

## COMPREHENSIVE SERVICES

We offer competitive repair and calibration services, as well as easily accessible documentation and free downloadable resources.

## SELL YOUR SURPLUS

We buy new, used, decommissioned, and surplus parts from every NI series. We work out the best solution to suit your individual needs.

 Sell For Cash    Get Credit    Receive a Trade-In Deal

## OBSOLETE NI HARDWARE IN STOCK & READY TO SHIP

We stock **New**, **New Surplus**, **Refurbished**, and **Reconditioned** NI Hardware.



*Bridging the gap between the manufacturer and your legacy test system.*

 1-800-915-6216

 [www.apexwaves.com](http://www.apexwaves.com)

 [sales@apexwaves.com](mailto:sales@apexwaves.com)

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

**Request a Quote**

 **CLICK HERE**

**VXIcpu-030**

# **GPIB**

---

## **GPIB Reference Manual for VXIpc™ Embedded Controllers and VxWorks**

**Internet Support**

E-mail: [support@natinst.com](mailto:support@natinst.com)

FTP Site: <ftp.natinst.com>

Web Address: <http://www.natinst.com>

**Bulletin Board Support**

BBS United States: 512 794 5422

BBS United Kingdom: 01635 551422

BBS France: 01 48 65 15 59

**Fax-on-Demand Support**

512 418 1111

**Telephone Support (USA)**

Tel: 512 795 8248

Fax: 512 794 5678

**International Offices**

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,  
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,  
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,  
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,  
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,  
United Kingdom 01635 523545

**National Instruments Corporate Headquarters**

6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

# Important Information

---

## Warranty

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

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

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

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

## Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

## Trademarks

HS488™, NAT4882™, NI-488™, NI-488.2™, NI-488DDK™, TNT4882C™, Turbo488™, and VXIpc™ are trademarks of National Instruments Corporation.

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

## WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

# Contents

---

## About This Manual

Organization of This Manual .....	ix
Conventions Used in This Manual.....	x
Related Documentation.....	x
Customer Communication .....	xi

## Chapter 1

### Introduction

GPIB Overview.....	1-1
Talkers, Listeners, and Controllers.....	1-1
Controller-In-Charge and System Controller .....	1-1
GPIB Addressing.....	1-2
Sending Messages across the GPIB .....	1-2
Data Lines .....	1-2
Handshake Lines .....	1-3
Interface Management Lines.....	1-3
Setting up and Configuring Your System.....	1-4
Controlling More Than One Board .....	1-4
Configuration Requirements .....	1-5

## Chapter 2

### Developing Your Application with the VxWorks Drivers

Using the VxWorks NI-488 and ESP-488 Drivers .....	2-1
Using VxWorks NI-488 Functions .....	2-2
Items to Include in Your Application .....	2-2
Checking Status with Global Variables .....	2-3
Status Word (ibsta).....	2-3
Error Variable (iberr).....	2-4
Count Variables (ibcnt and ibcntl) .....	2-5
Compiling and Linking Your Application .....	2-5
Debugging Considerations .....	2-6
Using the Global Status Variables.....	2-6
Configuration Errors.....	2-6
Timing Errors .....	2-7
Communication Errors .....	2-7
Repeat Addressing .....	2-7
Termination Method .....	2-7

## Chapter 3 NI-488 Functions

Function Names .....	3-1
Purpose .....	3-1
Format .....	3-1
Input and Output .....	3-1
Description .....	3-1
Examples .....	3-1
Possible Errors .....	3-2
List of NI-488 Functions .....	3-2
IBCAC .....	3-4
IBCMD .....	3-5
IBEOS .....	3-6
IBEOT .....	3-8
IBFIND .....	3-9
IBGTS .....	3-10
IBIST .....	3-11
IBLINES .....	3-12
IBLN .....	3-14
IBLOC .....	3-15
IBONL .....	3-16
IBPAD .....	3-17
IBPPC .....	3-18
IBRD .....	3-19
IBRPP .....	3-20
IBRSC .....	3-21
IBRSV .....	3-22
IBSAD .....	3-23
IBSIC .....	3-24
IBSRE .....	3-25
IBTMO .....	3-26
IBWAIT .....	3-28
IBWRT .....	3-30

## Chapter 4 GPIB Programming Techniques

Termination of Data Transfers .....	4-1
Waiting for GPIB Conditions .....	4-2
Talker/Listener Applications .....	4-2
Serial Polling .....	4-2
Service Requests from IEEE 488 Devices .....	4-3
Service Requests from IEEE 488.2 Devices .....	4-3

SRQ and Serial Polling with NI-488 Functions .....	4-3
Parallel Polling.....	4-4
Implementing a Parallel Poll with NI-488 Functions.....	4-4

## **Appendix A Multiline Interface Messages**

## **Appendix B Status Word Conditions**

ERR.....	B-2
TIMO .....	B-2
END .....	B-2
SRQI .....	B-2
CMPL.....	B-2
LOK .....	B-3
REM.....	B-3
CIC.....	B-3
ATN .....	B-3
TACS .....	B-4
LACS .....	B-4
DTAS .....	B-4
DCAS.....	B-4

## **Appendix C Error Codes and Solutions**

EDVR (0).....	C-2
ECIC (1).....	C-2
ENOL (2).....	C-3
EADR (3).....	C-3
EARG (4).....	C-4
ESAC (5).....	C-4
EABO (6).....	C-4
ENEB (7) .....	C-5
ECAP (11).....	C-5

## **Appendix D Customer Communication**

## Glossary

## Index

## Figures

Figure 1-1.	GPIB Address Bits .....	1-2
Figure 1-2.	Linear and Star System Configuration .....	1-4
Figure 1-3.	Example of Multiboard System Setup .....	1-5

## Tables

Table 1-1.	GPIB Handshake Lines .....	1-3
Table 1-2.	GPIB Interface Management Lines .....	1-3
Table 2-1.	Status Word Layout .....	2-3
Table 3-1.	NI-488 Functions .....	3-2
Table 3-2.	EOS Configurations .....	3-6
Table 3-3.	Timeout Code Values .....	3-26
Table 3-4.	Wait Mask Layout .....	3-28
Table A-1.	Multiline Interface Messages .....	A-2
Table B-1.	Status Word Layout .....	B-1
Table C-1.	GPIB Error Codes .....	C-1



# About This Manual

---

This manual describes the features and functions of the GPIB driver software for VxWorks and VXIpc embedded controllers. This manual assumes that you are already familiar with operating system fundamentals and VxWorks device driver development concepts.

## Organization of This Manual


---

This manual is organized as follows:

- Chapter 1, *Introduction*, gives an overview of GPIB and describes how to set up your VxWorks system.
- Chapter 2, *Developing Your Application with the VxWorks Drivers*, explains how to use the VxWorks GPIB drivers, and how to develop a GPIB application using the NI-488 API.
- Chapter 3, *NI-488 Functions*, lists the NI-488 functions and describes the purpose, format, input and output parameters, and possible errors for each function.
- Chapter 4, *GPIB Programming Techniques*, describes techniques for using some NI-488 functions in your application.
- Appendix A, *Multiline Interface Messages*, contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions.
- Appendix B, *Status Word Conditions*, describes the conditions reported in the status word, `ibsta`.
- Appendix C, *Error Codes and Solutions*, describes each error, including conditions under which it might occur and possible solutions.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

## Conventions Used in This Manual

---

	The following conventions are used in this manual.
	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
<b><i>bold italic</i></b>	Bold italic text denotes a note, caution, or warning.
IEEE 488 and IEEE 488.2	<i>IEEE 488</i> and <i>IEEE 488.2</i> refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.
<i>italic</i>	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text for which you supply the appropriate word or value.
monospace	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and for statements and comments taken from programs.
<b>monospace bold</b>	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
<i>monospace italic</i>	Italic text in this font denotes that you must enter the appropriate words or values in the place of these items.
paths	Paths in this manual are denoted using backslashes (\) or forward slashes (/) to separate drive names, directories, folders, and files.

## Related Documentation

---

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

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*

# Customer Communication

---

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix D, *Customer Communication*, at the end of this manual.

---

# Introduction

This chapter gives an overview of GPIB and describes how to set up your VxWorks system.

## GPIB Overview

---

The ANSI/IEEE Standard 488.1-1987, also known as GPIB (General Purpose Interface Bus), describes a standard interface for communication between instruments and controllers from various vendors. It contains information about electrical, mechanical, and functional specifications. The GPIB is a digital, 8-bit parallel communications interface with data transfer rates of 1 Mbyte/s and above, using a 3-wire handshake. The bus supports one System Controller, usually a computer, and up to 14 additional instruments. The ANSI/IEEE Standard 488.2-1992 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands.

### Talkers, Listeners, and Controllers

GPIB devices can be Talkers, Listeners, or Controllers. A Talker sends out data messages. Listeners receive data messages. The Controller, usually a computer, manages the flow of information on the bus. It defines the communication links and sends GPIB commands to devices.

Some devices are capable of playing more than one role. A digital voltmeter, for example, can be a Talker and a Listener. If your computer has a National Instruments GPIB interface board and NI-488 software installed, it can function as a Talker, Listener, and Controller.

### Controller-In-Charge and System Controller

You can have multiple Controllers on the GPIB, but only one Controller at a time can be the active Controller, or Controller-In-Charge (CIC). The CIC can either be active or inactive (Standby) Controller. Control can pass from the current CIC to an idle Controller, but only the System Controller, usually a GPIB interface board, can make itself the CIC.

## GPIB Addressing

All GPIB devices and boards must be assigned a unique GPIB address. A GPIB address is made up of two parts: a primary address and an optional secondary address.

The primary address is a number in the range 0 to 30. The GPIB Controller uses this address to form a talk or listen address that is sent over the GPIB when communicating with a device.

A talk address is formed by setting bit 6, the TA (Talk Active) bit of the GPIB address. A listen address is formed by setting bit 5, the LA (Listen Active) bit of the GPIB address. For example, if a device is at address 1, the Controller sends hex 41 (address 1 with bit 6 set) to make the device a Talker. Because the Controller is usually at primary address 0, it sends hex 20 (address 0 with bit 5 set) to make itself a Listener. Figure 1-1 shows the configuration of the GPIB address bits.

