined in the header file), then the read will be stopped when EOI is detected.

Returns

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

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

Usage Notes

1. None.

Example

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

BASIC

```
data$ = space$(50)
CALL Receive (0, 2, data$, STOPend)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ReceiveSetup

Purpose

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

Syntax

BASIC

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

C

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

Parameters

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

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

Returns

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

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

Usage Notes

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

2. 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.

3. 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 will be received and then stored in a string. The message will be terminated with an EOI.

BASIC

```
DIM message AS STRING*50
CALL ReceiveSetup(0,5)
CALL RcvRespMsg(0,message$,STOPend)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ResetSys

Purpose

Initializes GPIB System.

Syntax

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

C
```
void  ResetSys(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 will be 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.

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

Usage Notes

1. 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).

2. 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 will be able to 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 will reset the GPIB devices connected to GPIB board 0 and assigned GPIB bus addresses of 6 and 7.

BASIC
```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
CALL ResetSys (0, addresslist% ())
```

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Send

Purpose

Sends data to one GPIB device.

Syntax

BASIC

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

C

```
void  Send (int board, int 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 will be 0.

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

count specifies the maximum number of data bytes which are to be sent to the device. This parameter is not required in BASIC, since **count** will be set to the string length of the data.

data is the string which will receive the data. In BASIC, it is important that you allocate a large enough string to accomodate all of the data.

eotmode is the flag used to signal the end of data. It can equal one of the following:

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.

Returns

NOTE: These constants are defined in the header files.

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

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

Usage Notes

1. 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.

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.

BASIC

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

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SendCmds

Purpose

Send GPIB commands.

Syntax

BASIC

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

C

```
void  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 will be 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. This parameter is not required in BASIC, since `count` will be set to the string length of the data.

Returns

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

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

Usage Notes

1. 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.

BASIC

```
CALL SendCmds (0,chr$(&3F)+chr$(&40)+chr$(&H2
8)
```

C

```
SendCmds (0,"\x3F\x40\x28\x29\x04\x01", 6);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SendDataBytes

Purpose

Sends data to previously addressed devices.

Syntax

BASIC

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

C

```
void  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 will be 0.

`data` is the string that contains the data.

`count` specifies the maximum number of data bytes which are to be sent to the device. This parameter is not required in BASIC, since `count` will be set to the string length of the data.

`eotmode` is the flag used to signal the end of data. It can equal one of the following:

`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.

NOTE: These constants are defined in the header files.

Returns

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

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

Usage Notes

1. This routine assumes that the desired GPIB listeners have already been addressed.

Example

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

BASIC

```
Call SendDataBytes (0,"*IDN?", DABend)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SendIFC

Purpose

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

Syntax

BASIC `CALL SendIFC(board%)`

C `void SendIFC(int board)`

Parameters

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

Returns

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

`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.

BASIC
```
CALL SendIFC(0)
```

C
```
SendIFC(0);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SendList

Purpose

Sends data to multiple GPIB devices.

Syntax

BASIC

```
CALL SendList(board%, addresslist(), data$, eot-
mode%)
```

C

```
void  SendList(int board, int addresslist[],
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 will be 0.

count specifies the maximum number of data bytes which are to be sent to the device. This parameter is not required in BASIC, since **count** will be set to the string length of the data.

data is the string containing the data to be sent.

eotmode is the flag used to signal the end of data. It can equal one of the following:

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.

Returns

NOTE: These constants are defined in the header files.

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

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.

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.

BASIC

```
DIM addresslist%(3)
addresslist% (0) = 6
addresslist% (1) = 7
addresslist% (2) = NOADDR
Call SendList (0, addresslist%(),"*IDN?", DABend)
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SendLLO

Purpose

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

Syntax

BASIC `CALL SendLLO(board%)`

C `void SendLLO(int board)`

Parameters

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

Returns

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

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

Usage Notes

1. 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.

BASIC
```
CALL SendLLO (0)
```

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SendSetup

Purpose

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

Syntax

BASIC

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

C

```
void  SendSetup(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 will be 0.

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

Returns

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

`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.

BASIC

```
DIM addresslist%(3)
addresslist%(0) = 6
addresslist%(1) = 7
addresslist%(2) = NOADDR
CALL SendSetup(0,addresslist())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SetRWLS

Purpose

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

Syntax

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

C
```
void  SetRWLS(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 will be 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.

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

Usage Notes

1. 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.

BASIC
```
DIM addresslist%(3)
addresslist%(0) = 6
addresslist%(1) = 7
addresslist%(2) = NOADDR
CALL SetRWLS(0,addresslist())
```

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TestSRQ

Purpose

Evaluate state of SRQ line.

Syntax

BASIC

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

C

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

Parameters

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

Returns

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

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

`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.

