ADL-GPIB

IEEE 488 Commands

Function Reference Manual

Manual Rev. 2.01

Revision Date: December 21, 2006

Part No: 50-15043-1000

Recycled Paper

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Advance Technologies; Automate the World.

Copyright 2005 ADLINK TECHNOLOGY INC.

All Rights Reserved.

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of the manufacturer.

Trademarks

Product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Getting Service from ADLINK

Customer Satisfaction is top priority for ADLINK Technology Inc. Please contact us should you require any service or assistance.

ADLINK TECHNOLOGY INC.

Web Site:	http://www.adlinktech.com
Sales & Service:	Service@adlinktech.com
TEL:	+886-2-82265877
FAX:	+886-2-82265717
Address:	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan

Please email or FAX this completed service form for prompt and satisfactory service.

Company Information		
Company/Organization		
Contact Person		
E-mail Address		
Address		
Country		
TEL	FAX:	
Web Site		
Product Information		
Product Model		
Environment	OS: M/B: Chipset:	 CPU: Bios:

Please give a detailed description of the problem(s):

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Table of Contents

1 Using ADL-GPIB Functions 1

1.1 The Fundamentals of Building Windows 2000/NT/98 Application with ADL-GPIB 1

1.2 ADL-GPIB Functions Overview 4

1.3 Data Types 10

2 IEEE 488 Function Reference 11

2.1 ibask 11

2.2 ibbna 15

2.3 ibcac 15

2.4 ibclr 17

2.5 ibcmd 18

2.6 ibcmda 19

2.7 ibconfig 21

2.8 ibdev 28

2.9 ibdma 30

2.10 ibeot 31

2.11 ibeos 32

2.12 ibfind 34

2.13 ibgts 35

2.14 ibist 36

2.15 iblines 37

2.16 ibln 39

2.17 ibloc 41

2.18 ibonl 42

2.19 ibnotify 43

2.20 ibpad 46

2.21 ibsad 47

2.22 ibpct 48

2.23 ibppc 49

2.24 ibrd 51

2.25 ibrda 53

2.26 ibrdf 55

2.27 ibrpp 57

2.28 ibrsc 58

2.29 ibrsp 59

2.30 ibrsv 60

2.31 ibsic 61

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.32 ibsre 62
2.33 ibstop 63
2.34 ibtmo 64
2.35 ibtrg 66
2.36 ibwait 67
2.37 ibwrt 69
2.38 ibwrta 71
2.39 ibwrtf 73

3 Multi-Device IEEE 488 Function Reference 75
3.1 AllSpoll 75
3.2 DevClear 76
3.3 DevClearList 77
3.4 EnableLocal 78
3.5 EnableRemote 79
3.6 FindLstn 80
3.7 FindRQS 81
3.8 PassControl 82
3.9 PPoll 83
3.10 PPollConfig 84
3.11 PPollUnConfig 85
3.12 RcvRespMsg 86
3.13 ReadStatusByte 87
3.14 Receive 88
3.15 ReceiveSetup 89
3.16 ResetSys 90
3.17 Send 91
3.18 SendCmds 92
3.19 SendDataBytes 93
3.20 SendList 94
3.21 SendIFC 95
3.22 SendLLO 96
3.23 SendSetup 97
3.24 SetRWLS 98
3.25 TestSRQ 99
3.26 TestSys 100
3.27 Trigger 101
3.28 TriggerList 102
3.29 WaitSRQ 103
Appendix A: Status Codes 105

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Appendix B: Error Codes 106

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

List of Tables

Table 1-1: IEEE 488 Device Level Function Group 4
Table 1-2: IEEE 488 Board Level Function Group 6
Table 1-3: IEEE 488.2 Function Group 8
Table 1-4: Data Types 10
Table 2-1: ibask Board Configuration Parameter Options 12
Table 2-2: ibask Device Configuration Parameter Options 14
Table 2-3: Board Configuration Parameter Options 22
Table 2-4: Board Configuration Parameter Options 24
Table 2-5: Device Configuration Parameter Options 26
Table 4-1: Status Codes 105
Table 4-2: Error Codes 106

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

List of Figures

Figure 1-1: Open Project dialog box 2

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1 Using ADL-GPIB Functions

ADL-GPIB is a software driver for ADLINK GPIB interface.

1.1 The Fundamentals of Building Windows 2000/ NT/98 Application with ADL-GPIB

Creating a Windows 2000/NT/98 ADL-GPIB Application Using Microsoft Visual C/C++

To create a data acquisition application using ADL-GPIB and Microsoft Visual C/C++, follow these steps after entering Visual C/C++:

1. Open the project in which you want to use ADL-GPIB. This can be a new or existing project
2. Include header file ADGPIB.H in the C/C++ source files that call ADL-GPIB functions. ADGPIB.H contains all the function declarations and constants that you can use to develop your data acquisition application. Incorporate the following statement in your code to include the header file.

```
#include  "ADGPIB.H"
```

3. Build your application.

Setting the appropriate compile and link options, then build your application by selecting the Build command from Build menu (Visual C/C++ 4.0). Remember to link ADL-GPIB's import library GPIB-32.LIB.

Creating a Windows 2000/NT/98 ADL-GPIB Application Using Microsoft Visual Basic

To create a data acquisition application using ADL-GPIB and Visual Basic, follow these steps after entering Visual Basic:

1. Open the project in which you want to use ADL-GPIB. This can be a new or existing project

Open a new project by selecting the New Project command from the File menu. If it is an existing project, open it by select-

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

ing the Open Project command from the File menu. Then the Open Project dialog box appears.

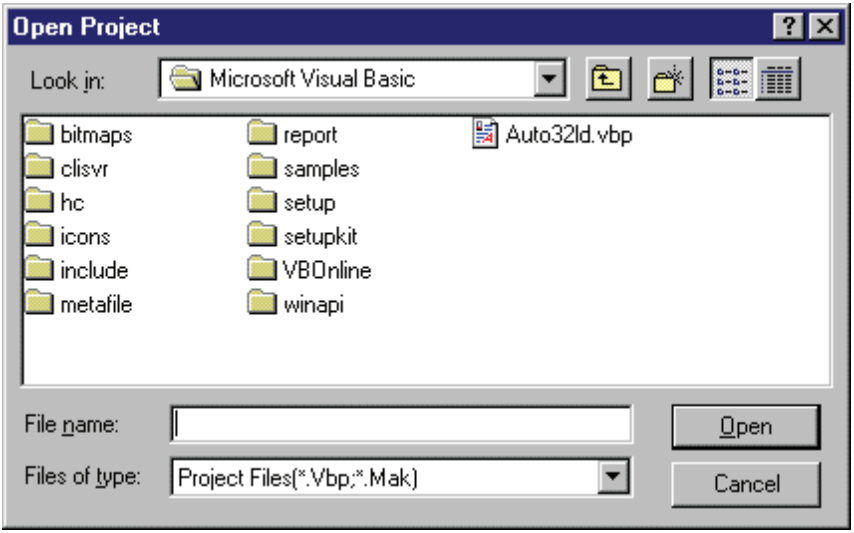


Figure 1-1: Open Project dialog box

Chang directory to the place the project file located. Double-click the project file name in the File Name list to load the project.

2. Add file ADGPIB.BAS into the project if this file is not included in the project. This file contains all the procedure declarations and constants that you can use to develop your data acquisition application.
3. Design the interface for the application.

To design the interface, you place the desired elements, such as command button, list box, text box, etc., on the Visual Basic form. These are standard controls from the Visual Basic Toolbox. To place a control on a form, you just move pointer to Toolbox, select the desired control and draw it on the form. Or you can double-click the control icon in the Toolbox to place it on the form.