<b>Bit Position</b>	7	6	5	4	3	2	1	0
<b>Meaning</b>	0	TA	LA	GPIB Primary Address (range 030)				

**Figure 1-1.** GPIB Address Bits

With some devices, you can use secondary addressing. A secondary address is a number in the range hex 60 to hex 7E. When secondary addressing is in use, the Controller sends the primary talk or listen address of the device followed by the secondary address of the device.

## Sending Messages across the GPIB

Devices on the bus communicate by sending messages. Signals and lines transfer these messages across the GPIB interface, which consists of 16 signal lines and eight ground return (shield drain) lines. The 16 signal lines are discussed in the following sections

### Data Lines

Eight data lines, DIO1 through DIO8, carry both data and command messages.

## Handshake Lines

Three hardware handshake lines asynchronously control the transfer of message bytes between devices. This process is a three-wire interlocked handshake, and it guarantees that devices send and receive message bytes on the data lines without transmission error. Table 1-1 summarizes the GPIB handshake lines.

**Table 1-1.** GPIB Handshake Lines

Line	Description
NRFD (not ready for data)	Listening device is ready/not ready to receive a message byte. Also used by the Talker to signal high-speed GPIB transfers.
NDAC (not data accepted)	Listening device has/has not accepted a message byte.
DAV (data valid)	Talking device indicates signals on data lines are stable (valid) data.

## Interface Management Lines

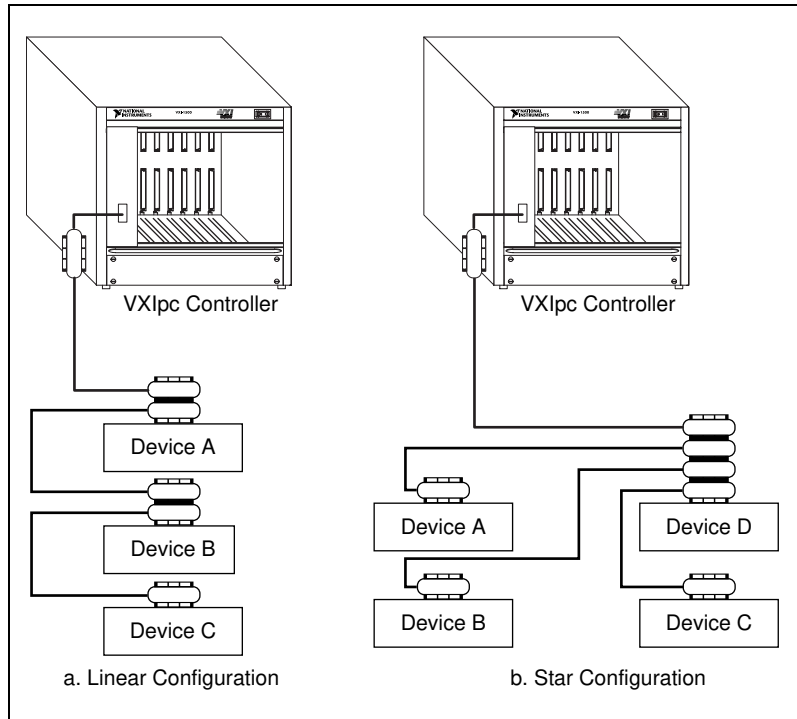
Five GPIB hardware lines manage the flow of information across the bus. Table 1-2 summarizes the GPIB interface management lines.

**Table 1-2.** GPIB Interface Management Lines

Line	Description
ATN (attention)	Controller drives ATN true when it sends commands and false when it sends data messages.
IFC (interface clear)	System Controller drives the IFC line to initialize the bus and make itself CIC.
REN (remote enable)	System Controller drives the REN line to place devices in remote or local program mode.
SRQ (service request)	Any device can drive the SRQ line to asynchronously request service from the Controller.
EOI (end or identify)	Talker uses the EOI line to mark the end of a data message. Controller uses the EOI line when it conducts a parallel poll.

# Setting up and Configuring Your System

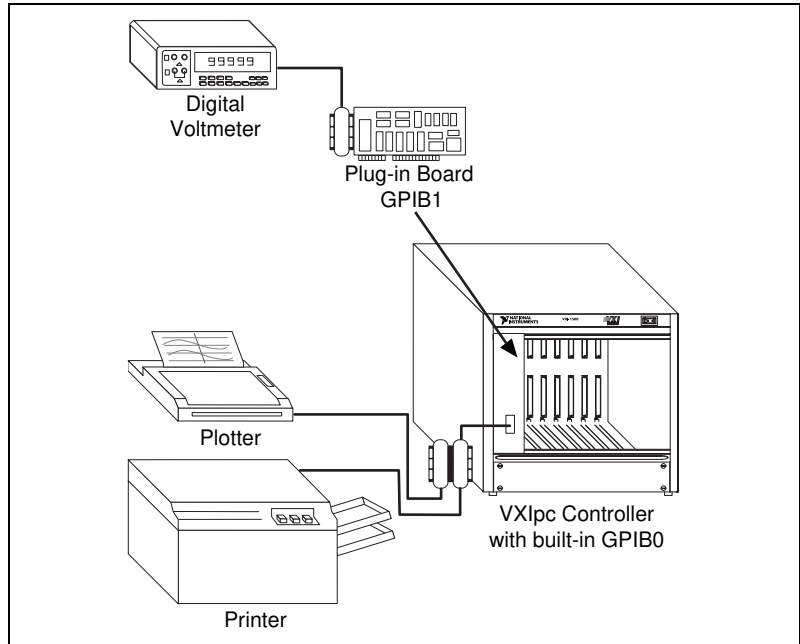
Devices are usually connected with a cable assembly consisting of a shielded 24-conductor cable with both a plug and receptacle connector at each end. With this design, you can link devices in a linear configuration, a star configuration, or a combination of the two. Figure 1-2 shows the linear and star configurations.



**Figure 1-2.** Linear and Star System Configuration

## Controlling More Than One Board

Figure 1-3 shows an example of a multiboard system configuration. `gpib0` is the access board for the plotter and printer, and `gpib1` is the access board for the voltmeter. The control functions of the devices automatically access their respective boards.



**Figure 1-3.** Example of Multiboard System Setup

## Configuration Requirements

To achieve the high data transfer rate that the GPIB was designed for, you must limit the physical distance between devices and the number of devices on the bus. The following restrictions are typical:

- A maximum separation of four meters between any two devices and an average separation of two meters over the entire bus
- A maximum total cable length of 20 m
- A maximum of 15 devices connected to each bus, with at least two-thirds powered on

For high-speed operation, the following restrictions apply:

- All devices in the system must be powered on
- Cable lengths as short as possible up to a maximum of 15 m of cable for each system
- With at least one equivalent device load per meter of cable

If you want to exceed these limitations, you can use bus extenders to increase the cable length or expanders to increase the number of device loads. Extenders and expanders are available from National Instruments.



---

# Developing Your Application with the VxWorks Drivers

This chapter explains how to use the VxWorks GPIB drivers, and how to develop a GPIB application using the NI-488 API.

## Using the VxWorks NI-488 and ESP-488 Drivers

---

The GPIB drivers that come with the VxWorks-based controller provide two different ways to use the GPIB functions. The NI-488 driver is derived from the NI-488DDK, and offers a 100% compatible subset of the full NI-488 drivers used on other operating systems. The ESP-488 driver gives users of the VXIcpu-030 an easy way to port their existing GPIB programs to newer hardware by providing a mapping of ESP-488 functions to NI-488 functions.

National Instruments strongly recommends you use the ESP-488 driver only for porting applications from the VXIcpu-030 to a new controller. The ESP-488 driver is merely a mapping of the corresponding NI-488 functions, and using the actual NI-488 functions will make it easier to transfer programs between a VxWorks system and other operating systems that use the standard NI-488 driver. Furthermore, the NI-488 driver offers support for multithreading, which is not available in the ESP-488 driver. We wish to emphasize that the ESP-488 library is provided only for the convenience of users making the transition from the VXIcpu-030 to a newer controller.

Users of the VXIcpu-030 should note that under certain error conditions, the new ESP-488 library might behave slightly differently from the ESP-488 library provided with the VXIcpu-030. This is because of the underlying NI-488 driver, but should not adversely affect user code under normal circumstances.

The GPIB library files provided with the VxWorks-based controller are:

- `nigpib.o`—the NI-488 library
- `espdpib.o`—the ESP-488 library

- `ibic.o`—the old VXIcpu-030 GPIB interactive control program, which requires the ESP-488 library
- `ugplib.h`—the header file for the NI-488 API
- `esp488.h`—the header file for the ESP-488 API (refer to [Compiling and Linking Your Application](#), later in this chapter, for more information about how to use this file)
- `ni4882.c`—source code for the 488.2 extensions library (in `util` subdirectory)
- `ni4882.h`—the header file for 488.2 extensions (in `util` subdirectory)
- `README.TXT`—a text file with further important information about the GPIB libraries

## Using VxWorks NI-488 Functions

---

The VxWorks NI-488 functions perform basic GPIB operations. These low-level functions access the interface board directly and require you to handle the addressing and bus management protocol. In addition to serving as a foundation upon which you can implement higher-level functions, these functions give you the flexibility and control to handle situations such as the following:

- Communicating with non-compliant (non-IEEE 488.2) devices
- Altering various low-level board configurations
- Developing non-controller applications
- Managing the bus in non-typical ways

The VxWorks NI-488 functions are completely compatible with the corresponding functions of standard NI-488.2 drivers from National Instruments.

## Items to Include in Your Application

---

Items you should include in your C application programs are as follows:

- The header file `ugplib.h` contains prototypes for the GPIB functions and constants that you can use in your application.
- One or more calls to the `ibfind` function to obtain a unit descriptor for each GPIB board that the application uses.
- Code to check for errors after each NI-488 function call.

- A function to handle GPIB errors. This function takes the board offline and closes the application. If the function is declared as:

```
void gpiberr (char * msg); /*function prototype*/
```

then your application invokes it as follows:

```
if (ibsta & ERR) {
    gpiberr("GPIB error");
}
```

## Checking Status with Global Variables

---

Each NI-488 function updates four global variables to reflect the status of the board that you are using. These global status variables are the status word (`ibsta`), the error variable (`iberr`) and the count variables (`ibcnt` and `ibcnt1`). They contain useful information about the performance of your application. Your application should check these variables after each GPIB call. The following sections describe each of these global variables and how you can use them in your application.

### Status Word (`ibsta`)

All functions update a global status word, `ibsta`, which contains information about the state of the GPIB and the GPIB hardware. The value stored in `ibsta` is the return value of all of the NI-488 functions except `ibfind`. You can examine various status bits in `ibsta` and use that information to make decisions about continued processing. If you check for possible errors after each call using the `ibsta` `ERR` bit, debugging your application is much easier.

`ibsta` is an integer-sized value. The least significant 16 bits of `ibsta` are meaningful. A bit value of one (1) indicates that a certain condition is in effect, and a bit value of zero (0) indicates that the condition is not in effect.

Table 2-1 shows the condition that each bit position represents and the bit mnemonics. For a detailed explanation of each of the status conditions, refer to Appendix B, *Status Word Conditions*.

**Table 2-1.** Status Word Layout

Mnemonic	Bit Pos.	Hex Value	Description
ERR	15	8000	GPIB error
TIMO	14	4000	Time limit exceeded

**Table 2-1.** Status Word Layout (Continued)

Mnemonic	Bit Pos.	Hex Value	Description
END	13	2000	END or EOS detected
SRQI	12	1000	SRQ interrupt received
CMPL	8	100	I/O completed
LOK	7	80	Lockout State
REM	6	40	Remote State
CIC	5	20	Controller-In-Charge
ATN	4	10	Attention is asserted
TACS	3	8	Talker
LACS	2	4	Listener
DTAS	1	2	Device Trigger State
DCAS	0	1	Device Clear State

The application header file `ugpib.h` included on your distribution medium defines each of the `ibsta` status bits. You can test for an `ibsta` status bit being set using the bitwise and operator (`&` in C/C++). For example, the `ibsta` `ERR` bit is bit 15 of `ibsta`. To check for a GPIB error, use the following statement after each GPIB call as shown:

```
if (ibsta & ERR)
    printf("GPIB error encountered");
```

## Error Variable (`iberr`)

If the `ERR` bit is set in `ibsta`, a GPIB error has occurred. When an error occurs, the error type is specified by the integer `iberr`. To check for a GPIB error, use the following statement after each GPIB call:

```
if (ibsta & ERR)
    printf("GPIB error %d encountered", iberr);
```



### Note

***The value in `iberr` is meaningful as an error type only when the `ERR` bit is set in `ibsta`, indicating that an error has occurred.***

For more information on error codes and solutions refer to the [Debugging Considerations](#) section later in this chapter, or Appendix C, [Error Codes and Solutions](#).

## Count Variables (`ibcnt` and `ibcntl`)

The count variables are updated after each read, write, or command function. `ibcnt` is an integer value and `ibcntl` is a long integer value. As implemented on most modern systems today, `ibcnt` and `ibcntl` are both 32-bit integers. On some older systems, such as MS-DOS, `ibcnt` is a 16-bit integer; on some newer systems, `ibcntl` is a 64-bit integer. For cross-platform compatibility, all applications should use `ibcntl`. If you are reading data, the count variables indicate the number of bytes read. If you are sending data or commands, the count variables reflect the number of bytes sent.

In your application you can use the count variables to null-terminate an ASCII string of data received from an instrument. For example, if data is received in an array of characters, you can use `ibcntl` to null-terminate the array and print the measurement on the screen as follows:

```
char rdbuf[512];
ibrd (ud, rdbuf, 20L);
if (!(ibsta & ERR)){
    rdbuf[ibcntl] = '\0';
    printf ("Read: %s\n", rdbuf);
}
else {
    error();
}
```

## Compiling and Linking Your Application

---

When writing NI-488 or ESP-488 programs, you need to include the appropriate header file, which depends on which GPIB library you are using. Normally, you will use NI-488, and the default setup uses the NI-488 version of `ugpib.h`. If you need the ESP-488 API instead, rename the default `ugpib.h` to `ni488.h` and copy the file `esp488.h` to `ugpib.h`. Always use the directive `#include "ugpib.h"` in your programs, whether they are NI-488 or ESP-488 based.

Compiling a program for either API works no differently from compiling other programs for VxWorks. If you are unfamiliar with VxWorks, the process is very similar to compilation on other systems. You can find detailed information in the Wind River Systems programming documentation. You may find the *Intel x86* appendix of the *VxWorks Programmer's Guide* and the *gcc/g++* chapter of the *GNU Toolkit User's Guide* particularly helpful.

On VxWorks, the functions in the NI-488 (or ESP-488) library are automatically linked into your program as long as the appropriate library is already loaded when you load your NI-488 or ESP-488 application into VxWorks. To load the library, change to the directory in which the libraries are installed (by default, /ide0) and use the command `ld < nigpib.o` or `ld < espgpib.o`.

Refer to the README.TXT file for more information about the two libraries. As noted at the beginning of this chapter, National Instruments strongly recommends you use the NI-488 driver unless you need the ESP-488 driver for backward compatibility with the VXIcpu-030.

## Debugging Considerations

---

This section contains typical errors you may encounter and some considerations for debugging your application.

### Using the Global Status Variables

After each function call to your NI-488 driver, `ibsta`, `iberr`, `ibcnt`, and `ibcnt1` are updated before the call returns to your application. You should check for an error after each GPIB call. Refer to the [Checking Status with Global Variables](#) section earlier in this chapter for more information about how to use these variables within your program to automatically check for errors.

After you determine which GPIB call is failing and note the corresponding values of the global variables, refer to Appendix B, [Status Word Conditions](#), and Appendix C, [Error Codes and Solutions](#). These appendices can help you interpret the state of the driver.

### Configuration Errors

Some applications require customized configuration of the GPIB driver. For example, you might want to terminate reads on a special end-of-string character, or you might require secondary addressing. In these cases, you can temporarily reconfigure the driver while your application is running using the `ibeos` and `ibsad` functions.

Refer to the descriptions of these functions and others in Chapter 3, [NI-488 Functions](#), for more information.

## Timing Errors

In some cases, your application might fail because it is issuing the NI-488 calls too quickly for your device to process and respond to them. This problem can also result in corrupted or incomplete data.

A well behaved IEEE 488 device should hold off handshaking and set the appropriate transfer rate. If your device is not well behaved, you can test for and resolve the timing error by single-stepping through your program and inserting finite delays between each GPIB call. One way to do this is to have your device communicate its status whenever possible. Although this method is not possible with many devices, it is usually the best option. Your delays are controlled by the device and your application can adjust itself and work independently on any platform. Other delay mechanisms might cause varying delay times on different platforms.

## Communication Errors

### Repeat Addressing

Devices adhering to the IEEE 488.2 standard should remain in their current state until specific commands are sent across the GPIB to change their state. However, some devices require GPIB addressing before any GPIB activity. Therefore, you might need to make additional calls to `ibcmd` in your application to perform repeat addressing if your device does not remain in its currently addressed state.

### Termination Method

You should learn the data termination method that your devices use. By default, your NI-488 software sends EOI on writes and terminates reads on EOI or a specific byte count. If you send a command string to your device and it does not respond, it might be because it does not recognize the end of the command. You might need to send a termination message such as CR LF after a write command as follows:

```
ibwrt (ud, "COMMAND\x0D\x0A", 9);
```

---

# NI-488 Functions

This chapter lists the NI-488 functions and describes the purpose, format, input and output parameters, and possible errors for each function.

## Function Names

---

The functions in this chapter are listed alphabetically.

## Purpose

---

Each function description includes a brief statement of the purpose of the function.

## Format

---

The format section describes the format of each function in the C programming language.

## Input and Output

---

The input and output parameters for each function are listed. *Function Return* describes the return value of the function.

## Description

---

The description section gives details about the purpose and effect of each function.

## Examples

---

Some function descriptions include sample code showing how to use the function. For more detailed and complete examples, refer to the source code support files included in the `util` directory with your VxWorks GPIB software.



## Possible Errors

---

Each function description includes a list of errors that could occur when it is invoked.

## List of NI-488 Functions

---

The following table contains an alphabetical list of the VxWorks NI-488 functions.

**Table 3-1.** NI-488 Functions

<b>Function</b>	<b>Purpose</b>
ibcac	Become Active Controller
ibcmd	Send GPIB commands
ibeos	Configure the end-of-string (EOS) termination mode or character
ibeot	Enable or disable the automatic assertion of the GPIB EOI line at the end of write I/O operations
ibfind	Open and initialize a GPIB board
ibgts	Go from Active Controller to Standby
ibist	Set or clear the board individual status bit for parallel polls
iblines	Return the status of the eight GPIB control lines
ibltn	Check for the presence of a device on the bus
ibloc	Go to local
ibonl	Place the interface board online or offline
ibpad	Change the primary address
ibppc	Parallel poll configure
ibrd	Read data into a user buffer
ibrpp	Conduct a parallel poll
ibrsc	Request or release system control
ibrsv	Request service and change the serial poll status byte
ibsad	Change or disable the secondary address
ibsic	Assert interface clear

**Table 3-1.** NI-488 Functions (Continued)

<b>Function</b>	<b>Purpose</b>
ibsre	Set or clear the Remote Enable (REN) line
ibtmo	Change or disable the I/O timeout period
ibwait	Wait for GPIB events
ibwrt	Write data from a user buffer

## IBCAC

---

### Purpose

Become Active Controller.

### Format

```
int ibcac (int ud, int v)
```

### Input

<code>ud</code>	A board unit descriptor
<code>v</code>	Determines if control is to be taken asynchronously or synchronously

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

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

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

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

### Possible Errors

EARG	<code>ud</code> is valid but does not refer to an interface board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBCMD

---

## Purpose

Send GPIB commands.

## Format

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

## Input

<code>ud</code>	A board unit descriptor
<code>cmdbuf</code>	Buffer of command bytes to send
<code>count</code>	Number of command bytes to send

## Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

## Description

`ibcmd` sends `count` bytes from `cmdbuf` over the GPIB as command bytes (interface messages). The number of command bytes transferred is returned in the global variable, `ibcnt1`. Refer to Appendix A, *Multiline Interface Messages*, for a table of the defined interface messages.

Command bytes configure the state of the GPIB, such as addressing devices to listen or talk.

## Possible Errors

EABO	The timeout period expired before all of the command bytes were sent.
EARG	<code>ud</code> is valid but does not refer to an interface board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners are on the GPIB.

## IBEOS

---

### Purpose

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

### Format

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

### Input

ud	Board descriptor
v	EOS mode and character information

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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



**Note** *Defining an EOS byte does not cause the driver to automatically send that byte at the end of write I/O. Your application is responsible for placing the EOS byte at the end of the data strings that it defines.*

Table 3-2 describes the different EOS configurations and the corresponding values of `v`. If no error occurs during the call, the value of the previous EOS setting is returned in `iberr`.

**Table 3-2.** EOS Configurations

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

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

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

For more information on the termination of I/O operations, refer to Chapter 4, [GPIB Programming Techniques](#).

## Examples

```

ibeos (ud, 0x140A); /* Configure the software to end reads on
                    newline character (hex 0A) for the unit
                    descriptor, ud */

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

```

## Possible Errors

EARG	The high byte of <i>v</i> contains invalid bits.
EDVR	Either <i>ud</i> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBEOT

---

### Purpose

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

### Format

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

### Input

ud	Board descriptor
v	Enables or disables the end of transmission assertion of EOI

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

For more information on the termination of I/O operations, refer to Chapter 4, [GPIB Programming Techniques](#).

### Possible Errors

EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBFIND

---

## Purpose

Open and initialize a GPIB board descriptor.

## Format

```
int ibfind (char *udname)
```

## Input

udname                      A GPIB board name

## Output

Function Return              The board descriptor, or **-1**

## Description

`ibfind` acquires a descriptor for a GPIB board; this board descriptor can be used in subsequent NI-488 functions.

`ibfind` performs the equivalent of an `ibonl 1` to initialize the board descriptor. The unit descriptor that `ibfind` returns remains valid until you use `ibonl 0` to put the board offline.

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

## Possible Errors

EDVR	Either <code>udname</code> is not recognized as a board name or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.



## IBGTS

---

### Purpose

Go from Active Controller to Standby.

### Format

```
int ibgts (int ud, int v)
```

### Input

ud	Board descriptor
v	Determines whether to perform acceptor handshaking

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

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

For details on the IEEE-488.1 handshake protocol, refer to ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*.

### Possible Errors

EADR	<code>v</code> is non-zero, and either ATN is low or the interface board is a Talker or Listener.
EARG	<code>ud</code> is valid but does not refer to an interface board.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBIST

---

## Purpose

Set or clear the board individual status bit for parallel polls.

## Format

```
int ibist (int ud, int v)
```

## Input

<code>ud</code>	Board descriptor
<code>v</code>	Indicates whether to set or clear the <code>ist</code> bit

## Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

## Description

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

For more information on parallel polling, refer to Chapter 4, [GPIB Programming Techniques](#).

## Possible Errors

EARG	<code>ud</code> is valid but does not refer to an interface board.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBLINES

---

### Purpose

Return the status of the eight GPIB control lines.

### Format

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

### Input

ud                                      Board descriptor

### Output

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

### Description

`iblines` returns the state of the GPIB control lines in `clines`. The low-order byte (bits 0 through 7) of `clines` contains a mask indicating the capability of the GPIB interface 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 following is a pattern of each byte.

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

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

### Example

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

## Possible Errors

EARG	<code>ud</code> is valid but does not refer to an interface board.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBLN

---

### Purpose

Check for the presence of a device on the bus.

### Format

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

### Input

ud	Board descriptor
pad	The primary GPIB address of the device
sad	The secondary GPIB address of the device

### Output

listen	Indicates if a device is present or not
Function Return	The value of <code>ibsta</code>

### Description

`ibln` determines whether there is a listening device at the GPIB address designated by the `pad` and `sad` parameters. If a Listener is detected, a non-zero value is returned in `listen`. If no Listener is found, zero is returned.

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

### Possible Errors

EARG	Either the <code>pad</code> or <code>sad</code> argument is invalid.
ECIC	The interface board is not Controller-In-Charge.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBLOC

---

## Purpose

Go to Local.

## Format

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

## Input

ud                                      Board descriptor

## Output

Function Return                      The value of `ibsta`

## Description

`ibloc` places the board in local mode if it is not in a lockout state. The board is in a lockout state if LOK appears in the status word `ibsta`. If the board is in a lockout state, the call has no effect.

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

## Possible Errors

EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBONL

---

### Purpose

Place the interface board online or offline.

### Format

```
int ibonl (int ud, int v)
```

### Input

ud	Board descriptor
v	Indicates whether the board is to be taken online or offline

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

If an interface board is taken offline, the board descriptor (`ud`) is no longer valid. You must execute an `ibfind` to access the board again.

### Possible Errors

EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBPAD

---

## Purpose

Change the primary address.

## Format

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

## Input

ud	Board descriptor
v	GPIB primary address

## Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

## Description

`ibpad` sets the primary GPIB address of the board to `v`, an integer ranging from 0 to 30. If no error occurs during the call, then `iberr` contains the previous GPIB primary address.

## Possible Errors

EARG	<code>v</code> is not a valid primary GPIB address; it must be in the range 0 to 30.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.



## IBPPC

---

### Purpose

Parallel poll configure.

### Format

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

### Input

ud	Board descriptor
v	Parallel poll enable/disable value

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

For more information on parallel polling, refer to Chapter 4, [GPIB Programming Techniques](#).

### Possible Errors

EARG	<code>v</code> does not contain a valid PPE or PPD message.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBRD

---

## Purpose

Read data into a user buffer.

## Format

```
int ibrd (int ud, void *rdbuf, long count)
```

## Input

<code>ud</code>	Board descriptor
<code>count</code>	Number of bytes to be read from the GPIB

## Output

<code>rdbuf</code>	Address of buffer into which data is read
Function Return	The value of <code>ibsta</code>

## Description

`ibrd` reads up to `count` bytes of data and places the data into the buffer specified by `rdbuf`. `ibrd` assumes that the GPIB is already properly addressed. The operation terminates normally when `count` bytes have been received or END is received. The operation terminates with an error if the transfer could not complete within the timeout period or, if the board is not CIC, the CIC sends a Device clear on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcnt1`.

## Possible Errors

EABO	Either <code>count</code> bytes or END was not received within the timeout period or a Device Clear message was received after the read operation began.
EADR	The GPIB is not correctly addressed; use <code>ibcmd</code> to address the GPIB.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBRPP

---

### Purpose

Conduct a parallel poll.

### Format

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

### Input

ud                                      Board descriptor

### Output

ppr                                      Parallel poll response byte  
Function Return                      The value of `ibsta`

### Description

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

For more information on parallel polling, refer to Chapter 4, [GPIB Programming Techniques](#).

### Possible Errors

ECIC	The interface board is not Controller-In-Charge.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBRSC

---

## Purpose

Request or release system control.

## Format

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

## Input

<code>ud</code>	Board descriptor
<code>v</code>	Determines if system control is to be requested or released

## Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

## Description

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

## Possible Errors

EARG	<code>ud</code> is a valid descriptor but does not refer to a board.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBRSV

---

### Purpose

Request service and change the serial poll status byte.

### Format

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

### Input

<code>ud</code>	Board descriptor
<code>v</code>	Serial poll status byte

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

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

### Possible Errors

EARG	<code>ud</code> is a valid descriptor but does not refer to a board.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

# IBSAD

---

## Purpose

Change or disable the secondary address.

## Format

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

## Input

ud	Board descriptor
v	GPIB secondary address

## Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

## Description

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

## Possible Errors

EARG	<code>v</code> is non-zero and outside the legal range 96 to 126.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBSIC

---

### Purpose

Assert interface clear.

### Format

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

### Input

ud Board descriptor

### Output

Function Return The value of `ibsta`

### Description

`ibsic` asserts the GPIB interface clear (IFC) line for at least 100  $\mu$ s if the GPIB board is System Controller. This initializes the GPIB and makes the interface board CIC and Active Controller with ATN asserted.

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

### Possible Errors

EARG	<code>ud</code> is a valid descriptor but does not refer to a board.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ESAC	The board does not have System Controller capability.

## IBSRE

---

### Purpose

Set or clear the Remote Enable line.

### Format

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

### Input

<code>ud</code>	Board descriptor
<code>v</code>	Indicates whether to set or clear the REN line

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

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

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

### Possible Errors

EARG	<code>ud</code> is a valid descriptor but does not refer to a board.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ESAC	The board does not have System Controller capability.



## IBTMO

---

### Purpose

Change or disable the timeout period.

### Format

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

### Input

ud	Board descriptor
v	Timeout duration code

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

`ibtmo` sets the timeout period of the board to `v`. The timeout period is used to select the maximum duration allowed for an I/O operation (for example, `ibrd` and `ibwrt`) or for an `ibwait` operation with `TIMO` in the wait mask. If the operation does not complete before the timeout period elapses, then the operation is aborted and `TIMO` is returned in `ibsta`. See Table 3-3 for a list of valid timeout values. These timeout values represent the minimum timeout period. The actual period may be longer.

**Table 3-3.** Timeout Code Values

Constant	Value of v	Minimum Timeout
TNONE	0	disabled/no timeout
T10us	1	10
T30us	2	30
T100us	3	100
T300us	4	300
T1ms	5	1 ms
T3ms	6	3 ms
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms

**Table 3-3.** Timeout Code Values (Continued)

Constant	Value of $v$	Minimum Timeout
T300ms	10	300 ms
T1s	11	1 s
T3s	12	3 s
T10s	13	10 s
T30s	14	30 s
T100s	15	100 s
T300s	16	300 s
T1000s	17	1000 s

### Possible Errors

EARG	$v$ is invalid.
EDVR	Either $ud$ is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBWAIT

---

### Purpose

Wait for GPIB events.

### Format

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

### Input

ud	Board descriptor
mask	Bit mask of GPIB events to wait for

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

`ibwait` monitors the events that `mask` specifies and delays processing until one or more of the events occurs. If the wait mask is zero, `ibwait` returns immediately with the updated `ibsta` status word. If `TIMO` is set in the wait mask, `ibwait` returns when the timeout period has elapsed, if one or more of the other specified events have not already occurred. If `TIMO` is not set in the wait mask, then `ibwait` waits indefinitely for one or more of the specified events to occur. The existing `ibwait` mask bits are identical to the `ibsta` bits, and they are described in Table 3-4. You can configure the timeout period using the `ibtmo` function.

**Table 3-4.** Wait Mask Layout

Mnemonic	Bit Pos.	Hex Value	Description
TIMO	14	4000	Use the timeout period (see <code>ibtmo</code> ) to limit the wait period
END	13	2000	END or EOS is detected
SRQI	12	1000	SRQ is asserted
CMPL	8	100	I/O completed
LOK	7	80	GPIB board is in Lockout State
REM	6	40	GPIB board is in Remote State
CIC	5	20	GPIB board is CIC
ATN	4	10	Attention is asserted

**Table 3-4.** Wait Mask Layout (Continued)

<b>Mnemonic</b>	<b>Bit Pos.</b>	<b>Hex Value</b>	<b>Description</b>
TACS	3	8	GPIB board is Talker
LACS	2	4	GPIB board is Listener
DTAS	1	2	GPIB board is in Device Trigger State
DCAS	0	1	GPIB board is in Device Clear State

## Possible Errors

EARG	The bit set in <code>mask</code> is invalid.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.

## IBWRT

---

### Purpose

Write data from a user buffer.

### Format

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

### Input

<code>ud</code>	Board descriptor
<code>wrtbuf</code>	Address of the buffer containing the bytes to write
<code>count</code>	Number of bytes to be written

### Output

Function Return	The value of <code>ibsta</code>
-----------------	---------------------------------

### Description

`ibwrt` writes `count` bytes of data from the buffer specified by `wrtbuf`; `ibwrt` assumes that the GPIB is already properly addressed. The operation terminates normally when `count` bytes have been sent. The operation terminates with an error if `count` bytes could not be sent within the timeout period or, if the board is not CIC, the CIC sends Device Clear on the GPIB. The actual number of bytes transferred is returned in the global variable `ibcnt1`.

### Possible Errors

EABO	Either <code>count</code> bytes were not sent within the timeout period, or a Device Clear message was received after the write operation began.
EADR	The GPIB is not correctly addressed; use <code>ibcmd</code> to address the GPIB.
EDVR	Either <code>ud</code> is invalid or the NI-488 driver is not installed.
ENEB	The interface board is not installed or is not properly configured.
ENOL	No Listeners were detected on the bus.

---

# GPIB Programming Techniques

This chapter describes techniques for using some NI-488 functions in your application.

For more detailed information about each function, refer to Chapter 3, *NI-488 Functions*.

---

## Termination of Data Transfers

GPIB data transfers are terminated either when the GPIB EOI line is asserted with the last byte of a transfer or when a preconfigured end-of-string (EOS) character is transmitted. By default, the NI-488 driver asserts EOI with the last byte of writes and the EOS modes are disabled.

You can use the `ibeot` function to enable or disable the end of transmission (EOT) mode. If EOT mode is enabled, the NI-488 driver asserts the GPIB EOI line when the last byte of a write is sent out on the GPIB. If it is disabled, the EOI line is *not* asserted with the last byte of a write.

You can use the `ibes` function to enable, disable, or configure the EOS modes. EOS mode configuration includes the following information:

- A 7-bit or 8-bit EOS byte
- EOS comparison method—This indicates whether the EOS byte has seven or eight significant bits. For a 7-bit EOS byte, the eighth bit of the EOS byte is ignored.
- EOS write method—If you enable this, the NI-488 driver automatically asserts the GPIB EOI line when the EOS byte is written to the GPIB. If the buffer passed into an `ibwrt` call contains five occurrences of the EOS byte, the EOI line is asserted as each of the five EOS bytes are written to the GPIB. If an `ibwrt` buffer does not contain an occurrence of the EOS byte, the EOI line is not asserted (unless the EOT mode is enabled, in which case the EOI line is asserted with the last byte of the write).
- EOS read method—If you enable this, the NI-488 driver terminates `ibrd` calls when the EOS byte is detected on the GPIB or when the GPIB EOI line is asserted or when the specified count is reached. If

you disable the EOS read method, `ibrd` calls terminate only when the GPIB EOI line is asserted or the specified count has been read.

## Waiting for GPIB Conditions

---

You can use the `ibwait` function to obtain the current `ibsta` value or to suspend your application until a specified condition occurs on the GPIB. If you use `ibwait` with a parameter of zero, it immediately updates `ibsta` and returns. If you want to use `ibwait` to wait for one or more events to occur, then pass a wait mask to the function. The wait mask should always include the TIMO event; otherwise, your application is suspended indefinitely until one of the wait mask events occurs.

## Talker/Listener Applications

---

Although designed for Controller-In-Charge applications, you can also use the NI-488 software in most non-Controller situations. These situations are known as Talker/Listener applications because the interface board is not the GPIB Controller.

A Talker/Listener application typically uses `ibwait` with a mask of 0 to monitor the status of the interface board. Then, based on the status bits set in `ibsta`, the application takes whatever action is appropriate. For example, the application could monitor the status bits TACS (Talker Active State) and LACS (Listener Active State) to determine when to send data to or receive data from the Controller. The application could also monitor the DCAS (Device Clear Active State) and DTAS (Device Trigger Active State) bits to determine if the Controller has sent the device clear (DCL or SDC) or trigger (GET) messages to the interface board. If the application detects a device clear from the Controller, it might reset the internal state of message buffers. If it detects a trigger message from the Controller, the application might begin an operation such as taking a voltage reading if the application is actually acting as a voltmeter.

## Serial Polling

---

You can use serial polling to obtain specific information from GPIB devices when they request service. When the GPIB SRQ line is asserted, it signals the Controller that a service request is pending. The Controller must then determine which device asserted the SRQ line and respond accordingly. The most common method for SRQ detection and servicing

is the serial poll. This section describes how you can set up your application to detect and respond to service requests from GPIB devices.

## Service Requests from IEEE 488 Devices

IEEE 488 devices request service from the GPIB Controller by asserting the GPIB SRQ line. When the Controller acknowledges the SRQ, it serial polls each open device on the bus to determine which device requested service. Any device requesting service returns a status byte with bit 6 set and then unasserts the SRQ line. Devices not requesting service return a status byte with bit 6 cleared. Manufacturers of IEEE 488 devices use lower order bits to communicate the reason for the service request or to summarize the state of the device.

## Service Requests from IEEE 488.2 Devices

The IEEE 488.2 standard refined the bit assignments in the status byte. In addition to setting bit 6 when requesting service, IEEE 488.2 devices also use two other bits to specify their status. Bit 4, the Message Available bit (MAV), is set when the device is ready to send previously queried data. Bit 5, the Event Status bit (ESB), is set if one or more of the enabled IEEE 488.2 events occurs. These events include power-on, user request, command error, execution error, device dependent error, query error, request control, and operation complete. The device can assert SRQ when ESB or MAV are set, or when a manufacturer-defined condition occurs.

## SRQ and Serial Polling with NI-488 Functions

The 488.2 application library included with the NI-488 driver contains some high-level routines that you can use to conduct SRQ servicing and serial polling. Routines pertinent to SRQ servicing and serial polling are `ni4882_ReadStatusByte`, `ni4882_TestSRQ`, and `ni4882_WaitSRQ`.

`ni4882_ReadStatusByte` serial polls a single device and returns its status byte.

`ni4882_TestSRQ` determines whether the SRQ line is asserted or unasserted, and returns to the caller immediately.

`ni4882_WaitSRQ` is similar to `ni4882_TestSRQ`, except that `ni4882_WaitSRQ` suspends the application until either SRQ is asserted or the timeout period is exceeded.



You can also use the IEEE 488.2 routines mentioned in this section to construct your own SRQ servicing routines using the low-level functions of the NI-488 driver. Refer to the file `ni4882.c` for more information.

## Parallel Polling

---

Although parallel polling is not widely used, it is a useful method for obtaining the status of more than one device at the same time. The advantage of parallel polling is that a single parallel poll can easily check up to eight individual devices at once. In comparison, eight separate serial polls would be required to check eight devices for their serial poll response bytes. The value of the individual status bit (`ist`) determines the parallel poll response.

### Implementing a Parallel Poll with NI-488 Functions

Follow these steps to implement parallel polling using NI-488 functions. Each step contains example code.

1. Configure the device for parallel polling using the `ibcmd` function, unless the device can configure itself for parallel polling.

Parallel poll configuration requires an 8-bit value to designate the data line number, the `ist` sense, and whether or not the function configures or unconfigures the device for the parallel poll. The bit pattern is as follows:

0 1 1 E S D2 D1 D0

E is 1 to disable parallel polling and 0 to enable parallel polling for that particular device.

S is 1 if the device is to assert the assigned data line when `ist = 1`, and 0 if the device is to assert the assigned data line when `ist = 0`.

D2 through D0 determine the number of the assigned data line. The physical line number is the binary line number plus one. For example, DIO3 has a binary bit pattern of 010.

The following example code configures a device at primary address 3 for parallel polling using NI-488 functions. The device asserts DIO7 if its `ist` bit = 0, therefore, 0110 0110 (hex 66) is the parallel poll configuration byte.

```
#include "ugpib.h"
char ppr;
ud = ibfind("gpib0");
ibsic(ud);
ibcmd(ud, "?#\x05\x66", 4);
```

If the GPIB interface board configures itself for a parallel poll, you should use the `ibppc` function. Pass the board unit descriptor value as the first argument in `ibppc`. In addition, if the individual status bit (`ist`) of the board needs to be changed, use the `ibist` function.

In the following example, the GPIB board is to configure itself to participate in a parallel poll. It asserts DIO5 when `ist = 1` if a parallel poll is conducted.

```
ibppc(ud, 0x6C);
ibist(ud, 1);
```

2. Conduct the parallel poll using `ibrpp` and check the response for a certain value. The following example code performs the parallel poll and compares the response to hex 10, which corresponds to DIO5. If that bit is set, the `ist` of the device is 1.

```
ibrpp(ud, &ppr);
if (ppr & 0x10) printf("ist = 1\n");
```

3. Unconfigure the device for parallel polling with `ibcmd`. Notice that any value having the parallel poll disable bit set (bit 4) in the bit pattern disables the configuration, so you can use any value between hex 70 and 7E.

```
ibcmd(ud, "?#\x05\x70", 4);
```

---

## Multiline Interface Messages

This appendix contains a multiline interface message reference list, which describes the mnemonics and messages that correspond to the interface functions. These multiline interface messages are sent and received with ATN asserted.

For more information on these messages, refer to the ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*.

**Table A-1.** Multiline Interface Messages

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

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

**Table A-1.** Multiline Interface Messages (Continued)

Hex	Dec	ASCII	Msg
40	64	@	MTA0
41	65	A	MTA1
42	66	B	MTA2
43	67	C	MTA3
44	68	D	MTA4
45	69	E	MTA5
46	70	F	MTA6
47	71	G	MTA7
48	72	H	MTA8
49	73	I	MTA9
4A	74	J	MTA10
4B	75	K	MTA11
4C	76	L	MTA12
4D	77	M	MTA13
4E	78	N	MTA14
4F	79	O	MTA15
50	80	P	MTA16
51	81	Q	MTA17
52	82	R	MTA18
53	83	S	MTA19
54	84	T	MTA20
55	85	U	MTA21
56	86	V	MTA22
57	87	W	MTA23
58	88	X	MTA24
59	89	Y	MTA25
5A	90	Z	MTA26
5B	91	[	MTA27
5C	92	\	MTA28
5D	93	]	MTA29
5E	94	^	MTA30
5F	95	_	UNT

Hex	Dec	ASCII	Msg
60	96	`	MSA0, PPE
61	97	a	MSA1, PPE, CFG1
62	98	b	MSA2, PPE, CFG2
63	99	c	MSA3, PPE, CFG3
64	100	d	MSA4, PPE, CFG4
65	101	e	MSA5, PPE, CFG5
66	102	f	MSA6, PPE, CFG6
67	103	g	MSA7, PPE, CFG7
68	104	h	MSA8, PPE, CFG8
69	105	i	MSA9, PPE, CFG9
6A	106	j	MSA10, PPE, CFG10
6B	107	k	MSA11, PPE, CFG11
6C	108	l	MSA12, PPE, CFG12
6D	109	m	MSA13, PPE, CFG13
6E	110	n	MSA14, PPE, CFG14
6F	111	o	MSA15, PPE, CFG15
70	112	p	MSA16, PPD
71	113	q	MSA17, PPD
72	114	r	MSA18, PPD
73	115	s	MSA19, PPD
74	116	t	MSA20, PPD
75	117	u	MSA21, PPD
76	118	v	MSA22, PPD
77	119	w	MSA23, PPD
78	120	x	MSA24, PPD
79	121	y	MSA25, PPD
7A	122	z	MSA26, PPD
7B	123	{	MSA27, PPD
7C	124		MSA28, PPD
7D	125	}	MSA29, PPD
7E	126	~	MSA30, PPD
7F	127	DEL	

<b>Multiline Interface Message Definitions</b>			
CFE †	Configuration Enable	PPD	Parallel Poll Disable
CFG †	Configure	PPE	Parallel Poll Enable
DCL	Device Clear	PPU	Parallel Poll Unconfigure
GET	Group Execute Trigger	SDC	Selected Device Clear
GTL	Go To Local	SPD	Serial Poll Disable
LLO	Local Lockout	SPE	Serial Poll Enable
MLA	My Listen Address	TCT	Take Control
MSA	My Secondary Address	UNL	Unlisten
MTA	My Talk Address	UNT	Untalk
PPC	Parallel Poll Configure		
† This multiline interface message is a proposed extension to the IEEE 488.1 specification to support the HS488 high-speed protocol.			

# Status Word Conditions

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

For information about how to use `ibsta` in your application program, refer to Chapter 2, *Developing Your Application with the VxWorks Drivers*.

Table B-1 shows the status word layout.

**Table B-1.** Status Word Layout

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

## ERR

---

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

## TIMO

---

TIMO indicates that the timeout period has been exceeded. TIMO is set in the status word following an `ibwait` call if the TIMO bit of the mask parameter is set and the time limit expires. TIMO is also set following any I/O functions (for example, `ibcmd`, `ibrdr`, and `ibwrt`) if a timeout occurs during one of these calls. TIMO is cleared in all other circumstances.

## END

---

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

## SRQI

---

SRQI indicates that a GPIB device is requesting service. SRQI is set whenever the GPIB board is CIC and the GPIB SRQ line is asserted. SRQI is cleared either when the GPIB board ceases to be the CIC or when the GPIB SRQ line is unasserted.

## CMPL

---

CMPL indicates the condition of I/O operations. Because I/O calls in the NI-488 driver are all synchronous (meaning the call does not return until the operation is complete), CMPL is always set.



## LOK

---

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

## REM

---

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

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

## CIC

---

CIC indicates whether the GPIB board is the Controller-In-Charge. CIC is set either when you execute the `ibsic` function while the GPIB board is System Controller or when another Controller passes control to the GPIB board. CIC is cleared either when the GPIB board detects Interface Clear (IFC) from the System Controller or when the GPIB board passes control to another device.

## ATN

---

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

## TACS

---

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

## LACS

---

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

## DTAS

---

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

## DCAS

---

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

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



---

# Error Codes and Solutions

This appendix describes each error, including conditions under which it might occur and possible solutions.

Table C-1 lists the GPIB error codes.

**Table C-1.** GPIB Error Codes

<b>Error Mnemonic</b>	<b>iberr Value</b>	<b>Meaning</b>
EDVR	0	System error
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	No Listeners on the GPIB
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
ECAP	11	No capability for operation

## EDVR (0)

---

EDVR is returned when the board name passed to `ibfind` cannot be accessed. The global variable `ibcnt1` contains an error code. This error occurs when you try to access a board that is not installed or configured properly.

EDVR is also returned if an invalid unit descriptor is passed to any NI-488 function call.

### Solutions

Following are some possible solutions:

- Use only board names configured in the driver source code as parameters to the `ibfind` function.
- Use the unit descriptor returned from `ibfind` as the first parameter in subsequent NI-488 functions. Examine the variable before the failing function to make sure its value has not been corrupted.

## ECIC (1)

---

ECIC is returned when one of the following board functions or routines is called while the board is not CIC:

- Any board-level NI-488 functions that issue GPIB command bytes: `ibcmd`, `ibln`, and `ibrpp`
- `ibcac` and `ibgts`

### Solutions

Following are some possible solutions:

- Use `ibsic` to make the GPIB board become CIC on the GPIB.
- Use `ibrsc 1` to make sure your GPIB board is configured as System Controller.
- In multiple CIC situations, always be certain that the CIC bit appears in the status word `ibsta` before attempting these calls. If it does not appear, you can perform an `ibwait` (for CIC) call to delay further processing until control is passed to the board.

## ENOL (2)

---

ENOL usually occurs when you attempt a write operation without addressing Listeners. ENOL can also indicate that the GPIB address the application uses for a device does not match the GPIB address of any device connected to the bus, that the GPIB cable is not connected to the device, or that the device is not powered on.

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

### Solutions

Following are some possible solutions:

- Make sure that the GPIB address you are using matches the GPIB address of the device to which you want to write data.
- Use the appropriate hex code in `ibcmd` to address your device.
- Check your cable connections and make sure at least two-thirds of your devices are powered on.
- Reduce the write byte count to that which is expected by the Controller.

## EADR (3)

---

EADR occurs when the GPIB board is CIC and is not properly addressing itself before read and write functions.

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

### Solutions

Following are some possible solutions:

- Make sure that the GPIB board is addressed correctly using `ibcmd` before calling `ibrdr` or `ibwrt`.
- Avoid calling `ibgts` except immediately after an `ibcmd` call. (`ibcmd` causes ATN to be asserted.)

## EARG (4)

---

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

- `ibtmo` called with a value not in the range 0 through 17
- `ibeos` called with meaningless bits set in the high byte of the second parameter
- `ibpad` or `ibsad` called with invalid addresses
- `ibppc` called with invalid parallel poll configurations

### Solution

Make sure that the parameters passed to the NI-488 function are valid.

## ESAC (5)

---

ESAC results when `ibsic` or `ibsrc` is called when the GPIB board does not have System Controller capability.

### Solution

Give the GPIB board System Controller capability by calling `ibrsc 1`.

## EABO (6)

---

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

### Solutions

Following are some possible solutions:

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

## **ENEB (7)**

---

ENEB occurs when no GPIB board exists at the I/O address specified when the driver is installed. This problem happens when the board is not physically plugged into the system, the I/O address specified during configuration does not match the actual board setting, or there is a system conflict with the base I/O address.

### **Solution**

Make sure there is a GPIB board in your computer that is properly configured both in hardware and software using a valid base I/O address.

## **ECAP (11)**

---

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

### **Solution**

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

---

# Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

## Electronic Services

### Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422

Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

### FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as anonymous and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.



## Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

## E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

[support@natinst.com](mailto:support@natinst.com)

## Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

<b>Country</b>	<b>Telephone</b>	<b>Fax</b>
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

# Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

Fax ( \_\_\_\_ ) \_\_\_\_\_ Phone ( \_\_\_\_ ) \_\_\_\_\_

Computer brand \_\_\_\_\_ Model \_\_\_\_\_ Processor \_\_\_\_\_

Operating system (include version number) \_\_\_\_\_

Clock speed \_\_\_\_\_ MHz RAM \_\_\_\_\_ MB Display adapter \_\_\_\_\_

Mouse \_\_\_\_ yes \_\_\_\_ no Other adapters installed \_\_\_\_\_

Hard disk capacity \_\_\_\_\_ MB Brand \_\_\_\_\_

Instruments used \_\_\_\_\_

\_\_\_\_\_

National Instruments hardware product model \_\_\_\_\_ Revision \_\_\_\_\_

Configuration \_\_\_\_\_

National Instruments software product \_\_\_\_\_ Version \_\_\_\_\_

Configuration \_\_\_\_\_

The problem is: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

List any error messages: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

The following steps reproduce the problem: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

# Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

## National Instruments Products

GPIB hardware revision \_\_\_\_\_

GPIB hardware settings (slot number, and so on) \_\_\_\_\_

NI-488 version \_\_\_\_\_

## Other Products

Computer make and model \_\_\_\_\_

Microprocessor \_\_\_\_\_

Clock frequency or speed \_\_\_\_\_

Type of video board installed \_\_\_\_\_

Operating system version \_\_\_\_\_

Operating system mode \_\_\_\_\_

Programming language \_\_\_\_\_

Programming language version \_\_\_\_\_

Other boards in system \_\_\_\_\_

Base I/O address of other boards \_\_\_\_\_

DMA channels of other boards \_\_\_\_\_

Interrupt level of other boards \_\_\_\_\_

# Documentation Comment Form

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

**Title:** *GPIB Reference Manual for VXIpc Embedded Controllers and VxWorks*

**Edition Date:** *December 1997*

**Part Number:** *321858A-01*

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

---

---

---

---

---

---

---

---

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

---

---

---

---

---

---

---

---

Thank you for your help.

Name \_\_\_\_\_

Title \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

E-Mail Address \_\_\_\_\_

Phone ( \_\_\_\_ ) \_\_\_\_\_ Fax ( \_\_\_\_ ) \_\_\_\_\_

**Mail to:** Technical Publications  
National Instruments Corporation  
6504 Bridge Point Parkway  
Austin, Texas 78730-5039

**Fax to:** Technical Publications  
National Instruments Corporation  
512 794 5678

# Glossary

---

Prefix	Meanings	Value
n-	nano-	$10^{-9}$
$\mu$ -	micro-	$10^{-6}$
m-	milli-	$10^{-3}$
k-	kilo-	$10^3$
M-	mega-	$10^6$

## A

- acceptor handshake      Listeners use this GPIB interface function to receive data, and all devices use it to receive commands. *See* handshake.
- access board              The GPIB board that controls and communicates with the devices on the bus that are attached to it.
- ANSI                        American National Standards Institute.
- API                         Application programming interface.
- ASCII                      American Standard Code for Information Interchange.
- asynchronous            An action or event that occurs at an unpredictable time with respect to the execution of a program.

## B

- base I/O address        *See* I/O address.
- board-level function    A rudimentary function that performs a single operation.

## C

- caller                     A place in the program from which a call is made; the calling function.

CFE	Configuration Enable. The GPIB command which precedes CFGn and is used to place devices into their configuration mode.
CFGn	These GPIB commands (CFG1 through CFG15) follow CFE and are used to configure all devices for the number of meters of cable in the system so that HS488 transfers occur without errors.
CIC	Controller-In-Charge. The device that manages the GPIB by sending interface messages to other devices.
CPU	Central processing unit.
<b>D</b>	
DAV	Data Valid. One of the three GPIB handshake lines. <i>See</i> handshake.
DCL	Device Clear. The GPIB command used to reset the device or internal functions of all devices. <i>See</i> SDC.
DIO1 through DIO8	The GPIB lines that are used to transmit command or data bytes from one device to another.
DMA	Direct memory access. High-speed data transfer between the GPIB board and memory that is not handled directly by the CPU. Not available on some systems.
driver	Device driver software installed within the operating system.
<b>E</b>	
END or END Message	A message that signals the end of a data string. END is sent by asserting the GPIB End or Identify (EOI) line with the last data byte.
EOI	A GPIB line that is used to signal either the last byte of a data message (END) or the parallel poll Identify (IDY) message.
EOS or EOS Byte	A 7- or 8-bit end-of-string character that is sent as the last byte of a data message.
EOT	End of transmission.
ESB	The Event Status bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.

**F**

Function Return            Describes the return value of the function.

**G**

GET                            Group Execute Trigger. It is the GPIB command used to trigger a device or internal function of an addressed Listener.

GPIB                            General Purpose Interface Bus is the common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*, and ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*.

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

GPIB board                    Refers to the National Instruments family of GPIB interface boards.

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

**H**

handshake                    The mechanism used to transfer bytes from the Source Handshake function of one device to the Acceptor Handshake function of another device. The three GPIB lines DAV, NRFD, and NDAC are used in an interlocked fashion to signal the phases of the transfer, so that bytes can be sent asynchronously (for example, without a clock) at the speed of the slowest device.

For more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987.

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

## I

<code>ibcnt</code>	After each NI-488 I/O function, this global variable contains the actual number of bytes transmitted.
<code>iberr</code>	A global variable that contains the specific error code associated with a function call that failed.
<code>ibsta</code>	At the end of each function call, this global variable (status word) contains status information.
IEEE	Institute of Electrical and Electronic Engineers.
interface message	A broadcast message sent from the Controller to all devices and used to manage the GPIB. Interface messages are also referred to as GPIB commands.
I/O	Input/Output. In the context of this manual, the transmission of commands or messages between the computer via the GPIB board and other devices on the GPIB.
I/O address	The address of the GPIB board from the point of view of the CPU, as opposed to the GPIB address of the GPIB board. Also called port address or board address.
<code>ist</code>	An Individual Status bit of the status byte used in the Parallel Poll Configure function.

## K

kernel	The set of programs in an operating system that implements basic system functions.
kernel-level implementation	The linking or installation of the NI-488 driver into the operating system kernel so that the driver functions as a general system resource available to all application programs.

## L

language interface	Code that enables an application program that uses NI-488 functions to access the driver.
--------------------	---



Listener	A GPIB device that receives data messages from a Talker.
LLO	Local Lockout. The GPIB command used to tell all devices that they may or should ignore remote (GPIB) data messages or local (front panel) controls, depending on whether the device is in local or remote program mode.
<b>M</b>	
m	Meters.
MAV	The Message Available bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.
MLA	My Listen Address. A GPIB command used to address a device to be a Listener. It can be any one of the 31 primary addresses.
MSA	My Secondary Address. The GPIB command used to address a device to be a Listener or a Talker when extended (two byte) addressing is used. The complete address is a MLA or MTA address followed by an MSA address. There are 31 secondary addresses for a total of 961 distinct listen or talk addresses for devices.
MTA	My Talk Address. A GPIB command used to address a device to be a Talker. It can be any one of the 31 primary addresses.
<b>N</b>	
NDAC	Not Data Accepted. One of the three GPIB handshake lines. <i>See</i> handshake.
NRFD	Not Ready For Data. One of the three GPIB handshake lines. <i>See</i> handshake.
<b>O</b>	
OS	Operating system.
<b>P</b>	
parallel poll	The process of polling all configured devices at once and reading a composite poll response. <i>See</i> serial poll.

PPC	Parallel Poll Configure. It is the GPIB command used to configure an addressed Listener to participate in polls.
PPD	Parallel Poll Disable. It is the GPIB command used to disable a configured device from participating in polls. There are 16 PPD commands.
PPE	Parallel Poll Enable. It is the GPIB command used to enable a configured device to participate in polls and to assign a DIO response line. There are 16 PPE commands.
PPU	Parallel Poll Unconfigure. It is the GPIB command used to disable any device from participating in polls.

## S

s	Seconds.
SDC	Selected Device Clear. The GPIB command used to reset internal or device functions of an addressed Listener. <i>See</i> DCL.
serial poll	The process of polling and reading the status byte of one device at a time. <i>See</i> parallel poll.
service request	<i>See</i> SRQ.
SPD	Serial Poll Disable. The GPIB command used to cancel an SPE command.
SPE	Serial Poll Enable. The GPIB command used to enable a specific device to be polled. That device must also be addressed to talk. <i>See</i> SPD.
SRQ	Service Request. The GPIB line that a device asserts to notify the CIC that the device needs servicing.
status byte	The IEEE 488.2-defined data byte sent by a device when it is serially polled.
status word	<i>See</i> <code>ibsta</code> .
synchronous	Refers to the relationship between the NI-488 driver functions and a process when executing driver functions is predictable; the process is blocked until the driver completes the function.

**System Controller** The single designated Controller that can assert control (become CIC of the GPIB) by sending the Interface Clear (IFC) message. Other devices can become CIC only by having control passed to them.

## T

**Talker** A GPIB device that sends data messages to Listeners.

**TCT** Take Control. The GPIB command used to pass control of the bus from the current Controller to an addressed Talker.

**timeout** A feature of the NI-488 driver that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.

## U

**ud** Unit descriptor. A variable name and first argument of each function call that contains the unit descriptor of the GPIB interface board or other GPIB device that is the object of the function.

**UNL** Unlisten. The GPIB command used to unaddress any active Listeners.

**UNT** Untalk. The GPIB command used to unaddress an active Talker.

**user-level implementation** The static or dynamic linking of the NI-488 driver directly to a user application program. This implementation method is not available on some operating systems, for which a kernel-level implementation is the only option.

# Index

---

## A

- active Controller, 1-1
- address functions
  - IBPAD, 3-17
  - IBSAD, 3-27
- addresses. *See* GPIB addresses.
- ANSI/IEEE Standard 488.1-1987, 1-1
- ANSI/IEEE Standard 488.2-1992, 1-1
- application development. *See also* GPIB programming techniques.
  - compiling and linking, 2-5 to 2-6
  - debugging, 2-6 to 2-8
    - communication errors, 2-7
    - configuration errors, 2-6
    - timing errors, 2-7
    - using global status variables, 2-6
  - global variables for checking status, 2-3 to 2-5
    - count variables - `ibcnt` and `ibcntl`, 2-5
    - error variable - `iberr`, 2-4
    - status word - `ibsta`, 2-3 to 2-4
  - GPIB library files provided with VxWorks, 2-2 to 2-3
  - items to include for C applications, 2-2 to 2-3
  - NI-488 functions, 2-2
  - VxWorks NI-488 and ESP-488 drivers, 2-1 to 2-2
- ATN (attention) line (table), 1-3
- ATN status word condition, B-3

## B

- bulletin board support, D-1

## C

- CIC. *See* Controller-in-Charge (CIC).
- CIC status word condition, B-3
- clear functions
  - IBIST, 3-11
  - IBSIC, 3-24
  - IBSRE, 3-25
- CMPL status word condition, B-2
- command function (`IBCMD`), 3-5
- communication errors
  - repeat addressing, 2-7
  - termination method, 2-7
- compiling and linking applications, 2-5 to 2-6
- configuration, 1-4 to 1-5
  - controlling more than one board, 1-4 to 1-5
  - linear and star system configuration (illustration), 1-4
  - requirements, 1-5
- configuration errors, debugging, 2-6
- control line status. *See* `IBLINES` function.
- controller functions
  - IBCAC, 3-4
  - IBGTS, 3-10
  - IBRSC, 3-21
- Controller-in-Charge (CIC)
  - active Controller as CIC, 1-1
  - System Controller as, 1-1
- Controllers
  - definition, 1-1
  - idle Controller, 1-1
  - System Controller, 1-1
- count variables (`ibcnt` and `ibcntl`), 2-5
- customer communication, *xi*, D-1 to D-2

**D**

- data lines, 1-2
- data transfers
  - terminating
    - GPIB programming technique, 4-1 to 4-2
- DAV (data valid) line (table), 1-3
- DCAS status word condition, B-4
- debugging
  - communication errors, 2-7
    - repeat addressing, 2-7
    - termination method, 2-7
  - configuration errors, 2-6
  - global status variables, 2-6
  - timing errors, 2-7
- developing applications. *See* application development.
- documentation
  - conventions used in manual, *x*
  - organization of manual, *ix*
  - related documentation, *x*
- drivers
  - ESP-488 driver, 2-1 to 2-2
  - NI-488 driver, 2-1 to 2-2
- DTAS status word condition, B-4

**E**

- EABO error code, C-4
- EADR error code, C-3
- EARG error code, C-4
- ECAP error code, C-5
- ECIC error code, C-2
- EDVR error code, C-2
- electronic support services, D-1 to D-2
- e-mail support, D-2
- END status word condition, B-2
- end-of-string. *See* EOS.
- ENEB error code, C-5

- ENOL error code, C-3
- EOI (end or identify) line
  - enabling or disabling. *See* IBEOT function.
  - purpose (table), 1-3
- EOS comparison method, 4-1
- EOS configurations (table), 3-6
- EOS read method, 4-1
- EOS write method, 4-1
- ERR status word condition, B-2
- error codes and solutions, C-1 to C-5
  - EABO, C-4
  - EADR, C-3
  - EARG, C-4
  - ECAP, C-5
  - ECIC, C-2
  - EDVR, C-2
  - ENEB, C-5
  - ENOL, C-3
  - ESAC, C-4
- error variable - *iberr*, 2-4
- ESAC error code, C-4
- ESP-488 driver, 2-1 to 2-2

**F**

- fax and telephone support numbers, D-2
- Fax-on-Demand support, D-2
- FTP support, D-1
- functions. *See* NI-488 functions.

**G**

- General Purpose Interface Bus. *See* GPIB.
- global variables, 2-3 to 2-5
  - count variables - *ibcnt* and *ibcntl*, 2-5
  - debugging applications, 2-6
  - error variable - *iberr*, 2-4
  - status word - *ibsta*, 2-3 to 2-4, B-1 to B-4

**GPIB**

- configuration, 1-4 to 1-5
  - controlling more than one board, 1-4 to 1-5
  - linear and star system configuration (illustration), 1-4
  - requirements, 1-5
- definition, 1-1
- overview, 1-1
- sending messages across, 1-2 to 1-3
  - data lines, 1-2
  - handshake lines, 1-3
  - interface management lines, 1-3
- Talkers, Listeners, and Controllers, 1-1

**GPIB addresses**

- address bits (figure), 1-2
- listen address, 1-2
- primary, 1-2
- purpose, 1-2
- secondary, 1-2
- talk address, 1-2

**GPIB programming techniques. *See also* application development.**

- parallel polling, 4-4 to 4-5
- serial polling, 4-2 to 4-4
  - service requests
    - from IEEE 488 devices, 4-3
    - from IEEE 488.2 devices, 4-3
  - SRQ and serial polling
    - with NI-488 device functions, 4-3 to 4-4
    - with NI-488.2 routines, 4-4
- Talker/Listener applications, 4-2
- termination of data transfers, 4-1 to 4-2
- waiting for GPIB conditions, 4-2

**H**

- handshake lines, 1-3

**I**

- IBCAC function, 3-4
- IBCMD function, 3-5
- ibcnt and ibcntl variables, 2-5
- IBEOS function
  - description, 3-6 to 3-7
  - EOS configurations (table), 3-6
  - examples, 3-6
  - terminating data transfers, 4-1
- IBEOT function
  - description, 3-8
  - terminating data transfer, 4-1
- iberr (error variable), 2-4 to 2-5
- IBFIND function, 3-9
- IBGTS function, 3-10
- IBIST function, 3-11
- IBLINES function, 3-12 to 3-13
- IBLN function, 3-14
- IBLOC function, 3-15
- IBONL function, 3-16
- IBPAD function, 3-17
- IBPPC function, 3-18
- IBRD function, 3-19
- IBRPP function, 3-20
- IBRSC function, 3-21
- IBRSV function, 3-22
- IBSAD function, 3-27
- IBSIC function, 3-24
- IBSRE function, 3-25
- ibsta. *See* status word - ibsta.
- IBTMO function, 3-26 to 3-27
- IBWAIT function
  - description, 3-28 to 3-29
  - programming techniques, 4-2
- IBWRT function, 3-30
- IEEE Standard 488.1-1987. *See* GPIB.
- IFC (interface clear) line (table), 1-3
- interface clear function (IBSIC), 3-24
- interface management lines (table), 1-3

**L**

LACS status word condition, B-4  
 linking and compiling applications, 2-5 to 2-6  
 listen address, setting, 1-2  
 Listeners  
   definition, 1-1  
   finding using IBLN function, 3-14  
   Talker/Listener applications, 4-2  
 local function (IBLOC), 3-15  
 LOK status word condition, B-3

**M**

manual. *See* documentation.  
 messages, sending across GPIB, 1-2 to 1-3  
   data lines, 1-2  
   handshake lines, 1-3  
   interface management lines, 1-3  
 multiline interface messages, A-1 to A-4

**N**

NDAC (not data accepted) line (table), 1-3  
 NI-488 driver  
   developing applications with, 2-1 to 2-2  
   GPIB library files provided with  
     VxWorks, 2-1 to 2-2  
 NI-488 functions  
   alphabetical list of functions (table),  
     3-2 to 3-3  
   developing applications, 2-2  
   IBCAC, 3-4  
   IBCMD, 3-5  
   IBEOS, 3-6 to 3-7  
   IBEOT, 3-8  
   IBFIND, 3-9  
   IBGTS, 3-10  
   IBIST, 3-11  
   IBLINES, 3-12 to 3-13  
   IBLN, 3-14  
   IBLOC, 3-15

IBONL, 3-16  
 IBPAD, 3-17  
 IBPPC, 3-18  
 IBRD, 3-19  
 IBRPP, 3-20  
 IBRSC, 3-21  
 IBRSV, 3-22  
 IBSAD, 3-27  
 IBSIC, 3-24  
 IBSRE, 3-25  
 IBTMO, 3-26 to 3-27  
 IBWAIT, 3-28 to 3-29  
 IBWRT, 3-30

NRFD (not ready for data) line (table), 1-3

**O**

online/offline function. *See* IBONL function.

**P**

parallel polling, 4-4 to 4-5  
 parallel polling functions  
   IBIST, 3-11  
   IBPPC, 3-18  
   IBRPP, 3-20  
 primary GPIB address  
   changing with IBPAD function, 3-17  
   definition, 1-2  
 programming. *See* application development;  
   GPIB programming techniques.

**R**

read function (IBRD), 3-19  
 REM status word condition, B-3  
 remote function (IBSRE), 3-25  
 REN (remote enable) line (table), 1-3  
 repeat addressing, 2-7

**S**

secondary GPIB address, 1-2  
 serial polling, 4-2 to 4-4  
   service requests  
     from IEEE 488 devices, 4-3  
     from IEEE 488.2 devices, 4-3  
   SRQ and serial polling  
     with NI-488 device functions,  
       4-3 to 4-4  
     with NI-488.2 routines, 4-4  
   using IBRSV function, 3-22  
 setting up your system. *See* configuration.  
 SRQ (service request) line  
   purpose (table), 1-3  
   serial polling, 4-3  
 SRQI status word condition, B-2  
 status word - *ibsta*  
   ATN, B-3  
   CIC, B-3  
   CMPL, B-2  
   DCAS, B-4  
   DTAS, B-4  
   END, B-2  
   ERR, B-2  
   LACS, B-4  
   LOK, B-3  
   REM, B-3  
   SRQI, B-2

status word layout (table), 2-3 to 2-4, B-1

TACS, B-4

TIMO, B-2

using in applications, 2-3 to 2-4

  debugging, 2-6

System Controller, 1-1

**T**

TACS status word condition, B-4

talk address, setting, 1-2

Talkers

  definition, 1-1

  Talker/Listener applications, 4-2

technical support, D-1 to D-2

telephone and fax support numbers, D-2

termination of data transfers

  methods for, 4-1 to 4-2

  problems in applications, 2-7

timeout code values (table), 3-26 to 3-27

timing errors, debugging, 2-6

TIMO status word condition, B-2

**W**

wait function (IBWAIT), 3-28 to 3-29

wait mask layout (table), 3-28 to 3-29

write function (IBWRT), 3-30