BASIC

```
CALL TestSRQ (0,result%)
IF result% = 1 then
     'SRQ is asserted
ELSE
     'No SRQ at this time
END IF
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TestSys

Purpose

Activate self-test procedures of specified devices.

Syntax

BASIC

```
CALL TestSys (board%, addresslist% (), result-
list%())
```

C

```
void  SendSetup(int board, int addresslist [],
int resultlist[])
```

Parameters

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

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

Returns

`resultlist` is an array which will contain 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 that an error occurred.

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

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

`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.

BASIC

```
DIM resultlist%(2)
DIM addresslist%(3)
addresslist%(0) = 6
addresslist%(1) = 7
addresslist%(2) = NOADDR
CALL TestSys(0,addresslist(), resultlist())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trigger

Purpose

Triggers one device.

Syntax

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

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

Parameters

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

address is an integer representing the GPIB address of the device that is to to be triggered. If **address** = **NOADDR** then all already addressed Listeners will be triggered.

Returns

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

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

Usage Notes

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

2. 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.

BASIC
```
CALL Trigger (0, 6 + 256*12)
```

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TriggerList

Purpose

Triggers multiple GPIB devices.

Syntax

BASIC

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

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 will be 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` (or , in C, a NULL value is passed), all previously addressed listeners will be triggered.

Returns

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

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

Usage Notes

1. 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.

BASIC

```
DIM addresslist%(3)
addresslist%(0) = 6
addresslist%(1) = 7
addresslist%(2) = NOADDR
CALL TriggerList(0,addresslist())
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WaitSRQ

Purpose

Wait until a device asserts SRQ.

Syntax

BASIC

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

C

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

Parameters

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

Returns

`result` indicates whether or not an SRQ occurred. If an SRQ ocurs before the timeout expires, `result` will equal 1. Otherwise, `result` will equal 0.

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

`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.

BASIC

```
DIM addresslist%(3), resultlist(2)
addresslist%(0) = 6
addresslist%(1) = 7
addresslist%(2) = NOADDR
CALL WaitSRQ(0,result%)
IF result% = 1 then
   CALL FindSRQ(0, addresslist%(), resultlist%())
END IF
```

C

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6 APPENDIX A - MULTILINE INTERFACE MESSAGES

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
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	SPD	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

Message Definitions

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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
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
4	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
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

Message Definitions

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7 APPENDIX B - **IBSTA**

Every DRVR-PC2A 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, 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. For example, if you are programming in QuickBASIC, the following two calls are equivalent:

```
IF IBSTA% AND TIMO THEN PRINT "TIMEOUT ERROR"