4. Set properties for the controls.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

To view the property list, click the desired control and then choose the Properties command from the View menu or press F4, or you can also click the Properties button  on the toolbar.

5. Write the event code.

The event code defines the action you want to perform when an event occurs. To write the event code, double-click the desired control or form to view the code module and then add code you want. You can call the functions that declared in the file ADGPIB.BAS to perform data acquisition operations.

6. Run your application.

To run the application, choose Start from the Run menu, or click the Start icon  on the toolbar (you can also press F5).

7. Distribute your application.

Once you have finished a project, you can save the application as an executable (.EXE) file by using the Make EXE File command on the File menu. And once you have saved your application as an executable file, you've ready to distribute it. When you distribute your application, remember also to include the ADL-GPIB's DLL and driver files.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.2 ADL-GPIB Functions Overview

ADL-GPIB functions are grouped to the following classes:

IEEE 488 Device Level Function Group

Function	Description
ibask	Return the current setting value of the selected configuration item
ibbna	Set the access board of a device
ibclr	Clear a specific device
ibconfig	Set the software configuration parameters
ibdev	Open and initialize a device
ibeos	Configure the end-of-string (EOS) termination mode or character
ibeot	Enable or disable the automatic assertion of the GPIB EOI line atthe end of write I/O operations
ibln	Check for the presence of a device on the bus
ibloc	Go to local
ibonl	Place the device online or offline
ibpad	Set the primary address
ibpct	Pass control to another GPIB device with Controller capability
ibppc	Configure Parallel polling
ibrd	Read data from a device into a buffer
ibrda	Read data asynchronously from a device into a buffer
ibrdf	Read data from a device into a file
ibrdi	Read data from a device into a buffer
ibrdia	Read data asynchronously from a device into a buffer
ibrpp	Perform a parallel poll
ibrsp	Perform a serial poll
ibsad	Set or disable the secondary address
ibstop	Abort asynchronous I/O operation
ibtmo	Set or disable the I/O timeout period
ibtrg	Trigger selected device
ibwait	Wait for GPIB events

Table 1-1: IEEE 488 Device Level Function Group

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Function	Description
ibwrt	Write data to a device from a buffer
ibwrta	Write data asynchronously to a device from a buffer
ibwrtf	Write data to a device from a file

Table 1-1: IEEE 488 Device Level Function Group

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

IEEE 488 Board Level Function Group

Function	Purpose
ibask	Return the current setting value of the selected configuration item
ibcac	Become Active Controller state
ibcmd	Send GPIB commands
ibcmda	Send GPIB commands asynchronously
ibconfig	Set the software configuration parameters
ibdma	Enable or disable DMA
ibeos	Configure the end-of-string (EOS) termination mode or character
ibeot	Enable or disable the automatic assertion of the GPIB EOI line atthe end of write I/O operations
ibfind	Open and initialize a GPIB board
ibgts	Go from Active Controller state to Standby Controller state
ibist	Set or clear the board individual status bit for parallel polls
iblines	Return the status of the GPIB control lines
ibln	Check for the presence of a device on the bus
ibloc	Go to local
ibonl	Place the interface board online or offline
ibpad	Set the primary address
ibppc	Configure Parallel polling
ibrd	Read data from a device into a buffer
ibrda	Read data asynchronously from a device into a buffer
ibrdf	Read data from a device into a file
ibrdi	Read data from a device into a buffer
ibrdia	Read data asynchronously from a device into a buffer
ibrpp	Perform a parallel poll
ibrsc	Request or release system control
ibrsv	Request service and change the serial poll status byte
ibsad	Set or disable the secondary address
ibsic	Assert interface clear
ibsre	Set or clear the Remote Enable (REN) line
ibstop	Abort asynchronous I/O operation

Table 1-2: IEEE 488 Board Level Function Group

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Function	Purpose
ibtmo	Set or disable the I/O timeout period
ibwait	Wait for GPIB events
ibwrt	Write data to a device from a user buffer
ibwrta	Write data asynchronously to a device from a user buffer
ibwrtf	Write data to a device from a file

Table 1-2: IEEE 488 Board Level Function Group

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

IEEE 488.2 Function Group

Routine	Purpose
AllSpoll	Serial polling all devices
DevClear	Clear a single device
DevClearList	Clear multiple devices
EnableLocal	Enable operations from the front panel of devices (leave remote programming mode)
EnableRemote	Enable remote GPIB programming for devices
FindLstn	Find listening devices on the GPIB
FindRQS	Determine which device is requesting service
PassControl	Pass control to another device with Controller capability
PPoll	Perform a parallel poll on the GPIB
PPollConfig	Configure a device for parallel polls
PPollUnconfig	Unconfigure devices for parallel polls
RcvRespMsg	Read data bytes from a device addressed to talk
ReadStatusByte	Conduct serial polling single device
Receive	Read data bytes from a device
ReceiveSetup	Address a device to be a Talker and the interface board to bea Listener
ResetSys	Reset and initialize devices
Send	Send data bytes to a device
SendCmds	Send GPIB command bytes
SendDataBytes	Send data bytes to devices addressed to listen
SendIFC	Reset the GPIB by sending interface clear
SendList	Send data bytes to multiple GPIB devices
SendLLO	Send the Local Lockout (LLO) message to all devices
SendSetup	Set up devices to receive data in preparation for Send-DataBytes
SetRWLS	Place devices in remote with lockout state
TestSRQ	Determine the current state of the GPIB Service Request (SRQ) line
TestSys	Trigger a devices to conduct self-tests
Trigger	Trigger a device

Table 1-3: IEEE 488.2 Function Group

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Routine	Purpose
TriggerList	Trigger multiple devices
WaitSRQ	Wait until a device asserts the GPIB Service Request (SRQ) line

Table 1-3: IEEE 488.2 Function Group

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.3 Data Types

We defined some data types in ADGPIB.H. These data types are used by ADL-GPIB library. We suggest you to use these data types in your application programs. The following table shows the data type names, their ranges and the corresponding data types in C/C++, Visual Basic and Delphi (We didn't define these data types in ADGPIB.BAS and ADGPIB.PAS. Here they are just listed for reference)

Type Name	Description	Range	Type		
			C/C++(for 32-bit compiler)	Visual Basic	Pascal (Delphi)
U8	8-bit ASCII character	0 to 255	unsigned char	Byte	Byte
I16	16-bit signed integer	-32768 to 32767	short	Integer	SmallInt
U16 Addr4882_t	16-bit unsigned integer	0 to 65535	unsigned short	Not supported by BASIC, use the signed integer (I16) instead	Word
I32 ssize_t	32-bit signed integer	-2147483648 to 2147483647	long	Long	LongInt
U32 size_t	32-bit unsigned integer	0 to 4294967295	unsigned long	Not supported by BASIC, use the signed long integer (I32) instead	Cardinal
F32	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38	float	Single	Single
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309	double	Double	Double

Table 1-4: Data Types

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2 IEEE 488 Function Reference

2.1 ibask

@ Description

Return the current setting value of the selected configuration item.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: board or device unit descriptor

option: the configuration item whose value is being returned. The valid option items are listed in the table 2.1 and 2.2.

value: returns current value of the specified configuration item.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, ECAP, EDVR