IF IBSTA% AND &H8000 THEN PRINT "TIMEOUT ERROR"
```

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**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	ERR	TIMO	END	SRQI	RQS	SPOLL	EEVENT	CMPL	LOK	RREM	CIC	AATN	TACS	LACS	DTAS	DCAS
Hex Value	**800**	**400**	**200**	**100**	**800**	**400**	**200**	**100**	**80**	**40**	**20**	**10**	**8**	**4**	**2**	**1**
	0	**0**	**0**	**0**												

Where:

EERR **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 ? for more information. EERR is cleared following any call that does not result in an error.

NOTE: You should check for errors after each call. If an undetected error occurs early in your program, it may not become apparent until later in the program. At which time, it will become 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:

- A synchronous I/O function exceeds the programmed timeout.
- An **IBWAIT** has exceeded the time limit value.

This bit is cleared (set to 0) during all other operations.

Note that different timeout periods may be set for each device. You can set the timeout values either in CBCONF or in your program by uing the IBTMO function.

EEND **END or EOS Detected.** If this bit = 1, EOI has been asserted or an EOS byte has been

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

detected. If the GPIB Interface board has been programmed via an **IBGTS** to perform shadow handshaking, any other routine can set this bit to one.

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:

- The GPIB Interface Board is the Active Controller.
- The GPIB SRQ line is asserted.
- Automatic serial poll capability is disabled.

This bit is cleared (= 0) upon the following conditions:

- The GPIB Interface Board is no longer the Active Controller.
- The GPIB SRQ line is unasserted.

SPOLL **Serial Poll.** If this bit = 1, the board has been serially polled. This can only occur when the board is not the controller.

EEVENT **Event Occurred.** If this bit =1, then an event occurred.

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 (LOL) 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.

RREM **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:

- REN is unasserted.
- The GPIB Interface Board has been addressed as a Listener and receives a GTL (Go to Local) command.
- 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:

- If the board is the System Controller and the IBSIC or SendIFC routine is executed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Control is passed to the GPIB Interface Board.

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.

AATN **Attention.** If this bit = 1, ATN is asserted. Likewise, if this bit = 0, the ATN line is unasserted.

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:

- The UNT command is sent.
- The GPIB Interface Board is sent its listen address.
- Another talker is addressed.
- 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:

- The GPIB Interface Board receives an UNL (Unlisten) command.
- The GPIB Interface Board is addressed as a talker.
- **IBGTS** is called disabling shadow handshaking.adow handshake.3

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.

NOTE: Some of the status word bits can also be cleared under the following circumstances:

- If **IBONL** is called, the EEND, LOK, RREM, CIC, TACS, LACS, DTAS, and DCAS bits are cleared.
- If an ENEB or EDVR (See Appendix C for more information.) error is returned, all of the status word bits except EERR are cleared.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8 APPENDIX C - IBERR

If the EERR 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 each of the possible GPIB Error Codes, its 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. More detailed explanations of the errors and possible solutions can be found in the paragraphs following the table.

ERROR CODE (DECIMAL)	MNEMONIC	DESCRIPTION
0	EDVR	DOS Error
1	ECIC	Specified GPIB Interface Board is Not Active Controller
2	ENOL	No present listening devices
3	EADR	GPIB Interface Board has not been addressed properly
4	EARG	Invalid arguement
5	ESAC	Specified GPIB Interface Board is not System Controller
6	EABO	I/O operation aborted
7	ENEB	Non-existant GPIB board
10	EOIP	Routine not allowed during asynchronous I/O operation
11	ECAP	No capability for operation
12	EFSO	File System Error
14	EBUS	Command byte transfer error
15	ESTB	Serial poll status byte lost
16	ESRQ	SRQ stuck in ON position
20	ETAB	Table problem
247	EINT	No interrupt configured on board
248	EWMD	Windows is not in Enhanced mode
249	EVDD	CBGPIB.386 is not installed
250	EOVR	Buffer Overflow
251	ESML	Two library calls running simultaneously
252	ECFG	Board type does not match GPIB.CFG
253	ETMR	No Windows timers available
254	ESLC	No Windows selectors available
255	EBRK	Control-Break pressed

Table C-1. Possible Error Codes

8.1 EDVR -- DOS ERROR

Cause: A device or board can not be found. ibcnt will contain the DOS error code, which will further identify the problem.

Solution: 1. If ibcnt = 2 (DOS error "Device Not Found"), an invalid device/board name was passed to **IBFIND**.

- Check the spelling of the device/board name and call **IBFIND** again.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Use the *CBCONF* utility to configure the driver to recognize the device/board name.

2. If `ibcnt` = 6 (DOS error "Invalid Handle"), an invalid unit descriptor was passed to a routine. Call **IBFIND** again and make sure that it has completed successfully.

NOTE: Remember that if **IBONL** is called specifying that a device is to be take offline, you must reopen the device (Call **IBFIND**.) before using it again.

8.2 ECIC -- SPECIFIED GPIB INTERFACE BOARD IS NOT THE CONTROLLER-IN-CHARGE

Cause: Many routines require that the GPIB Interface Board is the Controller-In-Charge. These include:

- ATN, EOI, or REN lines.
- Any routines which manipulate the GPIB

Solution: Make sure that the GPIB Interface Board is the Controller-In-Charge

8.3 ENOL -- NO LISTENING DEVICE(S)

Cause: The routine detected no listeners.

Solution:

1. Make sure that a device has been addressed properly.
2. Check that at least two-thirds of the devices on the GPIB bus are turned on, including the specified device.
3. Make sure that all connections are secure.
4. Verify the device(s) GPIB address.
5. Make sure that the device has been configured properly by CBCONF.