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Options (Constants)	Options (Values)	Returned Information
ibaPAD	0x0001	The current primary address of the board
ibaSAD	0x0002	The current secondary address of the board.
ibaTMO	0x0003	The current I/O timeout of the board.
ibaEOT	0x0004	0: The GPIB EOI line is not asserted at the end of a write operation.
		1: EOI is asserted at the end of a write.
ibaPPC	0x0005	The current parallel poll configuration settings the board
ibaAUTOPOLL	0x0007	0: Automatic serial polling is disabled.
		1: Automatic serial polling is enabled.
ibaCICPROT	0x0008	0: The CIC protocol is disabled.
		1: The CIC protocol is enabled.
ibaIRQ	0x0009	0: Interrupts are not enabled.
		1: Interrupts are enabled.
ibaSC	0x000A	0: The board is not the GPIB SystemController.
		1: The board is the System Controller..
ibaSRE	0x000B	0: The board does not automatically assert the GPIB REN line when it becomes the System Controller.
		1: The board automatically asserts REN when it becomes the System Controller.
ibaEOSrd	0x000C	0: The EOS character is ignored during read operations.
		1: Read operation is terminated by the EOS character.
ibaEOSwrt	0x000D	0: The EOI line is not asserted when the EOS character is sent during a write operation.
		1: The EOI line is asserted when the EOS character is sent during a write operation.
ibaEOScmp	0x000E	0: A 7-bit compare is used for all EOS comparisons.
		1: An 8-bit compare is used for all EOS comparisons.
ibaEOSchar	0x000F	The current EOS character of the board.

Table 2-1: ibask Board Configuration Parameter Options

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Options (Constants)	Options (Values)	Returned Information
ibaPP2	0x0010	0: The board is in PP1 mode–remote parallel poll configuration.
		1: The board is in PP2 mode–local parallel poll configuration.
ibaTIMING	0x0011	The current bus timing of the board.
		1: Normal timing (T1 delay of 2 µs.)
		2: High speed timing (T1 delay of 500 ns.)
		3: Very high speed timing (T1 delay of 350 ns.)
ibaDMA	0x0012	0: The board will not use DMA for GPIB transfers.
		1: The board will use DMA for GPIB transfers
ibaSpollBit	0x0016	0: The SPOLL bit of ibsta is disabled.
		1: The SPOLL bit of ibsta is enabled.
ibaSendLLO	0x0017	0: The GPIB LLO command is not sent when a device is put online-ibfind or ibdev.
		1: The LLO command is sent.
ibaPPollTime	0x0019	0: The board uses the standard duration (2 µs) when conducting a parallel poll.
		1 to 17 = The board uses a variable length duration when conducting a parallel poll. The duration values correspond to the ibtmo timing values.
ibaEndBitIs-Normal	al0x001A	0: The END bit of ibsta is set only when EOI or EOI plus the EOS character is received. If the EOS character is received without EOI, the END bit is not set.
		1: The END bit is set whenever EOI, EOS, or EOI plus EOS is received.
ibaist	0x0020	The individual status (ist) bit of the board.
ibaRsv	0x0021	The current serial poll status byte of the board.

Table 2-1: ibask Board Configuration Parameter Options

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Options (Constants)	Options (Values)	Returned Information
ibaPAD	0x0001	The current primary address of the device.
ibaSAD	0x0002	The current secondary address of the device.
ibaTMO	0x0003	The current I/O timeout of the device.
ibaEOT	0x0004	0: The GPIB EOI line is not asserted at the end of a write operation.
		1: EOI is asserted at the end of a write.
ibaREADDR	0x0006	0: No unnecessary addressing is performed between device-level read and write operations.
		1: Addressing is always performed before a device-level read or write operation.
ibaEOSrd	0x000C	0: The EOS character is ignored during read operations.
		1: Read operation is terminated by the EOS character.
ibaEOSwrt	0x000D	0: The EOI line is not asserted when the EOS character is sent during a write operation.
		1: The EOI line is asserted when the EOS character is sent during a write operation.
ibaEOScmp	0x000E	0: A 7-bit compare is used for all EOS comparisons.
		1: An 8-bit compare is used for all EOS comparisons.
ibaEOSchar	0x000F	The current EOS character of the board.
ibaSPollTime	0x0018	The length of time the driver waits for a serial poll response when polling the device. The length of time is represented by the ibtmo timing values.
ibaEndBitIs-Normal	al0x001A	0: The END bit of ibsta is set only when EOI or EOI plus the EOS character is received. If the EOS character is received without EOI, the END bit is not set.
		1: The END bit is set whenever EOI, EOS, or EOI plus EOS is received.
ibaBNA	0x0200	The index of the GPIB access board used by the given device descriptor.

Table 2-2: ibask Device Configuration Parameter Options

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.2 ibbna

@ Description

Assign the access board of the specified device.

@ Support Level

Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: device unit descriptor

board_name: the name of the access board, e.g. gpib0.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, ECAP, EDVR, EOIP, ENEB
```

2.3 ibcac

@ Description

Set the specified gpib board to be active controller by asserting ATN. Before you call ibcac, the GPIB board must already be CIC. To make the board CIC, use the ibsic function.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The board can take control synchronously or asynchronously. 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.

@ Support Level

board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: board unit descriptor

v: takes control asynchronously or synchronously

0: asynchronously

1: synchronously

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, ECIC, EDVR, EOIP, ENEB

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.4 ibclr

@ Description

Send the GPIB Selected Device Clear (SDC) message to the specified device.

@ Support Level

Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: device unit descriptor

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.5 ibcmd

@ Description

Send GPIB commands. Command bytes are used to configure the state of the GPIB. They are not used to send instructions to GPIB devices. Use ibwrt to send device-specific instructions. The number of command bytes transferred is returned in the global variable ibcntl.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

```
ilcmd(ByVal ud As Integer, ByVal buf As String,
     ByVal cnt As Long) As Integer
or
call ibcmd(ByVal ud As Integer, ByVal buf As
     String)
```

@ Parameter

ud: device unit descriptor

buf: the buffer contains command string to sent

cnt: number of command bytes to sent

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.6 ibcmda

@ Description

Send GPIB commands asynchronously. Command bytes are used to configure the state of the GPIB. They are not used to send instructions to GPIB devices. Use ibwrt to send device-specific instructions. The number of command bytes transferred is returned in the global variable ibcntl.

The asynchronous I/O calls (ibcmda, ibrda, ibwrta) are designed so that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O has begun, further GPIB calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed, the driver returns EOIP in this case.

Once the I/O is complete, the application must resynchronize with the adlgpib driver.

Resynchronization is accomplished by using one of the following three functions:

ibwait If the returned ibsta mask has the CMPL bit set, the driver and application are resynchronized.

ibnotify If the ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.

ibstop The I/O is canceled; the driver and application are resynchronized.

ibonl The I/O is canceled and the interface is reset; the driver and application are resynchronized.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Visual Basic

```
Ilcmda (ByVal ud As Integer, ByVal buf As String,
     ByVal cnt As Long) As Integer
or
call ibcmda (ByVal ud As Integer, ByVal buf As
     String)
```

@ Parameter

ud: device unit descriptor

buf: the buffer contains command string to sent

cnt: number of command bytes to sent

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.7 ibconfig

@ Description

Set the setting value of the selected configuration item.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: board or device unit descriptor

opt: the configuration item which wish to be changed. The valid option items are listed in the following tables.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Options (Constants)	Options (Values)	Legal Values
ibcPAD	0x0001	Set the primary address of the board.
ibcSAD	0x0002	Set the secondary address of the board.
ibcTMO	0x0003	Set the I/O timeout limit of the board.
ibcEOT	0x0004	Set the data termination mode for write operations.
ibcPPC	0x0005	Configures the board for parallel polls. Default: zero
ibcAUTOPOLL	0x0007	0: Disable automatic serial polling.
		1: Enable automatic serial polling.
ibcSC	0x000A	Request or release system control. Identical to ibrsc.
ibcSRE	0x000B	Assert the Remote Enable (REN) line. Identical to ibsre.
		Default: zero.
ibcEOSrd	0x000C	0: Ignore EOS character during read operations.
		1: Terminate reads when the EOS character is read match occurs.
ibcEOSwrt	0x000D	0: Do not assert EOI with the EOS character during write operations.
		1: Assert EOI with the EOS character during writes operations.
ibcEOScmp	0x000E	0: Use 7 bits for the EOS character comparison.
		1: Use 8 bits for the EOS character comparison.
ibcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character.
ibcPP2	0x0010	0: PP1 mode-remote parallel poll configuration.
		1: PP2 mode-local parallel poll configuration.
		Default: zero.
ibcTIMING	0x0011	1: Normal timing (T1 delay of 2 µs.)
		2: High speed timing (T1 delay of 500 ns.)
		3: Very high speed timing (T1 delay of 350 ns.).
		The T1 delay is the GPIB source handshake timing.
		Default : 3

Table 2-3: Board Configuration Parameter Options

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Options (Constants)	Options (Values)	Legal Values
ibcReadAdjust	0x0013	0 = No byte swapping.
		1 = Swap pairs of bytes during a read. Default: zero.
ibcWriteAdjust	0x0014	0 = No byte swapping.
		1 = Swap pairs of bytes during a write.
		Default: zero.
ibcSpollBit	0x0016	0: The SPOLL bit of ibsta is disabled.
		1: The SPOLL bit of ibsta is enabled.
		Default: zero.
ibcSendLLO	0x0017	0: Do not send LLO when putting a device online – ibfind or ibdev.
		1: Send LLO when putting a device online–ibfind or ibdev.
		Default: zero.
ibcPPollTime	0x0019	0: Use the standard duration (2 µs) when conducting a parallel poll.
		1 to 17: Use a variable length duration when conducting a parallel poll. The duration represented by 1 to 17 corresponds to the ibtmo values.
		Default: zero.
ibcEndBitIs-Normal	0x001A	0: Do not set the END bit of ibsta when an EOS match occurs during a read.
		1: Set the END bit of ibsta when an EOS match occurs during a read.
		Default: 1.
ibcIst	0x0020	Sets the individual status (ist) bit of the board.
ibcRsv	0x0021	Sets the serial poll status byte of the board.
		Default: zero.

Table 2-3: Board Configuration Parameter Options

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Options (Constants)	Options (Values)	Legal Values
ibcPAD	0x0001	Set the primary address of the board.
ibcSAD	0x0002	Set the secondary address of the board.
ibcTMO	0x0003	Set the I/O timeout limit of the board.
ibcEOT	0x0004	Set the data termination mode for write operations.
IbcREADDR	0x0006	0: No unnecessary readdressing is performed between device-level reads and writes.
		1: Addressing is always performed before a device-level read or write.
ibcEOSrd	0x000C	0: Ignore EOS character during read operations.
		1: Terminate reads when the EOS character is read match occurs.
ibcEOSwrt	0x000D	0: Do not assert EOI with the EOS character during write operations.
		1: Assert EOI with the EOS character during writes operations.
ibcEOScmp	0x000E	0: Use 7 bits for the EOS character comparison.
		1: Use 8 bits for the EOS character comparison.
ibcEOSchar	0x000F	Any 8-bit value. This byte becomes the new EOS character.
ibcSPollTime	0x0018	0 to 17 : Sets the length of time the driver waits for a serial poll response byte when polling the given device. The length of time represented by 0 to 17 corresponds to the ibtmo values. Default: 11.
ibcEndBitIs-Normal	0x001A	0: Do not set the END bit of ibsta when an EOS match occurs during a read.
		1: Set the END bit of ibsta when an EOS match occurs during a read. Default: 1.

Table 2-4: Device Configuration Parameter Options

`value`: the value wish to be changed to the specified configuration item.

@ Return Code

The value of ibsta.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Possiable Error Codes

EARG, ECAP, EDVR, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.8 ibdev

@ Description

Open and initialize a device descriptor. If ibdev is unable to get a valid device descriptor, a -1 is returned; the ERR bit is set in ibsta and iberr contains EDVR.

@ Support Level

Device level

@ Syntax

Microsoft C/C++ and Borland C++

```
Int ibdev( int board_index, int pad, int sad, int
     timo, int send_eoi, int eosmode )
```

Visual Basic

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

@ Parameter

board_index :the index of the access board for the device

pad: the primary GPIB address of the device

sad: the second GPIB address of the device

tmo: the I/O timeout value

eot: enable or disable EOI mode of the devcie

eos: configure EOS character and EOS modes of the devcie

@ Return Code

The device descriptor or -1.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Possiable Error Codes

EARG, EDVR, ENEB,

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.9 ibdma

@ Description

This fucntion is not supported in adlgpib.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board descriptor

dma: enable or disable dma mode

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, ECAP, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.10 ibeot

@ Description

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

If EOT mode is enabled, EOI is asserted when the last byte of a GPIB write is sent; otherwise, nothing occurs when the last byte is sent.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor

v: enable or disable eot mode

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.11 ibeos

@ Description

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

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

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor

v: EOS mode and character information. If v is zero, the EOS configuration is disabled. Otherwise, the low byte is the EOS character and the upper byte contains flags defining the EOS mode. The different EOS configurations and the corresponding values of v as the following table:

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Configuration bits A and C determine how to terminate read I/O operations. If bit A is set and bit C is clear, 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, 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, EOI is asserted when the written character matches the low seven bits of the EOS character. If bits B and C are both set, EOI is asserted when the written character matches all eight bits of the EOS character.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.12 ibfind

@ Description

Open and initialize a GPIB board descriptor. The returned board descriptor can be used in subsequent calls. ibfind performs the equivalent of an ibonl 1 to initialize the board descriptor. The descriptor returned by ibfind is valid until the board is put offline using ibonl 0. If ibfind is unable to get a valid descriptor, a -1 is returned; the ERR bit is set in ibsta and iberr contains EDVR.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`boardname`: the board name, e.g. gpib0.

@ Return Code

The board descriptor or -1.

@ Possiable Error Codes

```
EBUS, ECIC, EDVR, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.13 ibgts

@ Description

Set the board from active controller state to Standby Controller state. ibgts causes the GPIB interface to go to Standby Controller and the GPIB ATN line to be unasserted.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board descriptor

v: determines whether to perform acceptor handshaking

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EADR , EARG, ECIC, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.14 ibist

@ Description

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

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board descriptor

v: indicates whether to set or clear the ist bit

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.15 iblines

@ Description

Return the status of the GPIB control lines. The low-order byte (bits 0 through 7) of lines indicating the capability of the GPIB interface to sense the status of each GPIB control line. The upper byte (bits 8 through 15) indicates the GPIB control line state information. The following is the description 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), 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.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board descriptor

line_status: return GPIB control line state information

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.16 ibln

@ Description

Checks whether a device is present on the bus.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor. If ud is a board descriptor, the bus associated with that board is tested for Listeners. If ud is a device descriptor, ibln uses the access board associated with that device to test for Listeners. If a Listener is detected, a non-zero value is returned in found_listener. If no Listener is found, zero is returned

pad: the primary address of the device (a value between 0 and 30).

sad: the secondary address of the device (a value between 96 to 126 or NO_SAD or ALL_SAD, where NO_SAD is no secondary address is to be tested, i.e. only a primary address is tested and ALL_SAD designates that all secondary addresses are to be tested),

found_listener: indicates if a device is present

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, ECIC, EDVR, ENEB, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.17 ibloc

@ Description

For a board, ibloc place the board in local mode, if it is not in a lockout state. The board is in a lockout state if LOK does not appear 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.

For a device, unless the REN (Remote Enable) line has been unasserted with the ibsre function, all device-level calls automatically place the specified device in remote program mode. ibloc is used to move devices temporarily from a remote program mode to a local mode until the next device function is executed on that device.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EBUS, ECIC, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.18 ibonl

@ Description

Resets the board or device, sets all its software configuration parameters in their pre-configured state and place the device online or offline. If a device or an interface is taken offline, the board or device descriptor is no longer valid. You have to call ibdev or ibfind to access the board or device again.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor.

onl: online (1) or offfline (0).

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, ENEB

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.19 ibnotify

@ Description

Notify user of one or more GPIB events by invoking the user specified callback.

After an asynchronous I/O operation has completed, resynchronization of the handler is required and the global variables passed into the Callback after I/O has completed contain the status of the I/O operation.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

@ Parameter

ud: the board or device descriptor.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

mask: bit mask of GPIB events. The valid event mask are the following:

- 0: no mask
- TIMO: the timeout period (see ibtmo) to limit the notify period
- END: END or EOS is detected
- SRQI: SRQ is asserted (board-level only)
- RQS: Device requested service (device-level only)
- CMP: I/O is complete
- LOK: GPIB interface is in Lockout State (board-level only)
- REM: GPIB interface is in Remote State (board-level only)
- CIC: GPIB interface is CIC (board-level only)
- ATN: Attention is asserted (board-level only)
- TACS: GPIB interface is Talker (board-level only)
- LACS: GPIB interface is Listener (board-level only)
- DTAS: GPIB interface is in Device Trigger State (boardlevel only)
- DCAS: GPIB interface is in Device Clear State (board-level only).

If mask is non-zero, ibnotify monitors the events specified by mask, and when one or more of the events is true, the Callback is invoked. For a board-level ibnotify call, all mask bits are valid except for ERR and RQS. For a device-level ibnotify call, the only valid mask bits are CMPL, TIMO, END, and RQS. If TIMO is set in the notify mask, ibnotify calls the callback function 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 notify mask, the callback is not called until one or more of the specified events occur.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Callback: the address callback function.

- Callback Prototype for ibnotify
  - int __std call Callback (int LocalUd, int LocalIbsta, int LocalIberr, long LocalIbcntl, void *RefData)
- Callback Parameters
  - LocalUd : Board or device descriptor
  - LocalIbsta : Value of ibsta
  - LocalIberr : Value of iberr
  - LocalIbcntl : Value of ibcntl
  - RefData : User-defined reference data for the callback
- Callback Return Value
  - Bit mask of the GPIB events to notice next.
- Possible Error Code
  - EDVR

RefData: user-defined reference data for the callback.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, ECAP, EDVR, ENEB, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.20 ibpad

@ Description

Set primary GPIB address of a board or a device.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor.

v: the GPIB primary address. The valid range of value is 0 through 30.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.21 ibsad

@ Description

Set or disable secondary GPIB address of a board or a device.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor.

v: set or disable the GPIB secondary address. If If v is zero, secondary addressing is disabled. If v is non-zero, the secondary address is enabled and valid range of value is 96 to 126 (0x60 to 0x7E).

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.22 ibpct

@ Description

Pass Controller-in-Charge (CIC) status to another GPIB device with Controller capability. The access board automatically unasserts the ATN line and goes to Controller Idle State (CIDS).

@ Support Level

Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, EBUS, ECIC, EDVR, ENEB, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.23 ibppc

@ Description

Configure Parallel Polling.

If ud is a device descriptor, ibppc enables or disables the device from responding to parallel polls. The device is addressed and sent the appropriate parallel poll message parallel Poll Enable (PPE) or Disable (PPD). Valid parallel poll messages are 96 to 126 (hex 60 to hex 7E) or zero to send PPD.

If ud is a board descriptor, ibppc performs a local parallel poll configuration 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, iberr contains the previous value of the local parallel poll configuration.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: The device descriptor.

v: parallel poll enable/disable value.

@ Return Code

The value of ibsta.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.24 ibrd

@ Description

Reads data from a device into the user specified buffer.

If ud is a device descriptor, ibrd addresses the GPIB, reads up to count bytes of data, and places the data into the user buffer. 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. The actual number of bytes transferred is returned in the global variable ibcntl.

If ud is a board descriptor, ibrd reads up to count bytes of data and places the data into the buffer. A board-level 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 ibcntl.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

buf: the buffer to store the data read from the GPIB.

cnt: number of bytes to be read from the GPIB.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.25 ibrda

@ Description

Reads data asynchronously from a device into the user specified buffer .

If ud is a device descriptor, ibrda addresses the GPIB, reads up to count bytes of data, and places the data into the buffer. 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. The actual number of bytes transferred is returned in the global variable ibcntl.

If ud is a board descriptor, ibrda reads up to count bytes of data and places the data into the buffer. A board-level ibrda 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 ibcntl.

The asynchronous I/O calls (ibcmda, ibrda, ibwrta) are designed so that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O has begun, further GPIB calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed, the driver returns EOIP in this case.

Once the I/O is complete, the application must resynchronize with the adlgpib driver.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Resynchronization is accomplished by using one of the following three functions:

- ibwait: If the returned ibsta mask has the CMPL bit set, the driver and application are resynchronized.
- ibnotify: If the ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- ibstop: The I/O is canceled; the driver and application are resynchronized.
- ibonl: The I/O is canceled and the interface is reset; the driver and application are resynchronized.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

buf: the buffer to store the data read from the GPIB.

cnt: number of bytes to be read from the GPIB.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.26 ibrdf

@ Description

Reads data from a device into a file.

If ud is a device descriptor, ibrdf addresses the GPIB, reads data from a GPIB device, and places the data into the file. The operation terminates normally when END is received. The operation terminates with an error if the transfer could not complete within the timeout period. The actual number of bytes transferred is returned in the global variable ibcntl.

If ud is a board descriptor, ibrdf reads data from a GPIB device and places the data into the file. A board-level ibrdf assumes that the GPIB is already properly addressed. The operation terminates normally when 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 ibcntl.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

filename: the name of the file where the read data are stored.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.27 ibrpp

@ Description

Perform a parallel poll.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

ppr: the result of parallel poll.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EBUS, ECIC, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.28 ibrsc

@ Description

Request or release the System Controller capability by sending Interface Clear (IFC) and Remote Enable (REN) messages to devices. If the board releases system control, perform opertions requiring System Controller capability are not allowed. If the board requests system control, calls operation requiring System Controller capability are subsequently allowed.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

v: 0: release system control

1: request system control.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, EDVR, ENEB, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.29 ibrsp

@ Description

Perform a serial poll. If bit 6 (hex 40) of the response is set, the device is requesting service. When the automatic serial polling feature is enabled, the device might have already been polled. In this case, ibrsp returns the previously acquired status byte.

@ Support Level

Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

spr: the result of serial poll.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EABO, EARG, EBUS, ECIC, EDVR, ENEB, EOIP, ESTB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.30 ibrsv

@ Description

Request service and change the serial poll status byte.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

v: Serial poll status byte.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.31 ibsic

@ Description

Asserts the GPIB interfaces clear (IFC) line for at least 100 ns if the GPIB interface is System Controller. This initializes the GPIB and makes the interface CIC and Active Controller with ATN asserted.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, EDVR, ENEB, EOIP, ESAC

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.32 ibsre

@ Description

Set or clear the Remote Enable (REN) line. If remote enable line is set, the GPIB Remote Enable (REN) line is asserted. If remote enable line is cleared, REN is unasserted. REN is used by devices to choose between local and remote modes of operation. A device should not actually enter remote mode until it receives its listen address and REN is asserted.

@ Support Level

Board level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board descriptor.

v: 0: clear REN line.

1: set REN line

@ Return Code

The value of ibsta.

@ Possiable Error Codes

EARG, EDVR, ENEB, EOIP, ESAC

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.33 ibstop

@ Description

Abort asynchronous mode of I/O operation. If asynchronous I/O is in progress, the error bit is set in the status word, ibsta, and EABO is returned, indicating that the I/O was successfully stopped.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

```
ilstop (ByVal ud As Integer) As Integer
or
call ibstop(ByVal ud As Integer)
```

@ Parameter

ud: the board or device descriptor.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EABO, EBUS, EDVR, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.34 ibtmo

@ Description

Set the timeout period of the board or device. The timeout period is the maximum duration allowed for a synchronous I/O operation (for example, ibrd and ibwrt) or for an ibwait or ibnotify operation with TIMO in the wait mask. If the operation does not complete before the timeout period elapses, the operation is aborted and TIMO is returned in ibsta.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor.

v: timeout code. The valid timeout codes are the following:

Constant	Value of v	MinimumTimeout
TNONE	0	Disabled - no timeout
T10us	1	10 µs
T30us	2	30 µs
T100us	3	100 µs
T300us	4	300 µs
T1ms	5	1 ms
T3ms	6	3 ms

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Constant	Value of v	MinimumTimeout
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms
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

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.35 ibtrg

@ Description

Send the Group Execute Trigger (GET) message to the device.

@ Support Level

Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the device descriptor.

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, ENEB, EOIP
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.36 ibwait

@ Description

Monitor the events specified by mask and delays processing until one or more of the events occurs. If the wait mask is zero, ibwait returns immediately with the updated ibsta. 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, the function waits indefinitely for one or more of the specified events to occur. The existing ibwait mask bits are identical to the ibsta bits. If ud is a device descriptor, the only valid wait mask bits are TIMO, END, RQS, and CMPL. If ud is a board descriptor, all wait mask bits are valid except for RQS. You can configure the timeout period using the ibtmo function.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: the board or device descriptor.

mask: GPIB events to wait for. The valid mask codes are listed in the following table:

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Mask	Bit Pos.	Hex Value	Description
END	13	2000	GPIB board detected END or EOS
SRQI	12	1000	SRQ asserted (board only)
RQS	11	800	Device requesting service (device only)
SPOLL	10	400	The board has been serial polled by theController
EVENT	9	200	A DTAS, DCAS, or IFC event has occurred
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
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

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, ENEB, ESRQ
```

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.37 ibwrt

@ Description

Write data to a device from a data buffer.

If ud is a device descriptor, ibwrt addresses the GPIB and writes count bytes from the memory to a GPIB device. 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. The actual number of bytes transferred is returned in the global variable ibcntl.

If ud is a board descriptor, ibwrt writes count bytes of data from the buffer to a GPIB device; a board-level 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 ibcntl.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: device unit descriptor

buf: the buffer contains data bytes to sent

cnt: number of data bytes to sent

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.38 ibwrta

@ Description

Write data asynchronously to a device from a data buffer.

If ud is a device descriptor, ibwrt addresses the GPIB and writes count bytes from the memory to a GPIB device. 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. The actual number of bytes transferred is returned in the global variable ibcntl.

If ud is a board descriptor, ibwrt writes count bytes of data from the buffer to a GPIB device; a board-level 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 ibcntl.

The asynchronous I/O calls (ibcmda, ibrda, ibwrta) are designed so that applications can perform other non-GPIB operations while the I/O is in progress. Once the asynchronous I/O has begun, further GPIB calls are strictly limited. Any calls that would interfere with the I/O in progress are not allowed, the driver returns EOIP in this case.

Once the I/O is complete, the application must resynchronize with the adlgpib driver.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Resynchronization is accomplished by using one of the following three functions:

- ibwait: If the returned ibsta mask has the CMPL bit set, the driver and application are resynchronized.
- ibnotify: If the ibsta value passed to the ibnotify callback contains CMPL, the driver and application are resynchronized.
- ibstop: The I/O is canceled; the driver and application are resynchronized.
- ibonl: The I/O is canceled and the interface is reset; the driver and application are resynchronized.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

ud: device unit descriptor

buf: the buffer contains data bytes to sent

cnt: number of data bytes to sent

@ Return Code

The value of ibsta.

@ Possiable Error Codes

```
EADR, EABO, EBUS, ECIC, EDVR, EOIP, ENEB, ENOL
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.39 ibwrtf

@ Description

Write data to a device from a file.

If ud is a device descriptor, ibwrtf addresses the GPIB and writes all of the bytes from the file flname to a GPIB device. The operation terminates normally when all of the bytes have been sent. The operation terminates with an error if all of the bytes could not be sent within the timeout period. The actual number of bytes transferred is returned in the global variable ibcntl.

If ud is a board descriptor, ibwrtf writes all of the bytes of data from the file flname to a GPIB device. A board-level ibwrtf assumes that the GPIB is already properly addressed. The operation terminates normally when all of the bytes have been sent. The operation terminates with an error if all of the bytes could not be sent 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 ibcntl.

@ Support Level

Board / Device level

@ Syntax

Microsoft C/C++ and Borland C++

```
Int ibwrtf( int ud, const char *file_path )
```

Visual Basic

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

@ Parameter

ud: the device descriptor.

filename: the name of the file containing the data to write.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@ Return Code

The value of ibsta.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3 Multi-Device IEEE 488 Function Reference

3.1 AllSpoll

@ Description

Perform serial poll one or more devices. The poll responses are stores in resultList and the number of responses in ibcntl.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addressList`: the list of device addresses that is terminated by NOADDR

`resultList`: the list of serial poll response bytes corresponding to device addresses in addrlist

@ Possiable Error Codes

```
EARG, EABO, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.2 DevClear

@ Description

Send the Selected Device Clear (SDC) GPIB message to clear a device. If address is the constant NOADDR, the Universal Device Clear (DCL) message is sent to all devices.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

address: the device address wishing to be cleared

@ Possiable Error Codes

EARG, EBUS, ECIC, EDVR, EOIP, ENEB

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.3 DevClearList

@ Description

Clear mutiple devices. If address is the constant NOADDR, the Universal Device Clear (DCL) message is sent to all devices.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addressList`: a list of the device addresses terminated by NOADDR wishing to be cleared

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.4 EnableLocal

@ Description

Enable operations from the front panel of devices by sending the Go To Local (GTL) GPIB message to multiple devices. This places the devices into local mode. If addrlist contains only the constant NOADDR, the Remote Enable (REN) GPIB line is unasserted.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

addressList: a list of the device addresses terminated by NOADDR wishing to go to local.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.5 EnableRemote

@ Description

Enable remote GPIB programming for devices by asserting the Remote Enable (REN) GPIB line. The devices are put into a listen-active state.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addressList`: a list of the device addresses terminated by NOADDR wishing to go to local.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ESAC
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.6 FindLstn

@ Description

Find listening devices on the GPIB bus. This function tests all of the primary addresses in padlist as follows: If a device is present at a primary address given in padlist, the primary address is stored in resultlist. Otherwise, all secondary addresses of the primary address are tested, and the addresses of any devices found are stored in resultlist. ibcntl contains the actual number of addresses stored in resultlist.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`padList`: a list of the gpib primary addresses terminated by NOADDR.

`resultList`: addresses of all listening devices found by FindLstn.

`maxNumResults`: maximum count of entries that can be placed in resultList.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.7 FindRQS

@ Description

Serial poll the devices to determine which device is requesting service, until it finds a device which is requesting service. The serial poll response byte is placed in result. ibcntl contains the index of the device requesting service in addrList. If none of the devices are requesting service, the index corresponding to NOADDR in addrlist is returned in ibcntl and ETAB is returned in iberr.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addressList`: a list of the gpib primary addresses terminated by NOADDR.

`result`: Serial poll response byte of the device that is requesting service.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ETAB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.8 PassControl

@ Description

Pass control to another GPIB device with Controller capability by sending the Take Control (TCT) GPIB message to the device. The device becomes Controller-In-Charge (CIC) and the interface is no longer CIC.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

address: a list of the gpib primary addresses terminated by NOADDR.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.9 PPoll

@ Description

Perform a parallel poll. The board sents command to each device (see PPollConfig and PPollUnconfig). The Controller can use parallel polling to obtain one-bit, device-dependent status messages from up to eight devices simultaneously

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`result`: The parallel poll result.

@ Possiable Error Codes

```
EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.10 PPollConfig

@ Description

Configures the device to respond to parallel polls by asserting or not asserting the GPIB data line, dataline. If lineSense equals the individual status (ist) bit of the device, the assigned GPIB data line is asserted during a parallel poll. Otherwise, the data line is not asserted during a parallel poll. The Controller can use parallel polling to obtain 1-bit, device-dependent status messages from up to eight devices simultaneously.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

address: address of the device to be configured.

dataLine: Data line (a value in the range of 1 to 8) on which the device responds to parallel polls.

lineSense: Sense (either 0 or 1) of the parallel poll response.

@ Possiable Error Codes

EARG, EBUS, ECIC, EDVR, EOIP, ENEB

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.11 PPollUnConfig

@ Description

Unconfigures the device to respond to parallel polls. If addrlist contains only the constant NOADDR, the Parallel Poll Unconfigure (PPU) GPIB message is sent to all GPIB devices. The devices unconfigured by this function do not participate in subsequent parallel polls.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addressList`: A list of device addresses that is terminated by NOADDR.

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.12 RcvRespMsg

@ Description

Read data bytes from a device. RcvRespMsg assumes that the interface is already in its listen-active state and a device is already addressed to be a Talker. Data are read until either count data bytes have been read or the termination condition is detected. If the termination condition is STOPend, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when the 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable, ibcntl.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`buffer`: the buffer which stores the read data.

`count`: Number of bytes read.

`termination`: Description of the data termination mode (STOPend or an 8-bit EOS character).

@ Possiable Error Codes

`EABO, EADR, EARG, ECIC, EDVR, EOIP, ENEB`

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.13 ReadStatusByte

@ Description

Conduct serial polling single device. If bit 6 (hex 40) of the response is set, the device is requesting service.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

address: device address.

result: the serial poll response byte.

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.14 Receive

@ Description

Reads data from a device into the user specified buffer.

Receive addresses the device described by address to talk and the interface to listen, reads up to count bytes of data, and places the data into the buffer. The operation terminates normally when count bytes have been received or the termination condition is detected. If the termination condition is STOPend, the read is stopped when a byte is received with the EOI line asserted. Otherwise, the read is stopped when an 8-bit EOS character is detected. The actual number of bytes transferred is returned in the global variable, ibcntl.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`address`: address of the device to read data.

`buffer`: the buffer which stores the read data

`termination`: the data termination mode (STOPend or an EOS character)

@ Possiable Error Codes

```
EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.15 ReceiveSetup

@ Description

Set a device to be a Talker and the interface to be a Listener. This funciton is usually followed by a call to RcvRespMsg to transfer data from the device to the interface. This call is useful to make multiple calls to RcvRspMsg; it eliminates the need to readdress the device to receive every block of data.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`address`: address of the device addressed to be a talker

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.16 ResetSys

@ Description

Reset and initialize devices. It includes three steps. The first step resets the GPIB by asserting the Remote Enable (REN) line and then the Interface Clear (IFC) line. The second step clears all of the devices by sending the Universal Device Clear (DCL) GPIB message. The final step causes devices to perform device-specific reset and initialization. This step is accomplished by sending the message "*RST\n" to the devices described by addrlist.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

addressList: list of the device addresses that is terminated by NOADDR

@ Possiable Error Codes

```
EABO, EARG, EBUS, ECIC, EDVR, ENOL, EOIP, ENEB,
     ESAC
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.17 Send

@ Description

Write data to a device from a data buffer.

The operation terminates normally when count bytes have been sent. The last byte is sent with the EOI line asserted if eotmode is DABend. The last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend then a new line character ('\n') is sent with the EOI line asserted after the last byte of buffer. The actual number of bytes transferred is returned in the global variable, ibcntl.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

address: device address

buffer: the data bytes to be sent

count: data count

eot_mode: the data termination mode: DABend, NULLend, or NLend

@ Possiable Error Codes

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.18 SendCmds

@ Description

Send GPIB command. The number of command bytes transferred is returned in the global variable ibcntl

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`cmdbuf`: Command bytes to be sent

`count`: data count

@ Possiable Error Codes

```
EABO, ECIC, EDVR, ENOL, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.19 SendDataBytes

@ Description

Send number of bytes from the buffer to devices. SendDataBytes assumes that the interface is in talk-active state and that devices are already addressed as Listeners on the GPIB. The last byte is sent with the EOI line asserted if eotmode is DABend; the last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend then a new line character ('\n') is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global variable, ibcntl.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`buffer`: the data bytes to be sent

`count`: data count

`eot_mode`: the data termination mode: DABend, NULLend, or NLend

@ Possiable Error Codes

```
EABO, EADR, EARG, EBUS, ECIC, EDVR, ENOL, EOIP,
     ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.20 SendList

@ Description

Send data bytes to multiple GPIB devices. SendList addresses the devices described by addrlist to listen and the interface to talk and then data from buffer are sent to the devices. The last byte is sent with the EOI line asserted if eotmode is DABend. The last byte is sent without the EOI line asserted if eotmode is NULLend. If eotmode is NLend, a new line character ('\n') is sent with the EOI line asserted after the last byte. The actual number of bytes transferred is returned in the global variable, ibcntl.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

addresslist: list of device addresses to send data

buffer: the data bytes to be sent

count: data count

eot_mode: the data termination mode: DABend, NULLend, or NLend

@ Possiable Error Codes

```
EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.21 SendIFC

@ Description

Reset the GPIB by sending interface clear. SendIFC is used as part of GPIB initialization. It forces the interface to be Controller-In-Charge of the GPIB. It also ensures that the connected devices are all un-addressed and that the interface calls of the devices are in their idle states.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

@ Possiable Error Codes

ENEB, ESAC, EDVR, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.22 SendLLO

@ Description

Send the Local Lockout (LLO) message to all devices. While Local Lockout is in effect, only the Controller-In-Charge can alter the state of the devices by sending appropriate GPIB messages.

SendLLO is reserved for use in unusual local/remote situations. In the typical case of placing the devices in Remote With Local Lockout, use SetRWLS.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

@ Possiable Error Codes

EBUS, ECIC, ENEB, ESAC, EDVR, EOIP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.23 SendSetup

@ Description

Set up devices to receive data. SendSetup makes the devices described by addressList listen-active and makes the interface talk-active. This call is usually followed by SendDataBytes to actually transfer data from the interface to the devices. SendSetup is particularly useful to set up the addressing before making multiple calls to SendDataBytes; it eliminates the need to readdress the devices for every block of data.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

addresslist: list of device addresses that is terminated by NOADDR

@ Possiable Error Codes

```
EABO, EARG, EBUS, ECIC, EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.24 SetRWLS

@ Description

Place devices in Remote With Lockout State. SetRWLS places the devices described by addrlist in remote mode by asserting the Remote Enable (REN) GPIB line. Then those devices are placed in lockout state by the Local Lockout (LLO) GPIB message. You cannot program those devices locally until the Controller-In-Charge releases the Local Lockout by way of the EnableLocal call.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

board_desc: board id

addresslist: list of device addresses that is terminated by NOADDR

@ Possiable Error Codes

```
EARG, EBUS, ECIC, EDVR, EOIP, ENEB, ESAC
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.25 TestSRQ

@ Description

Check the current state of the GPIB Service Request (SRQ) line. If SRQ is asserted, result contains a non-zero value. Otherwise, result contains a zero. Use TestSRQ to get the current state of the GPIB SRQ line. Use WaitSRQ to wait until SRQ is asserted.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`result`: State of the SRQ line: non-zero if the line is asserted, zero if the line is not asserted.

@ Possiable Error Codes

```
EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.26 TestSys

@ Description

Make the devices to conduct self tests. TestSys sends the "*TST?" message to the devices. The "*TST?" message makes them to conduct their self-test procedures. A 16-bit test result code is read from each device. A test result of 0\n indicates that the device passed its self test. Refer to the documentation that came with the device to determine the meaning of the failure code. Any other value indicates that the device failed its self test. If the function returns without an error (that is, the ERR bit is not set in ibsta), ibcntl contains the number of services that failed. Otherwise, the meaning of ibcntl depends on the error returned. If a device fails to send a response before the timeout period expires, a test result of ? is reported for it, and the error EABO is returned.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addrlist`: a list of device addresses terminated by NOADDR.

`resultList`: A list of test results; each entry corresponds to an address in addrlist.

@ Possiable Error Codes

```
EABO, EARG, EBUS, EDVR, ECIC, EOIP, ENEB, ENOL
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.27 Trigger

@ Description

Send the Group Execute Trigger (GET) GPIB message to single device. If address is the constant NOADDR, the GET message is sent to all devices that are currently listen-active on the GPIB.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`address`: the address of the device to be triggered.

@ Possiable Error Codes

```
EARG, EBUS, EDVR, ECIC, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.28 TriggerList

@ Description

Send the Group Execute Trigger (GET) GPIB message to multiple devices. If the only address in addrlist is the constant NOADDR, no addressing is performed and the GET message is sent to all devices that are currently listen-active on the GPIB.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`addressList`: a list of device addresses terminated by NOADDR.

@ Possiable Error Codes

```
EARG, EBUS, EDVR, ECIC, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.29 WaitSRQ

@ Description

Wait until a device asserts the GPIB Service Request (SRQ) line. When WaitSRQ returns, result contains a non-zero if SRQ is asserted. Otherwise, result contains a zero. Use TestSRQ to get the current state of the GPIB SRQ line. Use WaitSRQ to wait until SRQ is asserted.

@ Syntax

Microsoft C/C++ and Borland C++

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

Visual Basic

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

@ Parameter

`board_desc`: board id

`result`: State of the SRQ line: non-zero if line is asserted, zero if line is not asserted.

@ Possiable Error Codes

```
EDVR, EOIP, ENEB
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Appendix

Appendix A: Status Codes

All calls update a global status word, ibsta, which contains information about the state of the GPIB and your GPIB hardware. You can check for errors after each call using the ibsta ERR bit.

ibsta is a 16-bit value. A bit value of one (1) indicates that a certain condition is happened. A bit value of zero (0) indicates that the condition is not happened.

Mnemonic	BitPos.	HexValue	Type	Description
ERR	15	8000	device, board	GPIB error
TIMO	14	4000	device, board	Timeout
END	13	2000	device, board	END or EOS detected
SRQI	12	1000	board	SRQ interrupt occurred
RQS	11	800	device	Device requesting service
SPOLL	10	400	board	Board has been serial polled by Controller
EVENT	9	200	board	DCAS, DTAS, or IFC event has occurred
CMPL	8	100	device, board	I/O completed
LOK	7	80	board	Lockout State
REM	6	40	board	Remote State
CIC	5	20	board	Controller-In-Charge
ATN	4	10	board	Attention is asserted
TACS	3	8	board	Talker
LACS	2	4	board	Listener
DTAS	1	2	board	Device Trigger State
DCAS	0	1	board	Device Clear State

Table 4-1: Status Codes

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Appendix B: Error Codes

The following table lists the ADL-GPIB error codes. Remember that the error variable is meaningful only when the ERR bit in the status variable, ibsta, is set. For a detailed description of each error and possible solutions, click on the error mnemonic.

ErrorMnemonic	iberrValue	Meaning
EDVR	0	Operating 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
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
EDMA	8	DMA error
EOIP	10	Asynchronous I/O in progress
ECAP	11	No capability for operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial poll status byte queue overflow
ESRQ	16	SRQ stuck in ON position
ETAB	20	Table problem

Table 4-2: Error Codes

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Warranty Policy

Thank you for choosing ADLINK. To understand your rights and enjoy all the after-sales services we offer, please read the following carefully.

1. Before using ADLINK's products please read the user manual and follow the instructions exactly. When sending in damaged products for repair, please attach an RMA application form which can be downloaded from: http://rma.adlinktech.com/policy/.
2. All ADLINK products come with a limited two-year warranty, one year for products bought in China:
   - The warranty period starts on the day the product is shipped from ADLINK's factory.
   - Peripherals and third-party products not manufactured by ADLINK will be covered by the original manufacturers' warranty.
   - For products containing storage devices (hard drives, flash cards, etc.), please back up your data before sending them for repair. ADLINK is not responsible for any loss of data.
   - Please ensure the use of properly licensed software with our systems. ADLINK does not condone the use of pirated software and will not service systems using such software. ADLINK will not be held legally responsible for products shipped with unlicensed software installed by the user.
   - For general repairs, please do not include peripheral accessories. If peripherals need to be included, be certain to specify which items you sent on the RMA Request & Confirmation Form. ADLINK is not responsible for items not listed on the RMA Request & Confirmation Form.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3. Our repair service is not covered by ADLINK's guarantee in the following situations:
   - Damage caused by not following instructions in the User's Manual.
   - Damage caused by carelessness on the user's part during product transportation.
   - Damage caused by fire, earthquakes, floods, lightening, pollution, other acts of God, and/or incorrect usage of voltage transformers.
   - Damage caused by unsuitable storage environments (i.e. high temperatures, high humidity, or volatile chemicals).
   - Damage caused by leakage of battery fluid during or after change of batteries by customer/user.
   - Damage from improper repair by unauthorized ADLINK technicians.
   - Products with altered and/or damaged serial numbers are not entitled to our service.
   - This warranty is not transferable or extendible.
   - Other categories not protected under our warranty.
4. Customers are responsible for shipping costs to transport damaged products to our company or sales office.
5. To ensure the speed and quality of product repair, please download an RMA application form from our company website: http://rma.adlinktech.com/policy. Damaged products with attached RMA forms receive priority.

If you have any further questions, please email our FAE staff: service@adlinktech.com.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000