8.4 EADR -- GPIB INTERFACE BOARD NOT ADDRESSED.

Cause: The GPIB Interface Board has not been addressed.

Solution:

1. Call IBCMD and address the board.
2. If you are calling IBGTS, with shadow handshaking enabled, call IBCAC.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8.5 EARG -- INVALID ARGUMENT

Cause: An invalid parameter has been provided to a routine.

Solution: 1. Check syntax and range of parameters. See Chapters 4 and/or 6.

8.6 ESAC -- BOARD NOT SYSTEM CONTROLLER

Cause: Routine requires that the GPIB Interface Board is the System Controller.

Solution: 1. Configure the board to be the System Controller. (Run CBCONF or call IBRSC.)

8.7 EABO -- I/O OPERATION ABORTED

Cause: I/O Operation aborted.

Solution: 1. Check that the device is powered on.

2. Verify proper cable connections.

8.8 ENAB -- NON-EXISTENT GPIB BOARD

Cause: GPIB board is not recognized.

Solution: 1. Make sure that GPIB board settings and *CBCONF* setup parameters agree.

8.9 EOIP -- FUNCTION NOT ALLOWED WHILE ASYNCHRONOUS I/O IN PROGRESS

Cause: An illegal call is made during an asynchronous I/O operation.

Solution: 1. Only **IBSTOP**, **IBWAIT**, and **IBONL** calls are allowed during asynchronous I/O operations.

2. 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 has been reset, and the driver and application are synchronized.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Re-synchronization is successful if a CMPL is returned to **ibsta**. See Appendix ? for further information.

8.10 ECAP -- NO CAPABILITY FOR OPERATION

Cause: A call attempts to make use of a capability which is not supported by this GPIB Interface Board.

Solution:
1. Don't use the call.
2. Upgrade your GPIB board

8.11 EFSO -- FILE SYSTEM ERROR

Cause: A problem was encountered while performing a file operation.

Solution:
1. Make sure that you specified the correct filename. Check the spelling and the path.
2. If you need more room on the disk, delete some files.
3. Make sure that *CBCONF* does not assign a name to a device which is already used by a file.

8.12 EBUS -- COMMAND BYTE TRANSFER ERROR

Cause: A GPIB bus error occurs during a device function.

Solution:
1. Determine which device, if any, is abnormally slow to accept commands and fix the problem with the device.
2. Lengthen the assigned time limit using IBTMO or IBCONF.

8.13 ESTB -- SERIAL POLL STATUS BYTE(S) LOST

Cause: One or more serial poll status bytes received during an automatic serial poll was lost.

Solution:
1. Call **IBRSP** more frequently to read the status bytes.
2. Ignore the ESTB error.

NOTE: This error occurs only during *IBRSP* functions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8.14 ESRQ -- SRQ IN "ON" POSITION

Cause: A wait for an RQS can not occur because the GPIB SRQ line is ON.

Solution:

1. Ignore the ESRQ until all devices are found.
2. Make sure an **IBFIND** is called to open all devices on the bus capable of asserting SRQ.
3. Test that each device is unasserting SRQ.
4. Check the cabling. Make sure that all devices are attached and that the connectors are seated properly.

8.15 ETAB -- TABLE PROBLEM

Cause: There was a problem with a table used in a **FindLstn** or **FindRQS**.

Solution:

1. If this occurs during a **FindLstn**, this is a warning and not an error. Ignore the warning or make the buffer larger.
2. If this occurs during a **FindRQS**, this indicated that none of the specified devices are requesting services. Make sure that the device list contains the addresses of all on line devices.

8.16 EINT No Interrupt level configured

Cause: The routine you called requires an interrupt and no interrupt level is configured for the board.

Solution: The routine you called requires an interrupt and no interrupt level is configured for the board.

8.17 EWMD Windows not in Enhanced Mode

Cause: The Windows GPIB Library can only run in Windows Enhanced Mode and Windows is currently running in a different mode.

Solution: Restart Windows is enhanced mode or upgrade to a newer version of Windows.

8.18 EVDD CBGPIB.386 is not installed

Cause: The Windows GPIB Library requires that the CBGPIB.386 device driver be installed - and it isn't.

Solution: Edit your Windows SYSTEM.INI file. It is normally located in the \windows directory. Add the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

following entry to the section that begins with [386Enh]
device=\gpib\cbgpib.386
If your software was not installed in the \gpib directory you will have to change that line to the appropriate directory.

8.19 EOVR Buffer Overflow

Cause: Abuffer was not large enough to hold the data

Solution: Increase the size of the buffer

8.20 ESML Library calls running simultaneously

Cause: The GPIB Library can only execute one function at a time and an attempt was made to call a GPIB routine while another routine was still executing. The most common causes are that more than one GPIB program is being executed or GPIB routines are being called from timer event handlers.

Solution: If more than one GPIB program is running than stop one of them. If you are calling GPIB routines from within Windows timer event handlers, add code to detect and prevent the problem. A common solution is to use a semaphore variable. In your main code set a variable to some known value before you begin executing GPIB routines, clear it when you finish. In your timer event handler, only call the GPIB routine if the semaphore is cleared. Refer to the VBDEMO example program's use of the GPIBCallInProgress variable for a demonstration of how to use this technique.

8.21 ECFG Board type does not match GPIB.CFG

Cause: The board type that is specified in the GPIB.CFG file is different than the board that is installed in the computer.

Solution: Run CBCONF and set the board type on the Board Options menu to the correct board type.

8.22 ETMR No Windows timers available

Cause: Windows has a limited number of system timers. The GPIB Library requested a timer and Windows reported that there were none left.

Solution: Stop any running Windows programs that you are not using.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8.23 EBRK Control-Break keys pressed

Cause: The program was terminated by the user pressing the Control-Break key.

Solution: ---

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP