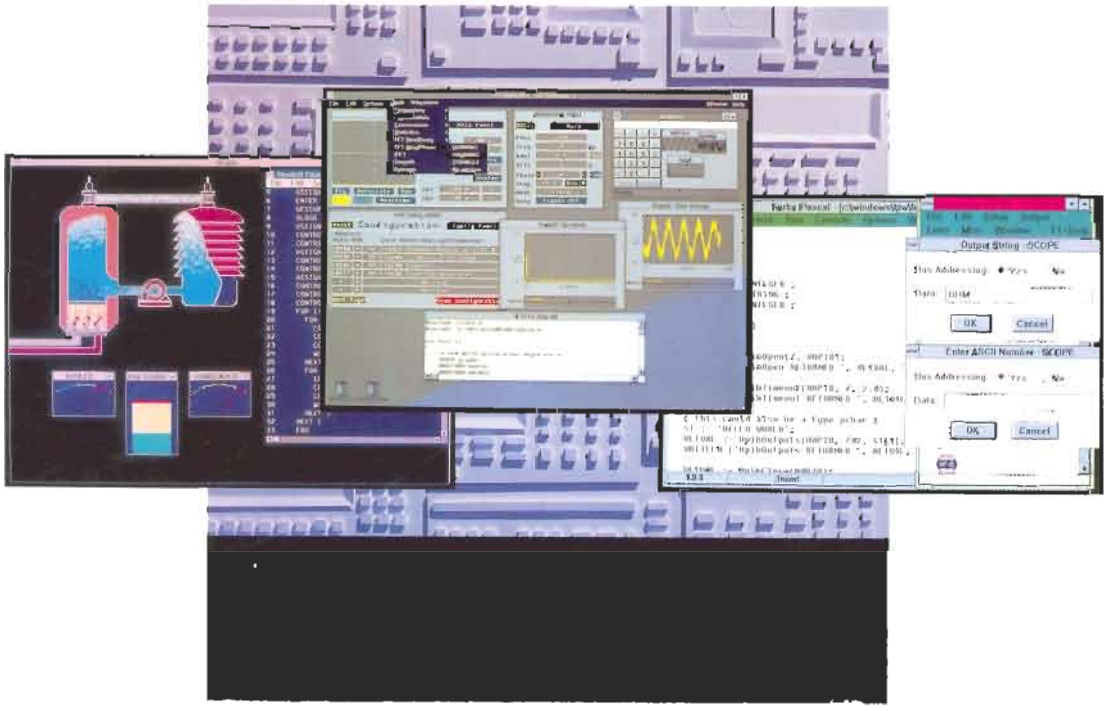
HEWLETT PACKARD

Using the HP-IB Interface with Microsoft® Windows.

Instrument Tools for Windows.



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard Company (HP) shall not be liable for any errors contained in this document. HP MAKES NO WARRANTIES OF ANY KIND WITH REGARD TO THIS DOCUMENT, WHETHER EXPRESS OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

Warranty Information

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Printing History

First Edition - December 1991

Contents

1

Introduction

Hewlett-Packard's Windows tools for HP-IB provide you, a Microsoft® Windows user, with everything you need to access and operate HP-IB instruments using your Windows development tools and applications. *Using the HP-IB Interface with Microsoft Windows* describes the three methods available to you for accessing instruments with Windows:

- Using the Interactive HP-IB Environment to interactively control the HP-IB bus.
- Using an application that has Dynamic Data Exchange (DDE) capabilities.
- Using the HP-IB Dynamic Link Library (DLL), which can be used in conjunction with Windows programming languages and other development tools.

Each method is distinct from the others, providing a selection to accommodate your needs and experience level.

The **Interactive HP-IB Environment** is for those who want to perform unrepetitive tasks such as simple instrument control, program debugging, or finding out how an instrument will respond to certain commands. It is also used to generate configuration files for the HP-IB DDE Server described in chapter 3. Chapter 2 describes the Interactive HP-IB program and summarizes the menu selections.

The **HP-IB Dynamic Data Exchange Server** is for novice or experienced users who want to use their favorite application, such as Microsoft Excel, to control instruments. Chapter 3 describes the use of the HP-IB DDE server.

The **HP-IB Dynamic Link Library** (DLL) is for programmers who want to use a programming language or other Windows development tool to create their own Windows application. The *Supported Languages* sheet includes a list of programming languages supported by the DLL. Chapter 4 describes the DLL.

Additional background information about the HP-IB interface and HP-IB communications is available in chapter 1 and appendices A-C of the *Using the HP-IB Interface and Command Library with DOS* manual.

System Requirements

You must have the following components to set up and use these software tools for HP-IB:

- An HP Vectra® PC or IBM PC/XT/AT (or compatible) computer with Windows 3.0 or later installed.
- The HP 82335 HP-IB Interface, which should be installed in your PC. (See the *Installing the HP-IB Interface* booklet for instructions.)
- The master disks, which contain the software described on the previous page.

We also suggest that you read chapter 1 of *Using the HP-IB Interface and Command Library with DOS* for general information about the capabilities of your software tools for HP-IB.

2

Using the Interactive HP-IB Environment

Introduction

The Interactive HP-IB Environment enables you to interactively control the HP-IB bus. You can use it for such purposes as simple instrument control, debugging, and finding out how an instrument responds to commands. It also provides you with the environment to define the configuration of your instrument system for use with the HP-IB DDE Server and macro programming described in chapter 3.

Installing the Interactive HP-IB Environment

To begin using the Interactive HP-IB Environment, named HPIBINT.EXE, you must first install it on your hard disk. Your master disks contain an install program named WINSTALL, which copies all necessary files for you. To use WINSTALL:

1. Insert the master disk into your flexible disk drive.
2. Run Windows, since WINSTALL is a Windows application.
3. From the Program Manager, select File, then Run. In the dialog box, type `A:WINSTALL` and choose OK. If your master disk is not in drive A:, enter the appropriate drive letter.
4. Follow the instructions displayed on the screen to complete the installation process.

When WINSTALL is complete, you will have a new program group named HP-IB, which will contain all of the installed programs.

Getting Started with the Interactive HP-IB Environment

To begin using the Interactive HP-IB Environment, follow these steps:

1. Open the HP-IB program group and select the Interactive HP-IB icon by double clicking it with your mouse.
2. At the main window of the HP-IB Interactive Environment, select Setup and Add Device. At the Add Device dialog box, enter the name of the device, such as "Scope", up to 10 characters. Choose OK.
3. The Setup Device Address dialog box appears, showing all valid HP-IB Interface Select Codes (ISC), a Virtual Device box, and a Device Address box. Select the correct Interface Card number if it is not already selected for you. Then, if you want to simulate a conversation with a device (e.g., if you don't have a device or HP-IB interface available), select the Virtual Device box. Then type the address of your device (0 through 31). Choose OK. A box displays the default characteristics of the added device.

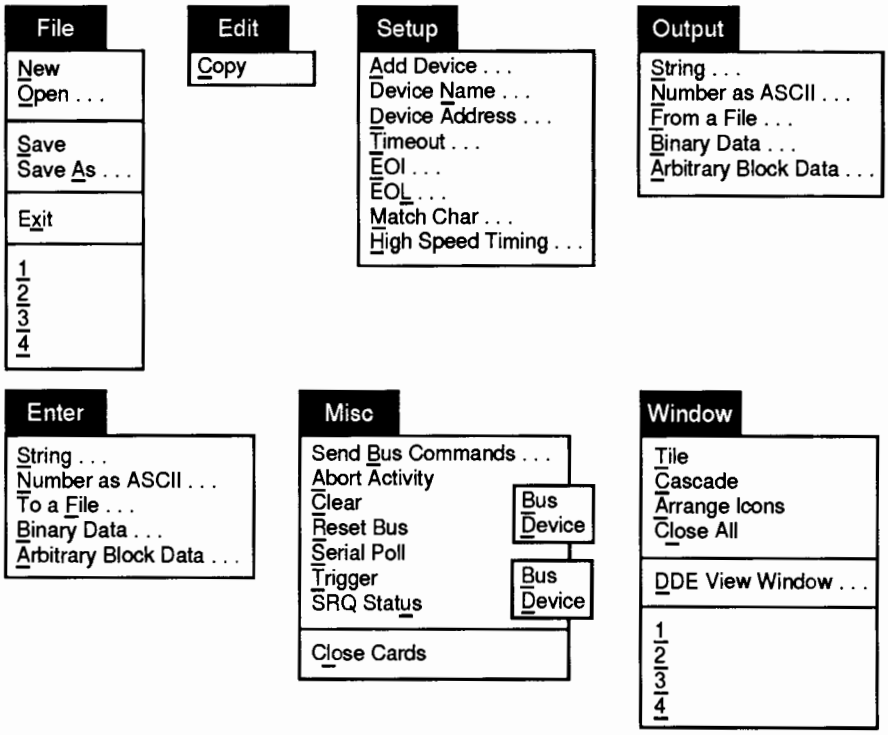
   Notice that a full menu is now displayed across the top of the window.
4. If you want to change any default settings for the instruments, click on the gray box that shows the current settings. A dialog box appears, in which you can set the new values. For example, you may want to disable the Match character if you are entering a binary file.
5. Repeat steps 2 through 4 until you have added as many instruments as you want, up to 14.

Now that you have at least one device available, you can use the other features of the Interactive HP-IB Environment to interactively control it. Refer to the following section for an explanation of each menu item.

You will want to save your setup information to use again without re-entering it. This configuration file that you save will also be used if you run the DDE server application described in chapter 3.

You can save your current configuration to disk by choosing Save or Save As from the File menu. Likewise, when you exit the Interactive HP-IB Environment, you are asked if you want to exit and save the configuration. If you choose Yes, the program exits and saves your current configuration to a file with the name you enter. More information about the configuration file is provided later in this chapter.

More About the Interactive HP-IB Environment



This section summarizes all of the possible menu picks for using the Interactive HP-IB Environment, HPIBINT.EXE.

An ellipsis (. . .) after a menu item indicates that a dialog box will appear when you select that item.

File Menu

New

Closes the current configuration and instrument windows so you can create a new configuration. If you have made any changes since saving the current configuration, a dialog box appears, asking if you want to save the changes to your current configuration.

Yes—Saves the current configuration. If you have not yet named the configuration, enter a file name up to 8 characters. A .IBC suffix is automatically appended unless you specify a different suffix. If you specify a different suffix, the interactive environment will *not* automatically identify it as a configuration file. This means that it will not appear on the list of choices shown when you select File Open. See "About the Configuration File" later in this chapter for more information about the configuration file.

No—Clears the window without saving your changes.

Cancel—Returns to the program without saving your changes.

Open ...

Opens a saved configuration file. If you have not saved your current configuration, a dialog box prompts you to save it or cancel. At the File Open dialog box, type a name in the File Name box or select from the list of files. If the file you want has a different suffix than .IBC, you will need to specify that extension.

Save

Saves the current configuration using its current name. The current configuration remains open so you can continue working with it.

Save As ...

Saves the current configuration, using options you select in the File Save dialog box. Type a new name in the Filename box.

OK—Accepts the new file name.

Cancel—Returns to the program without saving the configuration.

Exit

Exits the Interactive HP-IB Environment. If you have made any changes since saving the current configuration, a dialog box is displayed.

Yes—Saves the current configuration. If you have not yet named the configuration, enter a file name up to 8 characters. A .IBC suffix is automatically appended unless you specify a different suffix. If you specify a different suffix, the interactive environment will *not* automatically identify it as a configuration file. This means that it will not appear on the list of choices shown when you select File Open. See "About the Configuration File" later in this chapter for more information about the configuration file.

No—Exits the program without saving the configuration.

Cancel—Returns to the program without saving the configuration.

1,2,3,4

Lists the last four configurations you opened or saved so that you can quickly open them again without searching for them and without changing directories. If a file has been deleted, it may still appear on the list.

Edit Menu

Copy

Copies the data from the active Enter window to the clipboard, leaving the original data intact.

Setup Menu

Add Device ...

Brings up a dialog box in which you enter the name of a device to add. The name can be up to 10 characters. When you enter a name and choose OK, the Device Address dialog box appears. (See "Device Address", below.)

Device Name ...

Brings up a dialog box which enables you to change the name of the device.

Device Address ...

Brings up a dialog box which identifies all valid HP-IB Interface Select Codes (ISC) and a Device Address box. The valid select codes are highlighted, with the first valid code selected as the default. The ISC is determined by the switch settings on the interface card. The Device Address should be in the range 0 through 31, and is typically determined by switch settings on the instrument.

OK—Accepts the new device and displays a window that shows default values for commonly used settings.

Cancel—Cancels your selection.

Timeout ...

Allows you to set up a timeout value in seconds for I/O operations that do not complete (for example, the printer runs out of paper). This selection applies to the currently active instrument. Refer to the HpibTimeout command in chapter 4 for more information.

OK—Accepts the new timeout value.

Cancel—Cancels your selection.

EOI ...

Allows you to enable or disable EOI (End Or Identify) for the currently active instrument. Refer to the HpibEoi command in chapter 4 for more information.

OK—Accepts the selection you made.

Cancel—Cancels your selection and returns to the program.

EOL ...

Allows you to select one of four predefined EOL (End Of Line) strings (carriage return and linefeed, carriage return, linefeed, or semicolon), and to enable or disable the EOL string for the currently active instrument. You can also define your own EOL string using the Options button. Refer to the HpibEol command in chapter 4 for more information.

OK—Saves the selection.

Cancel—Returns EOL to its previous setting.

Default—Sets the EOL string to carriage return and linefeed, and enables it.

Options—Displays more information which lets you enter an arbitrary EOL string, up to a maximum of 8 ASCII characters.

Match Character ...

Allows you to set the termination character for the currently active instrument, and to enable or disable the match. Refer to the HpibMatch command in chapter 4 for more information.

OK—Accepts the termination character.

Default—Returns the termination character to 10 (linefeed), and enables it.

Cancel—Cancels your selection.

High Speed Timing ...

Allows you to enable or disable high speed timing when sending data on the bus. This selection applies to the currently active instrument. Refer to the HpibFastout command in chapter 4 for more information.

OK—Accepts the new high-speed timing selection.

Cancel—Cancels your selection.

Output Menu

String ...

Outputs a string to the currently active instrument. Bus addressing can be enabled (Yes) or disabled (No). This window remains open until you choose Cancel, enabling you to select another instrument and send a string to it without having to reselect Output and String. Refer to the HpibOutputs command in chapter 4 for more information.

OK—Accepts your selection.

Cancel—Returns you to the main menu.

Number as ASCII ...

Outputs a real number in its ASCII representation to the instrument defined by the currently active window. For example, if you output -6.234, first the ASCII negative sign is sent, then a 6, then a period, then 2, 3, and 4. Bus addressing can be enabled (Yes) or disabled (No). This output window remains open until you choose Cancel, enabling you to output data several times and to several instruments without having to reselect Output and Number as ASCII each time. Refer to the HpibOutput command in chapter 4 for more information.

OK—Accepts your selection.

Cancel—Cancels your selection.

From a File ...

Outputs the contents of a file to the currently active instrument. Bus addressing can be enabled (Yes) or disabled (No). This output window remains open until you choose Cancel, enabling you to output data several times and to several instruments without having to reselect Output and From a File each time. Refer to the HpibOutputf command in chapter 4 for more information.

OK—Accepts your selection.

Cancel—Cancels your selection.

Browse—Opens a dialog box to allow you to find a file easily.

Binary Data ...

Outputs binary data (numeric data with no coding or formatting) to the currently active instrument. Bus addressing can be enabled (Yes) or disabled (No). You can select `string` (up to 40 characters), `int` (2-byte signed integer), `long` (4-byte signed integer), `float` (4-byte real), or `double` (8-byte real) to output. You can also select whether to turn byte swapping off or on.

This output window remains open until you choose Cancel, enabling you to output data several times and to several instruments without having to reselect Output and Binary Data each time. Refer to the HpibOutputb command in chapter 4 for more information.

OK—Accepts your selection.

Cancel—Cancels your selection.

Arbitrary Block Data ...

Outputs arbitrary-block response data to the currently active instrument; the data has a header appended to the beginning as defined by the IEEE 488.2 arbitrary block format. Bus addressing can be enabled (Yes) or disabled (No). You can select `string` (up to 40 characters), `int` (2-byte signed integer), `long` (4-byte signed integer), `float` (4-byte real), or `double` (8-byte real) to output. You can also select whether to allow byte swapping.

This output window remains open until you choose Cancel, enabling you to output data several times and to several instruments without having to reselect Output and Arbitrary Block Data each time. Refer to the HpibOutputab command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

Enter Menu

String ...

Enters a string from the currently active instrument. Bus addressing can be enabled (Yes) or disabled (No). This window remains open until you choose Cancel, enabling you to enter data several times and from several instruments without having to reselect Enter and String each time. Refer to the HpibEnters command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

Number as ASCII ...

Enters an ASCII number from the currently active instrument, and converts it to a real number. Bus addressing can be enabled (Yes) or disabled (No). This window remains open until you choose Cancel, enabling you to enter data several times and from several instruments without having to reselect Enter and Number as ASCII each time. Refer to the HpibEnter command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

To a File ...

Reads data from the currently active instrument and outputs it directly to a file with no conversion. Bus addressing can be enabled (Yes) or disabled (No). The file can be appended to; if the Append to File box is not checked, any existing file will be overwritten. This window remains open until you choose Cancel, enabling you to enter data several times and from several instruments without having to reselect Enter and To a File each time. Refer to the HpibEnterf command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

Browse—Opens a dialog box to allow you to find a file easily.

Binary Data ...

Reads binary data (numeric data with no coding or formatting) from the currently active instrument. Bus addressing can be enabled (Yes) or disabled (No). You can choose how the data is to be interpreted by selecting `string` (up to 40 characters), `int` (2-byte signed integer), `long` (4-byte signed integer), `float` (4-byte real), or `double` (8-byte real). You can also select whether to allow byte swapping.

This window remains open until you choose Cancel, enabling you to enter data several times and from several instruments without having to reselect Enter and Binary Data each time. Refer to the HpibEnterb command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

Arbitrary Block Data ...

Reads arbitrary block data from the currently active instrument, and reads and interprets a header that is defined in IEEE 488.2 arbitrary block format. Bus addressing can be enabled (Yes) or disabled (No). You can select how the

data is to be interpreted by choosing `string` (up to 40 characters), `int` (2-byte signed integer), `long` (4-byte signed integer), `float` (4-byte real), or `double` (8-byte real). You can also select whether to allow byte swapping.

This window remains open until you choose Cancel, enabling you to enter data several times and from several instruments without having to reselect Enter and Arbitrary Block Data each time. Refer to the HpibEnterab command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

Misc Menu

Send Bus Commands ...

Allows you to send arbitrary HP-IB bus commands to the bus. You can specify arbitrary talk or listen addresses (TA and LA); Unlisten (UNL); Untalk (UNT); My Talk Address (MTA), My Listen Address (MLA), or neither; and the universal commands Device Clear (DCL), Group Execute Trigger (GET), Go To Local (GTL), Local Lockout (LLO), Selected Device Clear (SDC), and Take Control (TCT). You can also use the User Specific area to send any other arbitrary command. Refer to the HpibSend command in chapter 4 for more information.

OK—Accepts your selections.

Cancel—Cancels your selections.

Abort Activity

Aborts bus activity. Refer to the HpibAbort command in chapter 4 for more information.

Clear Bus

Sends a device clear message to all devices. Refer to the HpibClear(select_code) command in chapter 4 for more information.

Clear Device

Sends a device clear message to the selected device. Refer to the HpibClear(device_address) command in chapter 4 for more information.

Reset Bus

Resets the bus to its start-up state. If it is system controller, it becomes active controller; if it is not system controller, the active controller status does not change. Refer to the HpibReset command in chapter 4 for more information. Reset Bus does not change the settings of any devices.

Serial Poll

Performs a serial poll on the currently active instrument and displays the response. Refer to the HpibSpoll command in chapter 4 for more information.

Trigger Bus

Triggers all devices on the interface select code. Refer to the HpibTrigger(select_code) command in chapter 4 for more information.

Trigger Device

Triggers the selected device. Refer to the HpibTrigger(device_address) command in chapter 4 for more information.

SRQ Status

Returns the status of the SRQ (Service Request) line in a message box.

Close Cards

Closes all HP-IB cards. This is useful when you are using the DLL and you wish to force all cards to be closed.

Window Menu

Tile

Arranges all of the non-icon instrument windows so that all are visible and none overlap.

Cascade

Arranges all of the instrument windows that are not currently icons. They will be overlapped in such a way that all title bars are visible.

Arrange Icons

Arranges all of the icons in a neat row at the bottom of the main window.

Close All

Closes all of the instrument windows.

DDE View Window

Displays a box of information that reflects the server's interpretation of what the program is doing. This information is useful for debugging DDE programs. Use DDE View Window while running a macro and stepping through it. See chapter 3 for more information on using DDE.

This menu is available only in the DDE Server application; it is not available in the Interactive Environment application.

1,2,3,4

Lists the current instrument windows, up to nine. If more than nine instruments are current, the message "More Windows" is displayed at the end of the list.

Help

Index

Displays the index for help topics about the HPIBINT program. Choose any of the underlined items to get further information.

Getting Started

Lists the steps to follow when using the HPIBINT program for the first time.

Commands

Provides descriptions of each of the menu commands.

Using DDE

Provides help on using the Dynamic Data Exchange capabilities.

Using Help

Provides basic information on using Windows Help.

About ...

Brings up a dialog box displaying version information about the Interactive HP-IB Environment. Choose OK when finished viewing.

About the Configuration File

When you save a file, it is saved as an HPIB configuration file. The format of the saved file is described here.

There will always be an [HPIB] section that lists each of the configured instruments. The first instrument will be INSTR1, the second will be INSTR2, and so on. The values these instruments are set to are the names of the instruments.

All other sections in the configuration file are the settings for each instrument. There will be one section for each instrument, and each will contain the

following entries: Address, Timeout, DDETopic, EOI, Virtual, EOLLength, EOLEnable, EOLPick, EOLString, MatchEnable, MatchChar, and Timing.

The left column in the following table shows a sample HPIB configuration file, with explanations in the right column (the explanations are not part of the file):

[HPIB]	
INSTR1=SCOPE	Device address of the first instrument
[SCOPE]	
Address=702	HP-IB address of this device
Timeout=30.00	Timeout value in seconds
DDETopic=SCOPE	
EOI=1	1 = EOI enabled, 0 = disabled
Virtual=0	0 = real device, 1 = virtual device
EOLLength=2	Length of the EOL string
EOLPick=1	0 = user-defined EOL, 1 = CRLF, 2 = CR, 3 = LF, 4 = semicolon
EOLEnable	1 = enabled, 0 = disabled
EOLString=13,10,0,0,0,0,0,0	ASCII value of the EOL string
MatchEnable=1	1 = enabled, 0 = disabled
MatchChar=10	ASCII value of the match character
Timing=0	1 = high-speed timing enabled, 0 = disabled

You can edit these settings with any text editor (such as NOTEPAD).

Using the Windows Recorder

As you become more familiar with the Interactive HP-IB Environment you may find yourself frequently repeating the same sequence of steps. In that case, you may want to use the Windows application named Recorder, which enables you to record a macro (a sequence of keystrokes and mouse actions) to automate the steps you frequently repeat. Refer to *Microsoft Windows User's Guide* for details on writing a macro with Recorder.

3

Using the HP-IB Dynamic Data Exchange Server

Introduction

Dynamic Data Exchange (DDE) is a standard method for Windows applications to communicate and exchange data with each other. Because an application must be written to take advantage of DDE, not all Windows applications necessarily support the use of DDE. However, many, such as Microsoft Excel and Microsoft Word for Windows, do support DDE.

The HP-IB DDE program provided with your HP 82335 HP-IB interface—along with other Windows applications that support DDE through their macro language—enables you to control instruments and acquire data. To communicate with the HP-IB DDE Server you need to write a *macro*, using the macro language provided by your application. In turn, the HP-IB DDE Server controls the instruments you have configured on your computer, and returns the requested data.

HP recommends that you familiarize yourself with the DDE capabilities of your application before you continue reading this chapter.

This chapter provides a few simple examples, and describes how to use an application's macro language to talk to the HP-IB DDE Server. By referring to the examples, you can develop your own macros. Although you may be using an application other than Microsoft Excel or Microsoft Word, which the examples use, you can get an idea of how to structure macros for instrument control through DDE.

Following the examples is further reference information about DDE, including tables showing valid syntax for your macros using the HP-IB DDE Server. At the end of the chapter is a reference section which describes each of the DDE items.

Installing the DDE Server

To begin using the HP-IB DDE Server, named HPIBDDE, you must first install it on your hard disk. Your master disks contain an install program named WINSTALL, which copies all necessary files for you.

To use WINSTALL:

1. Insert the master disk into your flexible disk drive.
2. Run Windows, since WINSTALL is a Windows application.
3. From the Program Manager, select File, then Run. In the dialog box, type `A:WINSTALL` and choose OK. If your master disk is not in drive A:, enter the appropriate drive letter.
4. Follow the instructions displayed on the screen to complete the installation process.

When WINSTALL is complete, you will have a new program group named HP-IB, which will contain all of the installed programs.

Using DDE Macros

HP's HP-IB DDE Server application does not provide a macro language itself; it is a *server* for clients. You use the macro notation provided by the client, such as Excel, and fill in the values that are valid for the HP-IB DDE Server.

There is not yet a standard macro language for Windows applications, so notation may vary from one application to another. Fortunately, however, a limited number of DDE commands exist, and the syntax for these commands is similar in each macro language.

If you are familiar with any application's macro language, you will find the DDE capabilities of this application reasonably easy to learn. You can use DDE commands in macros to start the HP-IB DDE Server, set up a device, send data to a device, and get data from a device.

The configuration file required to operate the DDE Server is described in chapter 2.

DDE Terminology

Several terms are defined here to help clarify the concepts described in this chapter.

Two applications participating in dynamic data exchange are having a DDE *conversation*. The application that initiates the conversation is the *client* application (such as Excel); the application responding to the client is the *server* application. (HP-IB DDE is always the server.)

A DDE conversation takes place between two windows, one for each of the participating applications. The window may be the main window of the application, a window associated with a specific document, or a hidden window whose only purpose is to process DDE messages. The HP-IB DDE Server is generally iconized because none of its input or output needs to be visible to the user outside of the client application.

An Analogy

As an introduction to macro commands and DDE identifiers, let's look at an analogy between a telephone call and DDE:

Phone Call	DDE Excel Macro Commands	Word BASIC Macro Commands
Dial a number	INITIATE	DDEInitiate
Hang up the phone	TERMINATE	DDETerminate
Ask for information	REQUEST	DDERequest
Provide information	POKE	DDEPoke
Give a command	EXECUTE	DDEExecute

Similarly, the DDE identifiers can be related to a phone call:

Phone Call	DDE Identifiers
Place you called	Application
Person there you want to talk to	Topic
Subject you want to talk about	Item

More information on the specifics of DDE identifiers and macro commands follows the examples below. Check the documentation for your application to identify its macro command syntax and use of identifiers.

Excel Example

The following DDE macro example illustrates the use of DDE to communicate with instruments using a Microsoft Excel spreadsheet.

Before running a macro such as this, you must use the Interactive Environment to define the configuration of your system, the default operating characteristics, and the name of each instrument to use. Refer to chapter 2 for information on generating the configuration file.

Note that all command names and identifiers are case-insensitive. You may use uppercase or lowercase letters.

	xlexample	:AUTOSCALE
	=ECHO(FALSE)	:WAVEFORM:FORMAT WORD
	=INITIATE("HPIBDDE","MAIN")	:DIGITIZE CHANNEL1
	=EXECUTE(A3,"[OPENCONFIG(setup1)]")	:WAVEFORM:DATA?
	=TERMINATE(A3)	
	=INITIATE("HPIBDDE","SCOPE")	
	=EXECUTE(A6,"[Abort]")	
	=EXECUTE(A6,"[Clear(device)]")	
	=POKE(A6,"OutputS(30)",B1)	
	=POKE(A6,"OutputS(30)",B2)	
	=POKE(A6,"OutputS(30)",B3)	
	=POKE(A6,"OutputS(30)",B4)	
	=SET.VALUE(D1:D4000,REQUEST(A6,"EnterAB(8000, int)"))	
	=TERMINATE(A6)	
	=SELECT(D1:D4000)	
	=NEW(2)	
	=GALLERY.LINE(2,FALSE)	
	=RETURN()	

1. Indicates the name of the macro.
2. Turns off screen updates in the macro to increase speed.
3. Opens a channel with the Main topic in HPIBDDE. This returns a number, or handle, that identifies the link with Main. This handle must be used in all subsequent Excel cells that talk to Main. In the phone call analogy, the INITIATE command is like dialing the phone and beginning a conversation with Main.
4. Opens the configuration file named `SETUP1.IBC`. This is the file generated using the Interactive HP-IB Environment described in chapter 2.
5. Closes the channel to Main. This is similar to hanging up the phone.
6. Opens a channel to the device named SCOPE, which is defined in the configuration file SETUP1.IBC, and returns a handle that Excel will use to talk through that channel. Note that any further instructions regarding this device need to refer to the handle in this cell, which contains the handle for SCOPE—usually as the instruction's first parameter.

   This is similar to dialing the number (INITIATE) to someplace (the HPIBDDE application) and asking to speak to someone (SCOPE).
7. Aborts all activity on the interface.
8. Clears the device named SCOPE (identified by cell A6).
9. Takes the string from cell B1 (:AUTOSCALE) and sends it to the scope. The string has a maximum length of 30 bytes.
10. Takes the string from cell B2 (:WAVEFORM:FORMAT WORD) and sends it to the scope. The string has a maximum length of 30 bytes.
11. Takes the string from cell B3 (:DIGITIZE CHANNEL1) and sends it to the scope. The string has a maximum length of 30 bytes.
12. Takes the string from cell B4(:WAVEFORM:DATA?) and sends it to the scope. The string has a maximum length of 30 bytes.
13. Brings all of the data into the spreadsheet. Cells D1 through D4000 are loaded with the 4000 integer values resulting from the DDE request to the DDE server. REQUEST tells HPIBDDE to do an "arbitrary block enter" of 8000 bytes from SCOPE, and return them to the SET.VALUE macro command as 4000 integer values separated by carriage returns. (Carriage

return is the default separator for data.) The macro command then stores the data in cells D1:D4000.

14. Closes the communication channel between Excel and the scope. Similar to hanging up the phone.
15. Selects the data that was stored in cells D1 through D4000.
16. Makes a new chart.
17. Changes the format of the chart to be a line chart.
18. Ends the macro and returns control to whatever called the macro.

Microsoft Word BASIC Example

The following example illustrates a conversation between HPIBDDE and three instruments, using a Word BASIC program.

```
Sub MAIN
Dim dlg As Dialog UserDialog
'
'Initiate a link with 'Main' to open up the configuration file,
'which contains 3 instruments:  "SCOPE", "METER", and "FUNCGEN".
'
MainChannel = DDEInitiate("HPIBDDE", "Main")
DDEExecute MainChannel, "[OpenConfig(WordSamp)]"
DDETerminate MainChannel
'
'Now that three devices exist in HPIBDDE, initiate links with them
'
ScopeChannel = DDEInitiate("HPIBDDE", "SCOPE")
Meter = DDEInitiate("HPIBDDE", "METER")
FuncGenChannel = DDEInitiate("HPIBDDE", "FUNCGEN")
'
'Do some initial setup
'
DDEExecute ScopeChannel, "Reset"
DDEExecute ScopeChannel, "Abort"
```

```
DDEExecute ScopeChannel, "Clear(bus)"
'
'Configure the Function Generator
'
DDEPoke FuncGenChannel, "OutputS(30)", "FU3AM2V0"
'
'Configure the Scope
'
DDEPoke ScopeChannel, "OutputS(30)", ":AUTOSCALE"
DDEPoke ScopeChannel, "OutputS(30)", ":WAVEFORM:FORMAT WORD"
DDEPoke ScopeChannel, "OutputS(30)", ":DIGITIZE CHANNEL1"
DDEPoke ScopeChannel, "OutputS(30)", ":WAVEFORM:DATA?"
'
'Get data from the scope
'
ScopeReturn$ = DDERequest$(ScopeChannel, "EnterAB(8000, int)")
B$ = Left$(ScopeReturn$, 100)
'
'Print some data from the scope
'
Begin Dialog UserDialog 400, 70
  OKButton 170, 50, 70, 14
  GroupBox 3, 20, 394, 22, ""
  Text 6, 27, 390, 12, B$
End Dialog
Dialog dlg
'
'Configure the VoltMeter
'
DDEPoke Meter,"OutputS(30)", "INBUF ON;DCV 30,.1;NPLC 0;AZERO OFF"
DDEPoke Meter,"OutputS(30)", "DELAY 0;DISP OFF;LOCK ON;EMASK 2047"
DDEPoke Meter,"OutputS(30)", "RQS 105;TARM HOLD;MSIZE 1000,150"
DDEPoke Meter,"OutputS(30)", "MEM FIFO;MFORMAT SINT;OFORMAT ASCII"
DDEPoke Meter,"OutputS(30)", "TRIG AUTO;NRDGS 50,AUTO;SUB 1"
DDEPoke Meter,"OutputS(30)", "TARM SGL,10;BEEP ONCE;SUBEND"
DDEPoke Meter,"OutputS(30)", "CALL 1"
'
'Wait for the VoltMeter to set SRQ
```

```
'
SrqStatus$ = DDERequest$(Meter, "Status(SRQ)")
While Asc(SrqStatus$) <> Asc("1")
  SrqStatus$ = DDERequest$(Meter, "Status(SRQ)")
Wend
'
'Serial Poll the VoltMeter
'
SrqResponse$ = DDERequest$(Meter, "SerialPoll")
'
'Read the data from the VoltMeter and print some of it
'
MeterReturn$ = DDERequest$(Meter, "EnterS(1000)")
B$ = Left$(MeterReturn$, 100)
Begin Dialog UserDialog 400, 70
  OKButton 170, 50, 70, 14
  GroupBox 3, 24, 394, 22, ""
  Text 6, 32, 390, 12, "Data = " + B$
  GroupBox 3, 3, 394, 22, ""
  Text 6, 11, 390, 12,"Serial Poll Response = " + SrqResponse$
End Dialog
Dialog dlg
'
'Terminate the links with all devices
'
DDETerminate ScopeChannel
DDETerminate FuncGenChannel
DDETerminate Meter
'
End Sub
```

Macro Commands

To successfully use DDE, you must become familiar with the macro language of the application you are using. In every macro language, there are only five commands to implement DDE support:

INITIATE	Initiate a DDE conversation
TERMINATE	Terminate a DDE conversation
POKE	Send data to another application
REQUEST	Acquire data from another application
EXECUTE	Execute commands in the other application

The parameters to these commands tell Windows the application, topic, and item that you want to use.

DDE Identifiers

Each DDE conversation has three identifiers to tell Windows which application you are talking to and what you want to talk about. The identifiers are:

- Application
- Topic
- Item

Application	The name of the program you want to talk to (always HPIBDDE)
Topic	What you want to talk about
Item	The information within a topic

HP's DDE server application is always HPIBDDE. There is initially only one topic, *Main*, which enables you to add new topics. These new topics are the actual devices on the HP-IB bus. The item parameter indicates what you want to do to the device, such as output a string (OutputS) or clear the bus (Clear(bus)).

The following tables list all possible applications, topics, and items that HP's DDE Server recognizes. The access method refers to the macro commands used to access the capabilities of the server. Macro commands are described in the next section.

Applications for HPIBDDE

Application	Description
HPIBDDE	HP's HP-IB DDE Server

Topics for HPIBDDE

Topic	Description
System	Gives information about the state of the DDE Server
Main	Affects the DDE Server
Device names (e.g., Scope)	Affects the HP-IB bus

Items for the System Topic

Item	Description	Access Method
SysItems	Returns a list of all items for the System topic	REQUEST
Topics	Returns a list of all topics defined for the DDE server	REQUEST
ReturnMessage	Returns the last error message from the DDE server	REQUEST
Status	Returns a status message from the DDE server	REQUEST
Formats	Returns a list of all formats supported by the DDE server	REQUEST

Items for the Main Topic

Item	Syntax	Access Method
OpenConfig	[OpenConfig(*d:\path\filename*)]	EXECUTE

Items for Device Topics

Item	Syntax	Access Method
Abort	[Abort]	EXECUTE
Clear	[Clear(bus|device)]	EXECUTE
EnterAB	EnterAB(*length,type,separator*)	REQUEST
EnterS	EnterS(*length*)	REQUEST
EOI	[EOI(true|false)]	EXECUTE
EOL	EOL	POKE
Llockout	[Llockout]	EXECUTE
Local	[Local(bus|device)]	EXECUTE
Match	Match(true|false)	POKE
OutputS	OutputS(*length*)	POKE
Remote	[Remote(bus|device)]	EXECUTE
Reset	[Reset]	EXECUTE
SerialPoll	SerialPoll	REQUEST
Status	Status(SRQ)	REQUEST
Timeout	Timeout	POKE
Trigger	[Trigger(bus|device)]	EXECUTE

Each of the items for the device and main topics is described in more detail in the reference section later in this chapter.

Debugging DDE Macros

To aid in the debugging of your macros, a DDE View window is available in the HP-IB DDE Server. You can open this window by choosing Window and DDE View from the menus in the server. This window displays information about what your macro is sending to the server. It will display only valid commands that are sent, so if nothing shows up in this window, something is wrong with your macro.

If nothing appears in the window, check to make sure that you have initiated a link with the device. If you are using Excel, make sure that the first parameter to all of your DDE commands is the name of the cell containing the =INITIATE command. Also, make sure that all of the parameters sent to the server are valid.

If the INITIATE command fails, make sure that HPIBDDE is running, and that it contains the devices you are trying to control. For example, if you are trying to initiate a link to the device named SCOPE, make sure that SCOPE is available in the HP-IB DDE Server.

You can bring devices into HPIBDDE by using the OpenConfig() command. You can then verify the availability of a device by opening up the DDE Server icon, and looking at the device windows.

Error Handling

After each command is sent to the HP-IB DDE Server, an error status is available to your application. Two items are updated: `[Error]` and `[ReturnMessage]`. The `[Error]` item will contain an error number, with zero indicating that no error occurred. The `[ReturnMessage]` item will contain a string that corresponds to the `[Error]` error number. (For example, if `[Error]` = 0, `[ReturnMessage]` will be `"No Error"`.)

The following Microsoft Excel example shows how to retrieve this information from the HP-IB DDE Server:

```
=INITIATE("HPIBDDE","Scope")
=EXECUTE(A2, "[ABORT]")
=REQUEST(A2,"[Error]")
=REQUEST(A2,"[ReturnMsg]")
=IF(A4=0,TRUE,ALERT(A5,3))
```


Always include error checking after each command is sent to the HP-IB DDE Server.

Reference

This section presents a detailed DDE Library syntax reference for Microsoft Excel and Microsoft Word BASIC.

Abort

This command aborts all activity on the interface.

Syntax

```
[Abort]
```

Examples

Excel: `=EXECUTE(A2,"[abort]")`

Word BASIC: `DDEExecute ScopeChannel,"[Abort]"`

Clear

This command returns a device to a known, device-dependent state. It can be addressed to the interface or to a specific device.

Syntax

`[Clear (`*bus_or_device*`)]`

bus_or_device specifies whether to clear the bus or device. Can take one of two values: bus or device.

Examples

Excel: `=EXECUTE(A2,"[clear(device)]")`

Word BASIC: `DDEExecute ScopeChannel, "[Clear(bus)]"`

EnterAB

This command enters arbitrary-block program data (numeric or string data with IEEE-488.2 coding) from a device or the interface. This command also can convert binary data into a string representation of the data, making it usable by most Windows applications. Reading continues until one of these events occurs:

- The maximum number of bytes specified is received.
- A linefeed is encountered with the EOI line sensed true, if the coding indicates indefinite length.
- The number of bytes indicated by the coding is received, if the coding indicates definite length.

Syntax

`EnterAB (`*length*`,`*type*`,`*separator*`)`

length specifies the maximum number of bytes to enter.

type specifies the type of data to enter: `string`, `int` (2-byte signed integer), `long` (4-byte signed integer), `float` (4-byte real), or `double` (8-byte real).

separator defines the character that separates the data items in the string representation of the binary data. Many Windows applications cannot use binary data directly; therefore, the EnterAB command has the ability to convert the binary data into a string representation of the data. Valid separators are:

CR	carriage return
LF	linefeed
CRLF	carriage return/linefeed
Tab	tab character
comma	comma
space	space
bin	don't convert to string; keep as binary data

The default separator is CR if the separator parameter is omitted.

Examples

Excel: `=REQUEST(A2,"EnterAB(55,string,tab)")`

Word BASIC: `A$ = DDERequest$(ScopeChannel, "EnterAB(8000, int,LF)")`

EnterS

This command enters a character string from a device or the interface. Reading continues until one of these events occurs:

- The EOI line is sensed true, if it is enabled.
- The termination character set by Match is received (linefeed is the default).
- The maximum number of characters specified is received.

Syntax

`EnterS (`*length*`)`

length specifies the maximum length of the string.

Examples

Excel: `=REQUEST(C14,"EnterS(70)")`

Word BASIC: `C$ = DDERequest$(ScopeChannel, "EnterS(8000)")`

EOI

This command enables or disables the End Or Identify (EOI) mode of the interface. It is used to:

- Enable or disable a write operation to set the EOI line on the last byte of the write.
- Enable or disable a read operation to terminate upon sensing the EOI line true.

The default is EOI enabled.

Syntax

`[EOI(`*status*`)]`

status specifies whether to enable or disable EOI. Can take one of two values: true enables EOI; false disables EOI.

Examples

Excel: `=EXECUTE(A2,"[EOI(true)]")`

Word BASIC: `DDEExecute ScopeChannel, "[EOI(false)]"`

EOL

This command defines the End Of Line (EOL) string that is to be sent following every OutputS command. The default is carriage return and linefeed. The maximum EOL length is 8 characters. You can disable EOL by setting EOL to a string containing only a comma.

Syntax

`EOL`

Examples

Excel:

```
=SET.VALUE(B5,"10,13")    set the EOL string to CR/LF
=POKE(A2,"EOL",B5)        B5 contains the EOL string
```

Word BASIC:

```
Rem Disable EOL
DDEPoke FuncChan, "EOL", ","
```

Error

The [Error] item returns an error number corresponding to the last error that occurred in the HP-IB DDE Server. Check this item after each call to the DDE Server to ensure that no errors have occurred; a value of zero indicates that there are no errors.

Syntax

```
[Error]
```

Examples

Excel: `=REQUEST(A2,"[Error]"`

Word BASIC: `A$=DDERequest$(ScopeChannel,"[Error]")`

Llockout

This command sends a Local Lockout (LLO) to disable a device front panel. It is received by all devices on the interface, whether or not they are addressed to listen.

Syntax

```
[Llockout]
```

Examples

Excel: `=EXECUTE(A2,"[Llockout]")`

Word BASIC: `DDEExecute MeterChannel, "[Llockout]"`

Local

This command executes a Go To Local (GTL) or clears the REN line to enable a device front panel.

Syntax

`[Local (`*bus_or_device*`)]`

bus_or_device specifies whether to clear the REN line on the bus the device is connected to or to send a GTL command to the device. Can have one of two values: bus, which clears the REN line; or device, which sends a GTL command to the device.

Examples

Excel: `=EXECUTE(A2,"[Local(bus)]")`

Word BASIC: `DDEExecute ScopeChannel, "[Local(device)]"`

Match

This command defines the character used by EnterS for termination. The default character is linefeed.

Syntax

`Match (`*status*`)`

status specifies whether to enable or disable Match. *Status* can take one of two values: `true` enables Match, `false` disables Match.

Examples

Excel:

```
=SET.VALUE(B5,"10")
=POKE(A2,"Match(true)",B5)      B5 contains the Match character
```

Word BASIC:

```
Rem Disable Match character
DDEPoke ScopeChannel, "Match(false)", "10"
```

OpenConfig

This command opens a configuration file that was created using the Interactive HP-IB Environment.

Syntax

`[OpenConfig(`*drive:\path\filename.ext*`)]`

drive:	the letter of the drive that contains the configuration file. (The default is the current drive.)
path	the full path to the configuration files. (The default is the current directory.)
filename	the name of the configuration file.
.ext	the extension on the configuration file. (The default is .IBC.)

Examples

Excel: `=EXECUTE(A2,"[OpenConfig(C:\HPIB\myconfig)]"`

Word BASIC: `DDEExecute MainChannel, "[OpenConfig(myconfig)]"`

OutputS

This command outputs a string to a specified device or to the interface. After the string is sent, the EOL string is sent and the EOI line is set (if enabled).

Syntax

`OutputS (`*length*`)`

length specifies the maximum length of the string.

Examples

Excel: `=POKE(C14,"OutputS(30)",C9)` (C9 contains the string to be sent)

Word BASIC: `DDEPoke FuncChan, "OutputS(30)", ":AUTOSCALE"`

Remote

This command places a device in Remote mode to disable the device front panel. It can be addressed to the interface or to a specific device.

Syntax

`[Remote (`*bus_or_device*`)]`

bus_or_device specifies whether to set the REN on the bus or on the device. Can have one of two values: bus sets the REN line on the bus the device is attached to; device sets the REN line on the device, and addresses the device.

Examples

Excel: `=EXECUTE(A2,"[Remote(bus)]")`

Word BASIC: `DDEExecute ScopeChannel, "[Remote(device)]"`

Reset

This command sets the interface to its start-up state, in which it is not listening and not talking.

In addition, if the interface was system controller, then it will also become active controller.

Syntax

```
[Reset]
```

Examples

Excel: `=EXECUTE(C14,"[Reset]")`

Word BASIC: `DDEExecute ScopeChannel, "[Reset]"`

ReturnMsg

The [ReturnMsg] item returns a string that corresponds to the last error that occurred in the HP-IB DDE Server.

Syntax

```
[ReturnMsg]
```

Examples

Excel: `=REQUEST(A2,"[ReturnMsg]")`

Word BASIC: `A$=DDERequest$(Scopechannel, "[ReturnMsg]")`

SerialPoll

This command performs a serial poll of a specified device and returns the device's serial poll response byte.

Syntax

```
SerialPoll
```

Examples

Excel: `=REQUEST(A2,"SerialPoll")`

Word BASIC: `A$ = DDERequest$(ScopeChannel, "SerialPoll")`

Status

This command determines the current interface status regarding a particular condition. It sets a variable representing that status.

Syntax

`Status(`*condition*`)`

condition specifies what condition to get the status of. Currently, only one value can be used: SRQ.

Examples

Excel: `=REQUEST(C14,"Status(SRQ)")`

Word BASIC: `A$ = DDERequest$(ScopeChannel, "Status(SRQ)")`

Timeout

This command sets an interface timeout value in seconds for I/O operations that do not complete (for example, the printer runs out of paper).

Syntax

```
Timeout
```

Examples

Excel: `=POKE(A5,"Timeout",C7)` (C7 contains the timeout value)

Word BASIC: `DDEPoke FuncGenChannel, "Timeout", "30"`

Note Setting the timeout value to zero disables timeouts.

Trigger

This command triggers one or more devices.

Syntax

`[Trigger (`*bus_or_device*`)]`

bus_or_device specifies whether to trigger the bus the device is connected to or trigger the device. Can take one of two values: bus triggers the bus; device triggers the device.

Examples

Excel: `=EXECUTE(A5,"[Trigger(bus)]")`

Word BASIC: `DDEExecute ScopeChannel, "[Trigger(device)]"`

4

Using the Windows Dynamic Link Library

Introduction

This chapter describes the Windows Dynamic Link Library (DLL) for HP-IB, which enables programmers to access HP-IB instruments using general-purpose Windows software development tools. This chapter assumes that you have purchased a program development environment for Microsoft Windows and that you are familiar with writing Windows applications.

Installing the Dynamic Link Library

To begin using the Windows Dynamic Link Library, you must first install it on your hard disk. Your master disks contain an install program named WINSTALL, which copies all necessary files for you.

To use WINSTALL:

1. Insert the master disk into your flexible disk drive.
2. Run Windows, since WINSTALL is a Windows application.
3. From the Program Manager, select File, then Run. In the dialog box, type `A:WINSTALL` and choose OK. If your master disk is not in drive A:, enter the appropriate drive letter.
4. Follow the instructions displayed on the screen to complete the installation process.

Differences Between the DLL and the DOS Command Library

The Windows DLL is similar to the DOS Command Library, with only three differences in functionality:

- The Windows DLL does not support DMA. The DOS command libraries have the IODMA function.
- The Windows DLL has two additional commands, HpibOpen and HpibClose, which allow the Windows DLL to function in the multi-tasking environment of Windows 3.0.
- The Windows DLL has an additional function, HpibSetWaitHook, which defines a function to be called when the user application is waiting for devices to finish handshaking the HP-IB bus. It enables Windows to function normally despite a timeout condition in progress on the HP-IB bus.

In addition and more specifically, using the Windows DLL is similar to using the C library, CLHPIB.LIB, with three differences:

- All Windows DLL functions begin with *Hpib* instead of *IO*, with only the first letter after Hpib capitalized. For example, the IORESET command in the DOS library is named HpibReset in the Windows DLL.
- All Windows DLL functions have an added parameter. The first parameter in every function is the handle returned by HpibOpen, with all other parameters the same as their IO counterparts. For example, where the DOS library would call `HpibReset (7L)`, the DLL would call `HpibReset (hHpib, 7L)`.
- All Windows DLL functions have four error return values in addition to the error values returned by the DOS library commands. The new error return values are:

EOPEN	Indicates that an error occurred during a call to HpibOpen.
ENOOPEN	Indicates that the card has not been opened.
ECLOSE	Indicates that an error occurred during a call to HpibClose.
EHANDLE	Indicates that an invalid handle was received.

Writing Applications

The HpibOpen command, described in detail in the next section, must be called before any other library routines are used. It returns a handle to the HP-IB interface, which must be used as the first parameter to all other library routines. HpibOpen locks out all other applications that are using the DLL from using that HP-IB interface until the application closes the interface with the HpibClose command. It can return error values of NOERR (no error), ESEL (invalid select code), and EOPEN (card already open).

The HpibClose command, also described in the next section, has one parameter—the handle that HpibOpen returned. It unlocks an interface, enabling other applications to access the interface. It can return error values of NOERR (no error) and ECLOSE (cannot close the card).

Note

If you want exclusive use of a card, keep it open. Make sure, however, that you eventually close the card, to let other applications have a chance to use it. If you do not want exclusive use of the card, encapsulate all function calls with HpibOpen and HpibClose calls.

Note that it is possible to have a single application or multiple applications controlling a single or multiple interfaces.

Your application development environment should include a method to call functions in a DLL. You should become familiar with your development environment to find out how to call DLL functions. For example, in C, you need to use #include to include the file hpib.h, and link with the file hpib.lib to create your applications. In Visual BASIC, however, you need only to include a new global.BAS file.

Refer to the *Supported Languages* sheet for a complete list of Windows application development environments that are supported with include files. If you are using an environment that is not supported, you can create your own include file by following the instructions in your environment's manuals.

Visual BASIC Programming

To write a Visual BASIC program, you first design the interface, which involves adding controls to a form. Typical controls are pushbuttons, labels, textboxes, and pictureboxes. When the program runs, the user, another application, or Windows itself can generate events associated with the various controls. For example, if a user clicks on one of the pushbuttons, a click event occurs for that particular button. You write BASIC code called an event procedure for that particular pushbutton; the event procedure executes each time the click event occurs.

To use the HP-IB DLL with Visual BASIC, create your application in the usual way and perform these steps within Visual BASIC before running the application:

1. Add the file `HPIBGLBL.TXT` to the `GLOBAL.BAS` module of your program.

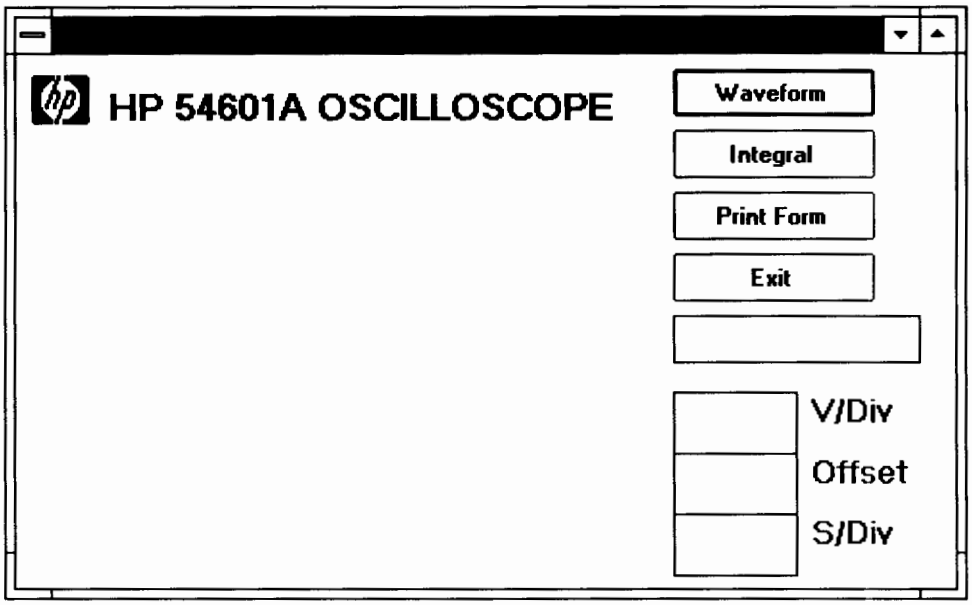
   To do this, double-click on `GLOBAL.BAS` in the project window, then select `Code, Load Text ...` , select `HPIBGLBL.TXT`, and click on the `Merge` button.

2. Include the error handling routine, `HpibErrStr$`, in your project.

   To do this, select `File, Add File ...` , then select the file `HPIBERR.TXT`. You now have access to all the functions in the DLL, as well as the error handling function `HpibErrStr$`.

Visual BASIC Example

The following example demonstrates the use of the HP-IB DLL as well as the error handling routine.



Visual BASIC Example - The Form

Visual BASIC Example - Properties and Events

Control	Property Settings	Significant Events
Form	FormName=Form1 Caption=Hewlett-Packard AutoRedraw=True	
Command Button	CtlName=Command1 Caption=Waveform	Click [see Sub Command1_Click ()]
Command Button	CtlName=Command4 Caption=Integral	Click [see Sub Command4_Click ()]
Command Button	CtlName=Command5 Caption=Print Form	Click [see Sub Command5_Click ()]
Command Button	CtlName=Command2 Caption=Exit	Click [see Sub Command2_Click ()]

Visual BASIC Example - Properties and Events (continued)

Control	Property Settings	Significant Events
Picture Box	CtlName=Picture1 Picture=hp.bmp	
Label	CtlName=Label4 Caption=HP 54601A OSCILLOSCOPE BackColor=&H00FFFF00& ForeColor=&H00FF0000& FontSize=13.5 Alignment=Center	
Label	CtlName=Label1 Caption=V/Div FontSize=12 BackColor=&H00FFFF00& ForeColor=&H00FF0000&	
Label	CtlName=Label2 Caption=Offset FontSize=12 BackColor=&H00FFFF00& ForeColor=&H00FF0000&	
Label	CtlName=Label3 Caption=S/Div FontSize=12 BackColor=&H00FFFF00& ForeColor=&H00FF0000&	
Text Box	CtlName=Text1 Text=	
Text Box	CtlName=Text2 Text=	
Text Box	CtlName=Text3 Text=	
Text Box	CtlName=status Text=	

Visual BASIC Example - Event Procedures

```
Sub Command1_Click ()

Rem set up some variables
    isc& = 7
    device& = isc& * 100 + 7
    swap% = 2
    max% = 4000 * swap%
    act% = 0

    max1% = 50
    TimeVal# = 3#

Rem Make sure text boxes are clear

    text1.text = ""
    text2.text = ""
    text3.text = ""

Rem Set up the scope

    errnum% = HpibOpen(isc&, hHpib%)
    status.text = "Open - " + HpibErrStr$(errnum%)

    errnum% = HpibReset(hHpib%, isc&)
    status.text = "Reset - " + HpibErrStr$(errnum%)

    errnum% = HpibTimeout(hHpib%, isc&, TimeVal#)
    status.text = "TimeOut - " + HpibErrStr$(errnum%)

    wave$ = ":AUTOSCALE"
    length% = Len(wave$)
    errnum% = HpibOutPutS(hHpib%, device&, wave$, length%)
    status.text = "OutPutS - " + HpibErrStr$(errnum%)

    wave$ = ":WAVEFORM:FORMAT WORD"
    length% = Len(wave$)
    errnum% = HpibOutPutS(hHpib%, device&, wave$, length%)
    status.text = "OutPutS - " + HpibErrStr$(errnum%)
```

```
    wave$ = ":DIGITIZE CHANNEL1"
    length% = Len(wave$)
    errnum% = HpibOutPutS(hHpib%, device&, wave$, length%)
    status.text = "OutPutS - " + HpibErrStr$(errnum%)

Rem read the preamble
    wave$ = ":WAVEFORM:PREAMBLE?"
    length% = Len(wave$)
    errnum% = HpibOutPutS(hHpib%, device&, wave$, length%)
    status.text = "OutPutS - " + HpibErrStr$(errnum%)

    errnum% = HpibEnterA(hHpib%, device&, preamble!(0), max1%)
    status.text = "EnterA- " + HpibErrStr$(errnum%)

Rem read the data
    wave$ = ":WAVEFORM:DATA?"
    length% = Len(wave$)
    errnum% = HpibOutPutS(hHpib%, device&, wave$, length%)
    status.text = "OutPutS - " + HpibErrStr$(errnum%)

    errnum% = HpibEnterAB(hHpib%, device&, waveform%(0), max%, swap%)
    status.text = "EnterAB - " + HpibErrStr$(errnum%)

Rem read the last character in manually.  If this character is
Rem    a linefeed, then the scope is done sending data.  If this
Rem    character is a semicolon or comma, then the scope wants
Rem    to send more data...

    lastchar$ = "  "
    max% = 1
    errnum% = HpibEnterS(hHpib%, device&, lastchar$, max%)
    status.text = "EnterS - " + HpibErrStr$(errnum%)

    If Left$(lastchar$, 1) <> Chr$(10) Then
        status.text = "more data"
    End If

    errnum% = HpibClose(hHpib%)
    status.text = "Close - " + HpibErrStr$(errnum%)

Rem Deal with the preamble
```

```
    VpD = (32 * preamble!(7))
    Off = (128 - preamble!(9)) * preamble!(7) + preamble!(8)
    SpD = preamble!(2) * preamble!(4) / 10
    text1.text = Str$(VpD)
    text2.text = Str$(Off)
    text3.text = Str$(SpD)

    Cls

Rem Set up the screen coordinate system
    ScaleLeft = 0
    ScaleTop = 330
    ScaleWidth = 6000
    ScaleHeight = -330

Rem Draw the Grid

Rem Main Border

    Line (100, 10)-(4100, 10), RGB(0, 128, 0)
    Line -(4100, 266), RGB(0, 128, 0)
    Line -(100, 266), RGB(0, 128, 0)
    Line -(100, 10), RGB(0, 128, 0)

Rem Y-axis grid

    Line (500, 10)-(500, 266), RGB(0, 128, 0)
    Line (900, 10)-(900, 266), RGB(0, 128, 0)
    Line (1300, 10)-(1300, 266), RGB(0, 128, 0)
    Line (1700, 10)-(1700, 266), RGB(0, 128, 0)
    Line (2100, 10)-(2100, 266), RGB(255, 0, 0)
    Line (2500, 10)-(2500, 266), RGB(0, 128, 0)
    Line (2900, 10)-(2900, 266), RGB(0, 128, 0)
    Line (3300, 10)-(3300, 266), RGB(0, 128, 0)
    Line (3700, 10)-(3700, 266), RGB(0, 128, 0)

Rem X-axis grid

    Line (100, 42)-(4100, 42), RGB(0, 128, 0)
    Line (100, 74)-(4100, 74), RGB(0, 128, 0)
```

```
    Line (100, 106)-(4100, 106), RGB(0, 128, 0)
    Line (100, 138)-(4100, 138), RGB(255, 0, 0)
    Line (100, 170)-(4100, 170), RGB(0, 128, 0)
    Line (100, 202)-(4100, 202), RGB(0, 128, 0)
    Line (100, 234)-(4100, 234), RGB(0, 128, 0)

Rem Draw the waveform

    CurrentX = 100
    CurrentY = waveform%(0) + 10
    For Xaxis% = 1 To 3999
        Line -(Xaxis% + 100, waveform%(Xaxis%) + 10)
    Next Xaxis%

End Sub

'=========================================================================

Sub Command2_Click ()
    End
End Sub

'=========================================================================

Sub Command4_Click ()

' first, make sure that there is a waveform in memory...
If preamble!(2) = 0 Then
    MsgBox ("Must retrieve waveform first...")
    Exit Sub
End If

' calculate the integral
ReDim math!(preamble!(2))
math!(0) = 0
For i% = 1 To preamble!(2) - 1
    math!(i%) = math!(i% - 1) + (waveform%(i%) - preamble!(9)) *
                                    preamble!(7) + preamble!(8)
Next i%

'calculate the min and max of the integral
```

```
max! = math!(0)
min! = math!(0)
For i% = 1 To preamble!(2) - 1
    If math!(i%) > max! Then max! = math!(i%)
    If math!(i%) < min! Then min! = math!(i%)
Next i%

' plot the integral
ScaleVal = 256 / (max! - min!)
AddVal! = (-min! * ScaleVal) + 10
For i% = 0 To preamble!(2) - 1
    PSet (i% + 100, math!(i%) * ScaleVal + AddVal!), RGB(0, 0, 255)
Next i%
End Sub

'====================================================================

Sub Command5_Click ()
    form1.PrintForm
End Sub
```

Turbo Pascal Programming

With Turbo Pascal for Windows, you need to add one statement to your program to access the HP-IB DLL functions. Immediately following the **uses** statement, add the following program line:

```
{$I TPWDECL.EX}
```

You will now have access to all of the functions in the DLL and the error handling routine `Hpiberrstr`.

Turbo Pascal Example

The following Turbo Pascal for Windows program demonstrates the use of the DLL as well as the error handling routine.

```
{
 This example uses the HP 34401A Multimeter as the primary device.
 We will also use the HP 3325A Function Generator as a source for
 the multimeter.

 This example sets up the meter to take 128 readings, reads the data
 into an array, then prints some simple statistics about the data.
 This program is also checking other devices that are on the bus to
 see if they need service.  The SRQ line along with parallel and
 serial polling is used to make these checks.  The program will
 continue until the user presses a key on the PC keyboard.
}

program HpibDemo(input, output);

uses WinCrt;

{$I TPWDECL.EX}

const
    NUM_READINGS = 128 ;

type   (* type declarations follow *)
    strtype = string[255];
    arrtype = array [1..NUM_READINGS] of single;

var
    isc                         :longint;
    dvm                         :longint;
    source                      :longint;
    device_addr_1       :longint;
    device_addr_2       :longint;
    cmd                         :strtype;
    len                         :integer;
    response            :integer;
    readings            :arrtype;
```

```
    hHpib                       :integer;

procedure cleanup; forward;

procedure error_handle(error : integer; routine: strtype);
var retval : INTEGER ;
begin
    if error <> NOERR then begin

        (* we have an error, so let's abort all activity on the HPIB bus
         *)
        retval := HpibAbort(hHpib, isc) ;

        cleanup;
        writeln('Error in call to ', routine, error:3, errstr(error));
        halt(1);
    end;
end;

procedure cleanup;
var
    retval : INTEGER ;
begin

    (* clear the dvm so we can send the commands to reset it
     *)
    retval := HpibClear(hHpib, dvm) ;

    (* reset the dvm
     *)
    cmd := ':DISP:STATE ON; *RST' ;
    len := length(cmd);
    retval := HpibOutputs(hHpib, dvm, cmd[1], len) ;

    (* unconfigure the parallel poll
     *)
    retval := HpibPpollu(hHpib, isc) ;

    (* We are done with HP-IB...
     *)
     retval := HpibClose (hHpib);
```

```
end;

procedure get_data;
var
    i       : integer;
    ymin,
    ymax    : real;
    sum     : real;
begin

    (*  Ask the DVM to send us the data
     *)

    str (NUM_READINGS:0, cmd) ;
    cmd := ':SAMPLE:COUNT ' + cmd + '; :READ?' ;
    len := length(cmd);
    error_handle(HpibOutputs(hHpib, dvm, cmd[1], len), 'HpibOUTPUTS #2');

    (*  Read the data
     *)

    len := NUM_READINGS ;
    error_handle(HpibEntera(hHpib, dvm, readings[1], len), 'HpibENTERA #1');

    (* find the minimum and maximum values in the data
     *)
    ymin := readings[1];
    ymax := readings[1];
    sum  := 0;

    for i:=1 to len do begin
        if (readings[i] < ymin) then ymin := readings[i];
                if (readings[i] > ymax) then ymax := readings[i];
        sum := sum + readings[i];
    end;

    (* print some information about the data
     *)
    CursorTo (0,0);
    writeln('MAX = ', ymax);
    writeln ('MIN = ', ymin);
```

```
    writeln ('MEAN = ', sum/len);

end;

procedure poll_device (dev_addr: longint) ;
var
    response : integer;
begin

    (* do a serial poll of the device configured to use parallel
     *  poll line 0
     *)
    error_handle(HpibSpoll(hHpib, dev_addr, response), 'HpibSPOLL #3');

    (* should check RESPONSE here to see if any action needs to be taken.
     * the values that RESPONSE can take are device dependent.
     *)
end;


procedure check_srq;
var
    response : integer;
begin
    (* conduct a parallel poll
     * note that the source doesn't respond to parallel poll's,
     *  so we need to poll that device separately.
     *)
    error_handle(HpibPpoll(hHpib, isc, response), 'HpibPPOLL #1');
    if ((response and 1) <> 0) then
        poll_device (device_addr_1);

    if ((response and 2) <> 0) then
        poll_device (device_addr_2);

    (* check all devices that were configured to respond to
     *  parallel poll
     *)

    (* check any other devices on the bus here that weren't
     *  configured to respond to parallel poll by performing
```

```
     *  a serial poll on each one.
     *)
    error_handle(HpibSpoll(hHpib, source, response), 'HpibSPOLL #2');

    (* see if we've cleared the srq yet
     *)
    error_handle(HpibStatus(hHpib, isc, 1, response), 'HpibSTATUS #3');
    if (response = 1) then begin
        cleanup;
        writeln('SRQ locked high');
        halt(1);
    end;
end;

procedure setup;
begin
    (* program the function generator
     *)
    cmd := 'RF1 FR10HZ FU1 ST1KH SP10KH MF1KH AM1VR TI5SE' ;
    len := length(cmd);
    error_handle(HpibOutputs(hHpib, source, cmd[1], len), 'HpibOUTPUTS #1');

    (* program the dvm
     *)
    cmd := ':CONF:VOLT:DC 30,.1;';
    cmd := cmd + ':ZERO:AUTO OFF;';
    cmd := cmd + ':TRIG:DELAY MIN;';
    cmd := cmd + ':DISP:STATE OFF;';
    cmd := cmd + '';
    len := length(cmd);
    error_handle(HpibOutputs(hHpib, dvm, cmd[1], len), 'HpibOUTPUTS #2');
end;

procedure initialize;
var
    ErrCode     : integer;
    TempChar    : pchar ;
begin
    isc := 7;
    dvm := isc * 100 + 22;
    source := isc * 100 + 12;
```

```
    device_addr_1 := isc * 100 + 20;
    device_addr_2 := isc * 100 + 7;


    InitWinCrt;
    (* initialize the hpib interface and scope
     *)
    ErrCode := HpibOpen(isc, hHpib);
    if (ErrCode <> NOERR) then
    begin
        writeln ('Could not open the HP-IB card...');
        halt(1);
    end;
    error_handle(HpibReset(hHpib, isc), 'HpibRESET');
    error_handle(HpibTimeout(hHpib, isc, 2.0), 'HpibTIMEOUT');
    error_handle(HpibClear(hHpib, source), 'HpibCLEAR #1');
    error_handle(HpibClear(hHpib, dvm), 'HpibCLEAR #2');
    error_handle(HpibFastout(hHpib, isc, 1), 'HpibFASTOUT');
    TempChar := chr(10);
    error_handle(HpibEol(hHpib, isc, TempChar, 0), 'HpibEOL');

    (* We will now configure all devices that can respond to a parallel
     *  poll.  This example assumes devices at addresses 20 and 7 can
     *  respond to a parallel poll.  see operators manual of individual
     *  devices to see if they can respond to a parallel poll.
     *)

    (* configure the device at address 20 for a parallel poll
     *)
    error_handle(HpibPpollc(hHpib, device_addr_1, $08), 'HpibPPOLLC #1');

    (* configure the device at address 7 for a parallel poll
     *)
    error_handle(HpibPpollc(hHpib, device_addr_2, $09), 'HpibPPOLLC #2');

    (* configure any other devices that can respond to parallel poll here
     *)

end;

begin  { main }
```

```
    initialize;

    setup;

    while not keypressed do
    begin
        error_handle(HpibStatus(hHpib, isc, 1, response), 'HpibSTATUS #1');
        if response = 1 then
            check_srq;
        get_data ;
    end;

    cleanup;
end.
```

Reference

This section presents a detailed HP-IB Dynamic Link Library syntax reference for Pascal, C, and Visual BASIC languages. Parameters for Library commands are separated into several groups according to the types of arguments you must provide. The following table summarizes these groups. See "Command Library Parameters" in chapters 5 and 6 of *Using the HP-IB Interface and Command Library* for more detail about parameter types for Pascal and C.

Parameter Type	Pascal	C	Visual BASIC
Select Codes and Addresses	LongInt	Long-integer expression	Long&
Flags and Discrete Information*	Integer	Integer variable or expression	Integer%
Numeric Data (Single)*	Real (for HpibEnter)—double (for HpibOutput and HpibTimeout)	Float variable (for HpibEnter)—double variable or expression (for HpibOutput and HpibTimeout)	Single! (for HpibEnter)—Double# (for HpibOutput and HpibTimeout)
Numeric Data (Array)*	Real array	Float array	Single! array
Binary Data (Array)*	Any type of array	Any type of array	Any type of array
String Data*	String variable	String variable or expression	String$
Character Data	Character	Character expression	String$

* For parameters marked *_REF*, a variable or array must be passed by reference.

You can use literals and expressions for simple parameters that provide information to the command—but *not* for parameters that return information.

Parameters that must be passed by reference are indicated by *_REF* attached to their designators in this section. For C, all array variables must be passed by far reference.

Throughout this section, HP-IB terms are listed by abbreviation rather than by name. For example, "Go To Local" is listed as "GTL." The following list defines the standard HP-IB abbreviations (mnemonics) used in this section:

Mnemonic	**Definition**
ATN	Attention
DCL	Device Clear
EOI	End or Identify
EOL	End of Line
GET	Group Execute Trigger
GTL	Go To Local
IFC	Interface Clear
LAD	Listen Address
LLO	Local Lockout
MLA	My Listen Address
MTA	My Talk Address
OSA	Other Secondary Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable
PPU	Parallel Poll Unconfigure
REN	Remote Enable
SDC	Selected Device Clear
SPD	Serial Poll Disable
SPE	Serial Poll Enable
SRQ	Service Request
TAD	Talk Address
TCT	Take Control
UNL	Unlisten
UNT	Untalk

HpibAbort

This command aborts all activity on the interface. HpibAbort will abort as much as it can depending upon its current system controller and active controller status.

Syntax

HpibAbort (*hHpib*,*select_code*)

hHpib specifies the handle returned by HpibOpen.

select_code specifies the interface select code.

Examples

For Pascal:

```
error  : INTEGER ;
hHpib  : INTEGER ;

error  := HpibAbort(hHpib,7)
if error <> NOERR then writeln('an error occurred...');
```

For C:

```
int    error ;
HANDLE hHpib ;

error = HpibAbort(hHpib,7L)
if (error != NOERR)  /*Do error handling */
```

For Visual BASIC:

```
dev& = 7
errnum% = HpibAbort(hHpib%,DEV&)
if errnum% <> NOERR then 'Do error handling
```

Bus Activity

If the HP 82335 is system controller:

- IFC is pulsed (at least 100 microseconds).
- REN is set.
- ATN is cleared.

If the HP 82335 is active, but not system controller:

- UNT is sent.

If the HP 82335 is neither active nor system controller:

- No bus activity.

Comments

Devices in Local Lockout will remain locked out.

Possible errors are NOERR, ESEL, ENOOPEN, and EHANDLE.

If the HP 82335 was the system, but not active controller, HpibAbort will make the HP 82335 both system and active controller.

HpibClear

This command returns a device to a known, device-dependent state. It can be addressed to the interface or to a specific device.

Syntax

`HpibClear (`*hHpib*`,`*device_address*`)`
`HpibClear (`*hHpib*`,`*select_code*`)`

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies the address of a device to be cleared.
select_code	specifies the select code of the interface on which all devices are to be cleared.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibClear (hHpib,723); {Clear the device at address 23.}

err := HpibClear (hHpib,7); {Clear all devices.}
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibClear(hHpib,723L); /*Clear device at address 23.*/
    .
    .
error = HpibClear(hHpib,7L); /*Clear all devices.*/
```

For Visual BASIC:

```
isc& = 7
dvm& = 723
errnum% = HpibClear(hHpib%,dvm&) 'Clear the device at address 23.
  .
  .
errnum% = HpibClear(hHpib%,isc&) 'Clear all devices.
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- SDC is sent.

If a select code is specified:

- ATN is set.
- DCL is sent.

Comments

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibClose

This command closes an interface, making it available to other applications. Once an interface is closed, no other HP-IB functions can be called (except HpibOpen). The state of the interface is not stored, so when an application calls HpibOpen again, you need to reset conditions such as EOL, EOI, TIMEOUT, MATCH, and FASTOUT.

While developing and debugging a program, an HP-IB card may be incorrectly left open. You can use the Interactive Environment to manually close all cards by selecting Close Cards from the Misc menu.

Syntax

`HpibClose (`*HANDLE*`)`

HANDLE specifies the handle returned by HpibOpen (the second parameter).

Examples

For Pascal:

```
    err : INTEGER;
  hHpib : INTEGER;

  err := HpibClose (hHpib);
```

For C:

```
  int     err;
  HANDLE  hHpib;

  err = HpibClose(hHpib);
```

For Visual BASIC:

```
  errnum% = HpibClose(hHpib%);
```

Bus Activity

None.

Comments

Possible errors are ECLOSE if the handle is invalid, usually meaning that the interface was not open; and NOERR if successful.

HpibControl

This command directly sets status conditions in the interface. It can be used to address or unaddress the interface as a talker or listener, or set the interface's bus address. HpibControl can also change system controller status of the HP 82335 interface.

Caution HpibControl should be used with caution since it operates directly on the interface.

Syntax

`HpibControl` (*hHpib*, *select_code*, *condition*, *status*)

hHpib specifies the handle returned by HpibOpen.

select_code specifies the interface select code.

condition specifies the status condition that is to be set. Conditions which can be set are:

Value	Description
3	Make the interface the non-system or system controller.
5	Address or unaddress the interface as talker.
6	Address or unaddress the interface as listener.
7	Set the interface's bus address.
8	Clear or set ATN.

status variable into which the condition's status is placed. It can have the following values:

Condition 3

Value	Meaning
0	Make interface non-system controller
1	Make interface system controller

Conditions 5 and 6

Value	Meaning
0	Clear this condition
1	Set specified condition

Condition 7

Value	Meaning
0 to 30	Bus address of interface

Condition 8

Value	Meaning
0	Clear ATN
1	Set ATN asynchronously
2	Set ATN synchronously
Other	ERANGE error

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibControl(hHpib,7,5,1);{Address interface as talker.}
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibControl(hHpib,7L,5,1);/*Address interface as talker.*/
```

For Visual BASIC:

```
isc& = 7
cond% = 5
status% = 1
errnum% = HpibControl(hHpib%,isc&,cond%,status%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Address the interface as talker
```

Bus Activity

None.

Comments

Possible errors are NOERR, ESEL, ECTRL, ETIME, ENOOPEN, EHANDLE, and ERANGE.

The functionality for changing system controller status of the HP 82335 is included for completeness in the HP-IB DLL. We strongly recommend, however, that you do not use this command unless it is absolutely necessary. The recommended method of using the interface as a non-system controller is to use the IOSYSCTL DOS command in your AUTOEXEC.BAT file.

For condition 8, you can set ATN either synchronously or asynchronously. Typically, you will set ATN asynchronously. If so, data may get lost if a data transfer is occurring that does not involve the HP 82335. For example, if a scope is talking to a printer and ATN is set asynchronously, some data may have been lost. If you want to avoid this situation, use status 2 to set ATN synchronously.

HpibEnter

This command reads a single real number. Reading continues until one of these events occurs:

- The EOI line is sensed true, if it is enabled.
- A linefeed is encountered after the number starts.

Numeric characters are the digits 0 through 9, "E", "e", "+", "–", and "." in the proper sequence for representing a number. Note that " " (space) is not a numeric character.

Syntax

`HpibEnter (`*hHpib*`,`*device_address*`,`*data_REF*`)`
`HpibEnter (`*hHpib*`,`*select_code*`,`*data_REF*`)`

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	variable into which the reading is placed.

Examples

For Pascal:

```
VAR
 reading : SINGLE;
     err : INTEGER;
   hHpib : INTEGER;

err := HpibEnter (hHpib,722,reading); {Input a number from device
                                722 and place it in READING.}
```

For C:

```
float     reading;
int       error;
HANDLE    hHpib;

error = HpibEnter(hHpib,722L,&reading); /*Input a number from
                                        device 722.*/
```

For Visual BASIC:

```
device& = 722
errnum% = HpibEnter(hHpib%,device&,reading!)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Input a number from device 722 and place it in reading
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.

If a select code is specified:

- If the interface is not addressed to listen, an error results.
- If the interface is addressed to listen, ATN is cleared and the data is read from the interface.

Comments

If a select code is to be specified in the command, the interface must first be addressed to listen (with HpibSend, for example) or an error occurs.

The approximate range of valid values is 10^{-38} to 10^{38}. The IEEE 754 standard for floating point numbers makes provisions for values less than 10^{-38}, however the internal number conversion may not properly handle

values less than 10^{-38} when entered via HP-IB or used in assignment or print statements.

Possible errors are NOERR, ETIME, ESEL, EADDR, ECTRL, ENOOPEN, EHANDLE, and ENUM.

HpibEntera

This command enters numbers from a device or the interface and places them into a real array. Reading continues until one of these events occurs:

- The EOI line is sensed true, if it is enabled.
- A linefeed is encountered after the specified number of elements is received.

Numeric characters are the digits 0 through 9, "E", "e", "+", "−", and "." in the proper sequence for representing a number. Note that " " (space) is not a numeric character.

Syntax

`HpibEntera` (*hHpib*, *device_address*, *readings_REF*, *elements_REF*)
`HpibEntera` (*hHpib*, *select_code*, *readings_REF*, *elements_REF*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
readings_REF	array into which the readings are placed.
elements_REF	variable that specifies the maximum number of elements to be read. (An error occurs if the number is less than 0.) Upon return it indicates the number of elements actually received.

Examples

For Pascal:

```
TYPE
 real50 = ARRAY[1..50] of SINGLE;
VAR
 readings : real50;
 elements : INTEGER;
    hHpib : INTEGER;
      err : INTEGER;

elements := 50;
err := HpibEntera (hHpib,723,readings,elements); {Read a maximum
       of 50 values from device 723 and put them in READINGS.}
```

For C:

```
float     readings[50];
int       elements;
int       error;
HANDLE    hHpib;

elements = 50;
error = HpibEntera(hHpib,723L,readings,&elements);
        /*Read a maximum of 50 values from device 723.*/
```

For Visual BASIC:

```
dim readings!(50)
device& = 723
max% = 50
errnum% = HpibEntera(hHpib%,device&,readings!(0),max%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Read a maximum of 50 values from device 723 and
'put them in readings.
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.

If a select code is specified:

- If the interface is not addressed to listen, an error results.
- If the interface is addressed to listen, ATN is cleared and the data is read from the interface.

Comments

If the specified maximum number of elements to read is greater than the size of the *readings* array, input data can overrun the array and corrupt existing data or programs.

Nonnumeric characters that do not properly belong in a real number are considered value separators. Thus, the sequence "1,234,567" is entered as three numbers, not as "1234567".

The number of readings available is dependent upon the source device.

The approximate range of valid values is 10^{-38} to 10^{38}. The IEEE 754 standard for floating point numbers makes provisions for values less than 10^{-38}, however the internal number conversion may not properly handle values less than 10^{-38} when entered via HP-IB or used in assignment or print statements.

If a select code is to be specified in the command, the interface must first be addressed to listen (with HpibSend, for example) or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ENUM, ECTRL, ENOOPEN, EHANDLE, and ERANGE.

HpibEnterab

This command enters arbitrary-block program data (numeric or string data with IEEE-488.2 coding) from a device or the interface. Reading continues until one of these events occurs:

- The maximum number of bytes specified is received.
- A linefeed is encountered with the EOI line sensed true, if the coding indicates indefinite length.
- The number of bytes indicated by the coding is received, if the coding indicates definite length.

Syntax

`HpibEnterab` (*hHpib*, *device_address*, *data_REF*, *bytes_REF*, *swapsize*)
`HpibEnterab` (*hHpib*, *select_code*, *data_REF*, *bytes_REF*, *swapsize*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	array into which the readings are placed.
bytes_REF	variable specifying the maximum number of bytes to be read (excluding the coding bytes). (An error occurs if the number is less than 0.) Upon return it indicates the number of bytes actually received (excluding the coding bytes).
swapsize	specifies how bytes are placed into memory. A value of 1 indicates that bytes are placed in order. Larger values indicate that bytes are reversed in memory in groups of this size. The value should correspond to the byte size of the *data* variable. (For example, a value of 4 specifies that each group of four bytes is swapped in memory.) Valid values are 1 through 8—other values return an error.

Examples

For Pascal:

```
TYPE
 double50 = ARRAY[1..50] of REAL8;
VAR
      val : double50;
 elements : INTEGER;
     swap : INTEGER;
      err : INTEGER;
    hHpib : INTEGER;

swap := 8;
elements := 50 * swap;
err := HpibEnterab (hHpib,723,val,elements,swap); {Read a maximum
       of 50 values from device 723 and put them in VAL.}
```

For C:

```
double    val[50]; /*Double-precision array (8 bytes/elem)*/
int       elements;
int       swap;
int       error;
HANDLE    hHpib;

swap = sizeof(double);
elements = 50 * swap;
error = HpibEnterab(hHpib,723L,val,&elements,swap);
        /*Read a maximum of 50 values from device 723.*/
```

For Visual BASIC:

```
dim val#(50) 'Double-precision array (8 bytes/elem)
device& = 723
swap% = 8 : max% = 50 * swap%
errnum% = HpibEnterab(hHpib%,device&,val#(0),max%,swap%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Read a maximum of 50 values from device 723 and
'put them in VAL.
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.

If a select code is specified:

- If the interface is not addressed to listen, an error results.
- If the interface is addressed to listen, ATN is cleared and the data is read from the interface.

Comments

IEEE-488.2 coding is described under "Arbitrary-Block Data Coding" in chapter 1 of *Using the HP-IB Interface and Command Library*. The coding bytes are not placed into *data*—this also applies to the ending linefeed character for indefinite-length data. Leading characters are ignored until a "#" character is received.

If the specified maximum number of elements to read is greater than the size of the *data* array, input data can overrun the array and corrupt existing data or programs.

For string transfers, only the string elements receiving data are affected. The string descriptor and other string elements remain unchanged for Pascal—no null character is appended for C.

The number of bytes available is dependent upon the source device.

If a select code is to be specified in the command, the interface must first be addressed to listen (with HpibSend, for example) or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ERANGE, ECTRL, ENOOPEN, EHANDLE, and EUNKNOWN.

HpibEnterb

This command enters binary data (numeric or string data with no coding or formatting) from a device or the interface. Reading continues until one of these events occurs:

- The maximum number of bytes specified is received.
- The EOI line is sensed true, if it is enabled.
- The termination character set by HpibMatch is received with EOI true. (Linefeed is the default character.)

Syntax

`HpibEnterb` (*hHpib*, *device_address*, *data_REF*, *bytes_REF*, *swapsize*)
`HpibEnterb` (*hHpib*, *select_code*, *data_REF*, *bytes_REF*, *swapsize*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	array into which the readings are placed.
bytes_REF	specifies the maximum number of bytes to be read. (An error occurs if the number is less than 0.) Upon return it indicates the number of bytes actually received.
swapsize	specifies how bytes are placed into memory. A value of 1 indicates that bytes are placed in order. Larger values indicate that bytes are reversed in memory in groups of this size. The value should correspond to the byte size of the *data* variable. (For example, a value of 4 specifies that each group of four bytes is swapped in memory.) Valid values are 1 through 8—other values return an error.

Examples

For Pascal:

```
TYPE
 double50 = ARRAY[1..50] of REAL8;
VAR
      val : double50
 elements : INTEGER;
     swap : INTEGER;
      err : INTEGER;
    hHpib : INTEGER;

swap := 8;
elements := 50 * swap;
err := HpibEnterb (hHpib,723,val,elements,swap); {Read a maximum
       of 50 values from device 723 and put them in VAL.}
```

For C:

```
double    val[50]; /*Double-precision array (8 bytes/elem)*/
int       elements;
int       swap;
int       error;
HANDLE    hHpib;

swap = sizeof(double);
elements = 50 * swap;
error = HpibEnterb(hHpib,723L,val,&elements,swap);
        /*Read a maximum of 50 values from device 723.*/
```

For Visual BASIC:

```
dim val#(50) 'Double-precision array (8 bytes/elem)
device& = 723
swap% = 8 : max% = 50 * swap%
errnum% = HpibEnterb(hHpib%,device&,val#(0),max%,swap%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
"Read a maximum of 50 values from device 723 and
'put them in val.
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.

If a select code is specified:

- If the interface is not addressed to listen, an error results.
- If the interface is addressed to listen, ATN is cleared and the data is read from the interface.

Comments

If the specified maximum number of elements to read is greater than the size of the *data* array, input data can overrun the array and corrupt existing data or programs.

All data received is stored in memory—except a final "match" character with EOI true if matching is enabled. For string transfers, only the string elements receiving data are affected. The string descriptor and other string elements remain unchanged for Pascal—no null character is appended for C.

The number of bytes available is dependent upon the source device.

If a select code is to be specified in the command, the interface must first be addressed to listen (with HpibSend, for example) or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ERANGE, ECTRL, ENOOPEN, EHANDLE, and EUNKNOWN.

HpibEnterf

This command reads from a device and places all received data into a file. Reading continues until one of these events occurs:

- The EOI line is sensed true, if it is enabled.
- The termination character set by HpibMatch is received (linefeed is the default). **Warning:** If you are transferring binary files, you should turn off character match using HpibMatch to make sure the transfer does not end prematurely.
- The maximum number of bytes specified is received.
- A file error occurs, usually meaning the disk is full.

Syntax

`HpibEnterf` (*hHpib*, *device_address*, *file_name*, *length*, *append_flag*)
`HpibEnterf` (*hHpib*, *select_code*, *file_name*, *length*, *append_flag*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
file_name	file into which the data is written.
length	specifies the maximum number of elements to be read. (An error occurs if the number is less than 0.) The actual number of bytes read is returned here.
append_flag	specifies whether to append to the file or to overwrite it. Zero overwrites; non-zero appends.

Examples

For Pascal:

```
error  : INTEGER ;
length : LONGINT ;
hHpib  : INTEGER;

length := 10;
error := HpibEnterf(hHpib,723, 'ENTERF.DAT',length, 0)
if error <> NOERR then writeln('an error occurred...');
```

For C:

```
int    error;
long   length;
HANDLE hHpib;

length = 10;
error = HpibEnterf(hHpib,723L, "ENTERF.DAT", &length, 0)
if (error != NOERR) /*Do error handling*/
```

For Visual BASIC:

```
dev& = 723
length& = 10
file.name$ = "enter.dat"
append% = 0
errnum% = HpibEnterf(hHpib%,dev&,file.name$,length&,append%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.

If a select code is specified:

- If the interface is not addressed to listen, an error results.
- If the interface is addressed to listen, ATN is cleared and the data is read from the interface.

Comments

Possible errors are NOERR, ETIME, ESEL, EADDR, ERANGE, ECTRL, ENOOPEN, EHANDLE, and EFILE.

If the file does not exist and a valid filename is given, HpibEnterf will create the file regardless of the append flag.

We recommend turning off character matching using the HpibMatch command, especially if you are transferring a binary file.

Note

This command does **not** transfer files to an HP-IB disk drive, but rather transfers bytes from the HP-IB bus to a built-in disk drive on your computer.

HpibEnters

This command enters a character string from a device or the interface. Reading continues until one of these events occurs:

- The EOI line is sensed true, if it is enabled.
- The termination character set by HpibMatch is received (linefeed is the default).
- The maximum number of characters specified is received.

Syntax

`HpibEnters` (*hHpib*, *device_address*, *data_REF*, *length_REF*)
`HpibEnters` (*hHpib*, *select_code*, *data_REF*, *length_REF*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	variable into which the string read is placed.
length_REF	variable specifying the maximum number of elements to be read. (An error occurs if the number is less than 0.) On return it indicates the number of elements actually received.

Examples

For Pascal:

```
VAR
 info   : STRING(10);
 length : INTEGER;
 err    : INTEGER;
 hHpib  : INTEGER;

length := 10;
err := HpibEnters (hHpib,723,info,length);
       {Read a string of 10 characters maximum from device 723.}
```

For C:

```
int     error;
int     length;
char    info[11];  /*10 characters plus null*/
HANDLE  hHpib;

length = 10;
error = HpibEnters(hHpib,723L,info,&length); /*Read a string of
        10 characters maximum from device 723.*/
```

For Visual BASIC:

```
dev& = 723
max.len% = 10
info$ = space$(max.len%)
errnum% = HpibEnters(hHpib%,dev&,info$,max.len%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Read a string of 10 characters maximum from
'device 723, put in info$
info$ = left$(info$,max.len%)
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.

If a select code is specified:

- If the interface is not addressed to listen, an error results.
- If the interface is addressed to listen, ATN is cleared and the data is read from the interface.

Comments

If the specified maximum number of elements to read is greater than the dimensioned length of the *data* string, then input data can overrun the string and corrupt existing data or programs.

If a select code is to be specified in the command, the interface must first be addressed to listen (with HpibSend or a previous HpibEnter, for example) or an error occurs.

The termination character is entered as part of the string. A null character is appended to the string.

Possible errors are NOERR, ETIME, ESEL, EADDR, ENUM, ERANGE, ENOOPEN, EHANDLE, ECTRL, and EUNKNOWN.

HpibEoi

This command enables or disables the End Or Identify (EOI) mode of the interface. It is used to:

- Enable or disable a write operation to set the EOI line on the last byte of the write.
- Enable or disable a read operation to terminate upon sensing the EOI line true.

The default is EOI enabled.

Syntax

`HpibEoi (`*`hHpib,select_code,state`*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
state	enables EOI if nonzero and disables EOI if zero.

Examples

For Pascal:

```
VAR
 state : INTEGER;
   err : INTEGER;
 hHpib : INTEGER;

state := 0;
  err := HpibEoi(hHpib;7,state); {Disable EOI.}
```

For C:

```
int     error;
HANDLE  hHpib;
error = HpibEoi(hHpib,7L,0); /*Disable EOI.*/
```

For Visual BASIC:

```
isc& = 7
state% = 0
errnum% = HpibEoi(hHpib%,isc&,state%) 'Disable EOI
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

None.

Comments

When reading with EOI enabled, receipt of a byte with EOI set causes the read operation to terminate, regardless of whether you are reading a string, a real number, or an array of real numbers. (The EOI state is ignored by HpibEnterab.)

When writing, EOI is set on the last byte of the End Of Line sequence if EOI is enabled. Note that if the EOL sequence is of 0 length, EOI is set on the last data byte sent. (The EOI line is *not* set on the last byte for HpibOutputab.)

When sending a real number array with HpibOutputa, the EOL sequence (and subsequent EOI) is appended after the last element in the array, not after each element.

Note that HpibSend does not set EOI because this line has a different meaning in Command mode.

Possible errors are NOERR, ENOOPEN, EHANDLE, and ESEL.

HpibEol

This command defines the End of Line (EOL) string that is to be sent following every HpibOutput, HpibOutputa, HpibOutputb, and HpibOutputs command.

The default is carriage return and linefeed.

Syntax

`HpibEol` (*hHpib*, *select_code*, *endline_REF*, *length*)

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
endline_REF	specifies the EOL string that is to be sent following a data transmission. A maximum of eight characters can be specified.
length	specifies the length of the termination string. If zero is specified, no characters are appended to a data transmission. If the length is less than 0 or more than 8, an error occurs.

Examples

For Pascal:

```
VAR
  length : INTEGER;
 endline : STRING(2);
     err : INTEGER;
   hHpib : INTEGER;

    length := 2;
endline[1] := CHR(13);
endline[2] := CHR(10);
       err := HpibEol(hHpib,7,endline,length); {EOL = CR/LF.}
```

For C:

```
int     length;
char    endline[2];
int     error;
HANDLE  hHpib;

length = 2;
endline[0] = 13;
endline[1] = 10;
error = HpibEol(hHpib,7,endline,length); /*EOL  =  CR/LF.*/
```

For Visual BASIC:

```
isc& = 7
endline$ = chr$(13)+chr$(10)
length% = len(endline$)
errnum% = HpibEol(Hhpib%,isc&,endline$,length%)'eol = cr/lf
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

None.

Comments

With HpibOutputa and HpibOutputb, the EOL sequence is appended after all data has been sent, not following each element.

Possible errors are NOERR, ESEL, ENOOPEN, EHANDLE, and ERANGE.

HpibFastout

This command enables or disables high-speed bus timing for output transfers only.

The default is high-speed output disabled (standard speed).

Syntax

`HpibFastout (`*hHpib*`,`*select_code*`,`*state*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
state	enables high-speed output if nonzero and disables high-speed output if zero.

Examples

For Pascal:


```
VAR
  state : INTEGER;
    err : INTEGER;
  hHpib : INTEGER;

state := 0;
  err := HpibFastout(hHpib,7,state); {Disable high-speed output.}
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibFastout(hHpib,7L,0); /*Disable high-speed output.*/
```

For Visual BASIC:

```
isc& = 7
state% = 0
'Disable high-speed output
errnum% = HpibFastout(hHpib%,isc%,state%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

None.

Comments

For proper operation, high-speed output requires the HP-IB system to meet all of these requirements:

- All HP-IB devices must have tri-state drivers, not open-collector drivers. (The HP-IB interface meets this requirement.)
- All HP-IB devices must be turned on.
- HP-IB cable length should be as short as possible, but not longer than 15 meters (50 feet). At least one HP-IB device should be connected for each meter (3 feet) of cable, with a maximum of 15 devices. (The HP-IB interface counts as one device.)
- Each HP-IB device must have a capacitance of less than 50 pF on each HP-IB line except REN and IFC. (The HP-IB interface meets this requirement.)

High-speed output applies only during output transfers—but not between transfers and not during input transfers. The speed of an input transfer depends upon the talker device.

High-speed output decreases the data-settling time from 2.5 microseconds to 840 nanoseconds.

Possible errors are NOERR, ENOOPEN, EHANDLE, and ESEL.

HpibGetterm

This command determines the reason the last read terminated.

Syntax

`HpibGetterm (`*hHpib*`,`*select_code*`,`*reason_REF*`)`

hHpib specifies the handle returned by HpibOpen.

select_code specifies the interface select code.

reason_REF variable to receive the sum of the values for the reasons the last read terminated. The possible reasons for termination are

Value	Description
0	The read was terminated for some reason not covered by any of the other reasons.
1	The expected number of elements was received.
2	The termination character set by HpibMatch was encountered.
4	The EOI line was sensed true.

Examples

For Pascal:

```
VAR
 reason : INTEGER;
    err : INTEGER;
  hHpib : INTEGER;

err := HpibGetterm(hHpib,7,reason);
IF ((reason and 4) = 4) then
  WRITELN('EOI ENCOUNTERED');
```

For C:

```
int     reason;
int     error;
HANDLE  hHpib;

error = HpibGetterm(hHpib,7L,&reason);
if((reason & 4) == 4)
    /*Do error checking*/
```

For Visual BASIC:

```
isc& = 7
errnum% = HpibGetterm(hHpib%,isc&,reason%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
if ((reason% and 4) = 4) then print "eoi encountered"
```

Bus Activity

None.

Comments

Upon return, the reason integer contains the sum of the values for the reasons for termination. For example, if the last read terminated when the termination character was encountered and EOI was set, the value of reason would be $2 + 4 = 6$.

Possible errors are NOERR, ENOOPEN, EHANDLE, and ESEL.

HpibGetVersion

This command returns an integer representing the version of the installed HP-IB DLL. The major revision is returned in the high byte and the minor revision is returned in the low byte.

Syntax

```
HpibGetVersion ()
```

Examples

For Pascal:

```
VAR
version: INTEGER;

version:=HpibGetVersion;
```

For C:

```
int     version;

version=HpibGetVersion();
```

For Visual BASIC:

```
version%=HpibGetVersion()
```

Bus Activity

None.

HpibLlockout

This command sends a Local Lockout (LLO) to disable a device front panel. It is received by all devices on the interface, whether or not they are addressed to listen.

Syntax

`HpibLlockout (`*`hHpib,select_code`*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

 err := HpibLlockout (hHpib,7);
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibLlockout(hHpib,7L);
```

For Visual BASIC:

```
isc& = 7
errnum% = HpibLlockout(hHpib%,isc&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

- ATN is sent.
- LLO is sent.

Comments

If a device is in Local mode when LLO is received, LLO does not take effect until the device is addressed to listen.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibLocal

This command executes a Go To Local (GTL) or clears the REN line to enable a device front panel.

Syntax

`HpibLocal (`*hHpib*`,`*device_address*`)`
`HpibLocal (`*hHpib*`,`*select_code*`)`

hHpib specifies the handle returned by HpibOpen.

device_address specifies a device address.

select_code specifies the interface select code.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibLocal (hHpib,722); {Place device 722 in Local mode.}
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibLocal(hHpib,722L);/*Place device 722 in local mode.*/
```

For Visual BASIC:

```
device& = 722
'Place device 722 in local mode.
errnum% = HpibLocal(hHpib%,device&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- GTL is sent.

If a select code is specified:

- REN is cleared.
- ATN is cleared.

Comments

If a device address is specified, the device is temporarily placed in Local mode—it will return to Remote mode if it is later addressed to listen. If Local Lockout is in effect, the device will return to the Lockout state if it is later addressed to listen.

If an interface select code is specified, all instruments on the bus are placed in Local mode and any Local Lockout is cancelled.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibMatch

This command defines the character used by HpibEnterb and HpibEnters for termination. The default character is linefeed.

Syntax

`HpibMatch (`*hHpib*`,`*select_code*`,`*character*`,`*flag*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
character	specifies the character used by HpibEnterb and HpibEnters for termination checking.
flag	indicates whether character matching should be enabled or disabled. Zero disables matching, and any nonzero value enables it.

Examples

For Pascal:

```
VAR
 match : CHAR;
 flag  : INTEGER;
   err : INTEGER;
 hHpib : INTEGER;

match := CHR(10); {Terminate on linefeed.}
 flag := 1;
  err := HpibMatch (hHpib,7,match,flag);
```

For C:

```
char    match;
int     flag;
int     error;
HANDLE  hHpib;

match = 10; /*Terminate on linefeed.*/
flag = 1;
error = HpibMatch(hHpib,7L,match,flag);
```

For Visual BASIC:

```
isc& = 7
match% = 10 'Terminate on a linefeed.
flag% = 1
errnum% = HpibMatch(hHpib%,isc&,match%,flag%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

None.

Comments

Only a single match character may be specified in this command.

For HpibEnters, the match character becomes part of the entered string. For HpibEnterb, the match character must be received with EOI true, and the character does *not* become part of the data.

HpibMatch does not apply to HpibEnter, HpibEntera, or HpibEnterab.

Possible errors are NOERR, ENOOPEN, EHANDLE, and ESEL.

HpibOpen

This command opens up an interface for use by an application. An application must call HpibOpen before any other HP-IB functions can be used. An interface remains open for use until the application calls HpibClose.

While developing and debugging a program, an HP-IB card may be incorrectly left open. You can use the Interactive Environment to manually close all cards by selecting Close Cards from the Misc menu.

Syntax

`HpibOpen (`*`long`*`,`*`HANDLE_REF`*`)`

long	specifies the Interface Select Code (ISC) of the card to be opened. Valid ISCs range from 1 through 16.
HANDLE_REF	points to a handle that HpibOpen will return. This handle must be used as the first parameter in all subsequent HP-IB calls.

Example

For Pascal:

```
    err : INTEGER;
hHpib : INTEGER;

 err := HpibOpen (7,hHpib)
```

For C:

```
int       err;
HANDLE    hHpib;

err = HpibOpen(7L,&hHpib)
```

For Visual BASIC:

```
errnum% = HpibOpen(7,hHpib%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

None.

Comments

Possible errors are EOPEN if the first parameter is out of range, or if another application has already allocated this card; ESEL if no card exists at that ISC; and NOERR if successful.

HpibOutput

This command outputs a real number to a device or to the interface. After the number is sent, the EOL string is sent and the EOI line is set (if enabled).

Syntax

`HpibOutput` (*hHpib*, *device_address*, *data*)
`HpibOutput` (*hHpib*, *select_code*, *data*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data	specifies the number to be output.

Examples

For Pascal:

```
VAR
 data : SINGLE;
  err : INTEGER;
hHpib : INTEGER;

data := 12.3;
 err := HpibOutput (hHpib,722,data); {Output ' 12.3' to dev 722.}
```

For C:

```
double    data;
int       error;
HANDLE    hHpib;

data = 12.3;
error = HpibOutput(hHpib,722L,data);/*Output '12.3' to dev 722.*/
```

For Visual BASIC:

```
info! = 12.3
dev& = 722
'Output " 12.3" to device 722
errnum% = HpibOutput(hHpib%,dev&,info!)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is output.
- EOL is output.

If a select code is specified:

- If the interface is not addressed to talk, an error results.
- If the interface is addressed to talk, ATN is cleared and the data is sent followed by an EOL.

Comments

Numbers with absolute values between 10^{-5} and 10^{6} are rounded to seven significant digits and output in floating point notation. If the number rounds to an integer value, the decimal point is not sent. Numbers outside this range are rounded to seven significant digits and output in scientific notation.

If the number is positive, a leading space is output for the sign; if it's negative, a leading "–" is output.

If a select code is to be specified, the interface must first be addressed to talk (with HpibSend, for example), or an error occurs.

Possible errors are NOERR, ETIME, ESEL, ECTRL, ENOOPEN, EHANDLE, and EADDR.

HpibOutputa

This command outputs an array of real numbers to a specified device or to the bus. Values output are separated by commas. After the last number is sent, the EOL string is sent and the EOI line is set (if enabled).

Syntax

`HpibOutputa` (*hHpib*, *device_address*, *data_REF*, *elements*)
`HpibOutputa` (*hHpib*, *select_code*, *data_REF*, *elements*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	array containing the real numbers to be transmitted.
elements	specifies the number of elements in the array to be transmitted. (An error occurs if the number is less than 0.)

Examples

For Pascal:

```
TYPE
 real10 = ARRAY[1..10] of SINGLE;
VAR
          info : real10;
 num_elements : INTEGER;
           err : INTEGER;
         hHpib : INTEGER;

num_elements := 10;
err := HpibOutputa (hHpib,722,info,num_elements);
       {Output the array INFO to device 722.}
```

For C:

```
float    info[10];
int      num_elements;
int      error;
HANDLE   hHpib;

num_elements = 10;
error = HpibOutputa(hHpib,722L,info,num_elements);
        /*Output elements of 'info' to device 722.*/
```

For Visual BASIC:

```
dim info!(10)
elements% = 10
dev& = 722
errnum% = HpibOutputa(hHpib%,dev&,info!(0),elements%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Output array info to device 722; begin with element 0.
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is output.
- EOL is output.

If a select code is specified:

- If the interface is not addressed to talk, an error results.
- If the interface is addressed to talk, ATN is cleared and the data is sent followed by an EOL.

Comments

If the specified maximum number of elements to output is greater than the size of the *data* array, the output transfer can go beyond the array and send meaningless data.

Numbers with absolute values between 10^{-5} and 10^{6} are rounded to seven significant digits and output in floating point notation. If the number rounds to an integer value, the decimal point is not sent. Numbers outside this range are rounded to seven significant digits and output in scientific notation.

If the number is positive, a leading space is output for the sign; if it's negative, a leading "–" is output.

If a select code is to be used as a parameter, the interface must first be addressed to talk (with HpibSend, for example), or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ECTRL, ENOOPEN, EHANDLE, and ERANGE.

HpibOutputab

This command outputs arbitrary-block response data (numeric or string data with IEEE-488.2 coding) to a specified device or to the bus. After the last data byte is sent, nothing additional occurs.

Syntax

`HpibOutputab` (*hHpib*, *device_address*, *data_REF*, *bytes*, *swapsize*)
`HpibOutputab` (*hHpib*, *select_code*, *data_REF*, *bytes*, *swapsize*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	array containing the data to be transmitted.
bytes	specifies the number of bytes to output (excluding the coding bytes). This value should be no more than the number of elements in the array times the number of bytes per element. (An error occurs if the number is less than 0.)
swapsize	specifies how bytes are read from memory. A value of 1 indicates that bytes are read in order. Larger values indicate that bytes are reversed as read from memory in groups of this size. The value should correspond to the byte size of the *data* variable. (For example, a value of 4 specifies that each group of four bytes is swapped when output.) Valid values are 1 through 8—other values return an error.

Examples

For Pascal:

```
TYPE
 double10 = ARRAY[1..10] of REAL8;
VAR
       val : double10;
 num_bytes : INTEGER;
      swap : INTEGER;
       err : INTEGER;
     hHpib : INTEGER;

swap := 8;
num_bytes := 10 * swap;
err := HpibOutputab (hHpib,722,info,num_elements,swap);
       {Output the array INFO to device 722.}
```

For C:

```
double   info[10]; /*Double-precision array (8 bytes/elem)*/
int      num_bytes;
int      swap;
int      error;
HANDLE   hHpib;

swap = sizeof(double);
num_bytes = 10 * swap;
error = HpibOutputab(hHpib,722L,info,num_bytes,swap);
        /*Output elements of 'info' to device 722.*/
```

For Visual BASIC:

```
dim info#(10) 'Double-precision array (8 bytes/elem)
swap% = 8
elements% = 10 * swap%
dev& = 722
errnum% = HpibOutputab(Hhpib%,dev&,info#(0),elements%,swap%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Output array info to device 722; begin with element 0.
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is output.

If a select code is specified:

- If the interface is not addressed to talk, an error results.
- If the interface is addressed to talk, ATN is cleared and the data is sent.

Comments

IEEE-488.2 coding is described under "Arbitrary-Block Data Coding" in chapter 1 of *Using the HP-IB Interface and Command Library*. The coding bytes are automatically computed and inserted in front of the data.

If the specified maximum number of elements to output is greater than the size of the *data* array, the output transfer can go beyond the array and send meaningless data.

If a select code is to be specified in the command, the interface must first be addressed to talk (with HpibSend, for example) or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ERANGE, ECTRL, ENOOPEN, EHANDLE, and EUNKNOWN.

HpibOutputb

This command outputs binary data (numeric or string data with no coding or formatting) to a specified device or to the bus. After the last number is sent, the EOL string is sent and the EOI line is set (if enabled).

Syntax

`HpibOutputb` (*hHpib*, *device_address*, *data_REF*, *bytes*, *swapsize*)
`HpibOutputb` (*hHpib*, *select_code*, *data_REF*, *bytes*, *swapsize*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	array containing the data to be transmitted.
bytes	specifies the number of bytes to output. This value should be no more than the number of elements in the array times the number of bytes per element. (An error occurs if the number is less than 0.)
swapsize	specifies how bytes are read from memory. A value of 1 indicates that bytes are read in order. Larger values indicate that bytes are reversed as read from memory in groups of this size. The value should correspond to the byte size of the *data* variable. (For example, a value of 4 specifies that each group of four bytes is swapped when output.) Valid values are 1 through 8—other values return an error.

Examples

For Pascal:

```
TYPE
 double10 = ARRAY[1..10] of REAL8;
VAR
       val : double10;
 num_bytes : INTEGER;
      swap : INTEGER;
       err : INTEGER;
     hHpib : INTEGER;

swap := 8;
num_bytes := 10 * swap;
err := HpibOutputb (hHpib,722,info,num_elements,swap);
       {Output the array INFO to device 722.}
```

For C:

```
double   info[10]; /*Double-precision array (8 bytes/elem)*/
int      num_bytes;
int      swap;
int      error;
HANDLE   hHpib;

swap = sizeof(double);
num_bytes = 10 * swap;
error = HpibOutputb(hHpib,722L,info,num_bytes,swap);
        /*Output elements of 'info' to device 722.*/
```

For Visual BASIC:

```
dim info#(10) 'Double-precision array (8 bytes/elem)
swap% = 8
elements% = 10 * swap%
dev& = 722
errnum% = HpibOutputb(hHpib%,dev&,info#(0),elments%,swap%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Output array info to device 722; begin with element 0.
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is output.
- EOL is output.

If a select code is specified:

- If the interface is not addressed to talk, an error results.
- If the interface is addressed to talk, ATN is cleared and the data is sent followed by an EOL.

Comments

If the specified maximum number of elements to output is greater than the size of the *data* array, the output transfer can go beyond the array and send meaningless data.

If a select code is to be specified in the command, the interface must first be addressed to talk (with HpibSend, for example) or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ERANGE, ECTRL, ENOOPEN, EHANDLE, and EUNKNOWN.

HpibOutputf

This command outputs the contents of a file to a specified device or interface. After the file is sent, the EOL string is sent and the EOI line is set (if enabled).

Syntax

`HpibOutputf (`*hHpib*`,`*device_address*`,`*file_name*`,`*length*`)`
`HpibOutputf (`*hHpib*`,`*select_code*`,`*file_name*`,`*length*`)`

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
file_name	specifies the name of the file to output.
length	specifies the maximum number of elements to be written. (An error occurs if the number is less than 0.)

Examples

For Pascal:

```
error   : INTEGER ;
length  : LONGINT ;
hHpib   : INTEGER;

length := 10 ;
error := HpibOutputf(hHpib,723, 'OUTPUT.DAT', length)
if error <> NOERR then writeln('an error occurred...');
```

For C:

```
int    error ;
long   length ;
HANDLE hHpib;

length = 10 ;
error = HpibOutputf(hHpib,723L, "OUTPUT.DAT", &length)
if (error != NOERR) /*Do error handling here*/
```

For Visual BASIC:

```
dev& = 723
length& = 10
file.name$="OUTPUT.DAT"
errnum% = HpibOutputf(hHpib%,dev&,file.name$,length&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is entered.
- EOL is output.

If a select code is specified:

- If the interface is not addressed to talk, an error results.
- If the interface is addressed to talk, ATN is cleared and the data is sent followed by the EOL string.

Comments

Possible errors are NOERR, ETIME, ESEL, EADDR, ERANGE, ECTRL, ENOOPEN, EHANDLE, and EFILE.

If you are transferring a binary file, we recommend that you turn off the EOL string using the HpibEol command. If you do not, the current EOL string will be appended to the file.

Note

This command does **not** transfer files from an HP-IB disk drive, but rather transfers bytes from a built-in disk drive on your computer to the HP-IB bus.

HpibOutputs

This command outputs a string to a specified device or to the interface. After the string is sent, the EOL string is sent and the EOI line is set (if enabled).

Syntax

`HpibOutputs` (*hHpib*, *device_address*, *data_REF*, *length*)
`HpibOutputs` (*hHpib*, *select_code*, *data_REF*, *length*)

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.
data_REF	array specifying the string to be sent.
length	specifies the length of the output string. (An error occurs if the number is less than 0.)

Examples

For Pascal:

```
VAR
   info : STRING(4);
 length : INTEGER;
    err : INTEGER;
  hHpib : INTEGER;

info := '1ST1';
length := 4;
err := HpibOutputs (hHpib,723,info,length);
       {Send the programming code '1ST1' to device 723.}
```

For C:

```
char    *info
int     length;
int     error;
HANDLE  hHpib;

info = "1ST1";
length = 4;
error = HpibOutputs(hHpib,723L,info,length);
        /*Send the programming code '1ST1' to device 723.*/
```

For Visual BASIC:

```
dev& = 723
info$ = "1st1"
length% = len(info$)
errnum% = HpibOutputs(hHpib%,dev&,info$,length%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Send "1st1" to device 723.
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- ATN is cleared.
- Data is output.
- EOL is output.

If a select code is specified:

- If the interface is not addressed to talk, an error results.
- If the interface is addressed to talk, ATN is cleared and the data is sent followed by an EOL.

Comments

If the specified maximum number of elements to output is greater than the current length of the *data* string, the output transfer can go beyond the string and send meaningless data.

If a select code is to be used in the command, the interface must first be addressed to talk (with HpibSend, for example), or an error occurs.

Possible errors are NOERR, ETIME, ESEL, EADDR, ECTRL, ENOOPEN, EHANDLE, and ERANGE.

HpibPassctl

This command passes active control from the HP 82335 HP-IB interface to a device on the bus. The device must be capable of taking control.

Syntax

`HpibPassctl` (*hHpib*, *device_address*)

hHpib specifies the handle returned by HpibOpen.

device_address specifies a device address.

Examples

For Pascal:

```
error  : INTEGER ;
hHpib  : INTEGER ;

error  := HpibPassctl(hHpib,723)
if error <> NOERR then writeln('an error occurred...');
```

For C:

```
int    error ;
HANDLE hHpib ;

error = HpibPassctl(hHpib,723L)
if (error !=NOERR) /*Do error checking here*/
```

For Visual BASIC:

```
dev& = 723
errnum% = HpibPassctl(hHpib%,dev&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- TCT is sent.
- ATN is cleared.

Comments

If your program does not seem to work properly after passing control, make sure that you do not have an interrupt (IRQ) conflict with another device. You can find out what IRQ your HP-IB board is using by running the INSTALL utility.

The HP-IB DLL defaults to address 30. If desired, you can change this using the HpibControl command.

The HpibPassctl command passes active control only. This command will not change the state of the system controller status of the HP 82335.

Possible errors are NOERR, ETIME, ESEL, ENOOPEN, EHANDLE, and ECTRL.

HpibPpoll

This command performs a parallel poll of the interface. It sets a variable to a value (0 to 255) representing the response of those instruments on the interface that respond to a parallel poll.

Syntax

`HpibPpoll (`*hHpib*`,`*select_code*`,`*response_REF*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
response_REF	variable into which the parallel poll response byte is to be placed. The allowable range is 0 to 255. The eight bits of the response byte correspond to the eight HP-IB data lines (DIO1 through DIO8). Thus, a value of 32 would indicate that some device has responded to the parallel poll with a "1" on DIO6.

Examples

For Pascal:

```
VAR
 response : INTEGER;
      err : INTEGER;
    hHpib : INTEGER;

err := HpibPpoll (hHpib,7,response);
```

For C:

```
int     response;
int     error;
HANDLE  hHpib;

error = HpibPpoll(hHpib,7L,&response);
```

For Visual BASIC:

```
isc& = 7
errnum% = HpibPpoll(hHpib%,isc&,response%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

- ATN and EOI are asserted for 25 microseconds.
- The poll byte is read.
- EOI is cleared.
- ATN is restored to its previous state.

Comments

During a parallel poll, each enabled device may put a "1" on an assigned HP-IB data line according to its service request status—otherwise, the line is a "0". There are eight data lines (though more than one device may be assigned to one line). Using a parallel poll, several devices can indicate their service request status simultaneously. The *response* variable contains the state of the eight data lines: DIO1 (in bit 0) through DIO8 (in bit 7).

If the *response* variable contains a "1" in any bit, a device assigned to the corresponding HP-IB line has the service request status the device was set up to report. (See HpibPpollC.) For example, a device may be set up to report on line DIO4 when it requests service. If an HpibPpoll command shows a "1" in bit 3 of *response*, your program knows the device needs service (assuming no other device is assigned to that line).

Not all devices are capable of responding to a parallel poll. Consult your particular device manuals for specifics.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibPpollc

This command performs a Parallel Poll Configure. In preparation for a parallel poll command, it tells an instrument how to respond affirmatively to the parallel poll, and on which data line to respond.

In general, it sets a parallel poll response byte to reflect the response of a desired arrangement of instruments. Typically, you could define the bits to reflect the responses of particular instruments, or the result of a logical OR operation on several instrument responses.

Refer to HpibPpoll for more information.

Syntax

`HpibPpollc (`*hHpib*`,`*device_address*`,`*configuration*`)`
`HpibPpollc (`*hHpib*`,`*select_code*`,`*configuration*`)`

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies the bus address of the device to be configured.
select_code	specifies the interface select code.
configuration	sent to the specified device indicating how it's to respond to a parallel poll. (See "Comments" below.)

Examples

For Pascal:

```
VAR
 configuration : INTEGER;
           err : INTEGER;
         hHpib : INTEGER;

configuration := 10; {Respond with a '1' on line DIO3.}

err := HpibPpollc (hHpib,723,configuration);
```

For C:

```
int     error;
int     configuration;
HANDLE  hHpib;

configuration = 10; /*Respond with a '1' on line DIO3.*/
error = HpibPpollc(hHpib,723L,configuration);
```

For Visual BASIC:

```
device& = 723
conf% = 10 'Respond with a "1" on line DIO3.
errnum% = HpibPpollc(hHpib%,device&,conf%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- PPC is sent.
- PPE is sent.

If a select code is specified:

- PPC is sent.
- PPE is sent.

Comments

The *configuration* parameter defines both the HP-IB line on which to respond and the service request status to indicate. It represents an eight-bit byte described below.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Response (0 or 1)	DIO Line (DIO1 to DIO8)		

Bit 3 specifies the meaning of an affirmative response. Bits 2 through 0 specify the data line (DIO8 through DIO1). The valid range for *configuration* is 0 to 15—other values cause an error.

Parallel Poll Configuration	Bits	Value
Affirmative response for service request	00001xxx	8
Affirmative response for no service request	00000xxx	0
Respond on line DIO8	0000x111	7
Respond on line DIO7	0000x110	6
Respond on line DIO6	0000x101	5
Respond on line DIO5	0000x100	4
Respond on line DIO4	0000x011	3
Respond on line DIO3	0000x010	2
Respond on line DIO2	0000x001	1
Respond on line DIO1	0000x000	0

For example, to set up a device to indicate an affirmative response ("1") on line DIO5 if it needs service, the configuration value would be 8 + 4 = 12. Alternatively, for the device to indicate an affirmative response ("1") on line DIO5 when it *doesn't* need service, the configuration value would be 0 + 4 = 4.

Not all devices can be configured to respond to a parallel poll. Consult your particular device manuals for specifics.

Possible errors are NOERR, ETIME, ESEL, ECTRL, ENOOPEN, EHANDLE, and ERANGE.

HpibPpollu

This command performs a Parallel Poll Unconfigure (PPU). It directs an instrument to not respond to a parallel poll. It can be addressed to the interface or a specific device. Refer to HpibPpollc for more information.

Syntax

`HpibPpollu (`*hHpib*`,`*device_address*`)`
`HpibPpollu (`*hHpib*`,`*select_code*`)`

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibPpollu (hHpib,722);
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibPpollu(hHpib,722L);
```

For Visual BASIC:

```
dev& = 722
errnum% = HpibPpollu(hHpib%,dev&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- PPC is sent.
- PPD is sent.

If a select code is specified:

- ATN is sent.
- PPU is sent.

Comments

Some devices cannot be unconfigured from the bus. Consult your particular device manuals for specifics.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibRemote

This command places a device in Remote mode to disable the device front panel. It can be addressed to the interface or to a specific device.

Syntax

`HpibRemote (`*hHpib*`,`*device_address*`)`
`HpibRemote (`*hHpib*`,`*select_code*`)`

hHpib	specifies the handle returned by HpibOpen.
device_address	specifies a device address.
select_code	specifies the interface select code.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibRemote (hHpib,723); {Places device 723 in Remote.}

err := HpibRemote (hHpib,7);   {Set the interface REN line.}
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibRemote(hHpib,723L); /*Place device 723 in Remote.*/
    .
    .
error = HpibRemote(hHpib,7L); /*Set the interface REN line.*/
```

For Visual BASIC:

```
isc% = 7
dvm& = 723
errnum% = HpibRemote(hHpib%,dvm&) 'Place the dvm in remote mode.
  .
  .
errnum% = HpibRemote(hHpib%,isc&) 'Set the interface REN line.
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- REN is set.
- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.

If a select code is specified, then REN is set.

Comments

If a select code is specified, a device will not switch into Remote mode until it is addressed to listen.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibRequest

This command sets up a serial poll status byte for the HP 82335 and optionally asserts the Service Request (SRQ) line.

Syntax

`HpibRequest (`*`hHpib`*`,`*`select_code`*`,`*`status`*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
status	specifies the serial poll status byte. If bit 6 in the status byte is set, the SRQ line will be asserted. If bit 6 is clear, SRQ will not be asserted.

Examples

For Pascal:

```
error  : INTEGER ;
hHpib  : INTEGER ;

error := HpibTakectl(hHpib,7, $042)
if error <> NOERR then writeln('an error occurred...');
```

For C:

```
int    error ;
HANDLE hHpib ;
error = HpibTakectl(hHpib,7L, 0x42)
if (error !=NOERR) /* Do error checking */
```

For Visual BASIC:

```
dev& = 7
status% = &h42
errnum% = HpibRequest(hHpib%,dev&,status%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If bit 6 is set in the status parameter: SRQ is asserted.

If bit 6 is clear in the status parameter: no bus activity.

Comments

Possible errors are NOERR, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibReset

This command sets the interface to its start-up state, in which it is not listening and not talking. In addition, it sets the following HP-IB parameters as indicated:

- The interface timeout is set to 0 seconds (the timeout is disabled).
- The interface EOI mode is enabled.
- High-speed data output is disabled.
- CR/LF is set as the EOL default.
- LF is set as the HpibMatch default.
- If the interface was system controller, then it will also become active controller.

Syntax

`HpibReset (`*`hHpib`*`,`*`select_code`*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibReset (hHpib,7);
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibReset(hHpib,7L);
```

For Visual BASIC:

```
isc& = 7
errnum% = HpibReset(hHpib%,isc&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If the interface is system controller:

- IFC is pulsed (at least 100 microseconds).
- REN is cleared (at least 10 microseconds).
- ATN is cleared.

If the interface is non-system controller:

- No bus activity.

Comments

This command returns all devices on the interface to local (front panel) control.

Possible errors are NOERR, ENOOPEN, EHANDLE, and ESEL.

HpibSend

This command sends any sequence of user-specified HP-IB commands to the interface. For example, to send an output command to several instruments simultaneously, you can establish their talk/listen status with the HpibSend command, then issue the output command specifying a select code rather than a device address.

Syntax

`HpibSend (`*hHpib*`,`*select_code*`,`*commands_REF*`,`*length*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
commands_REF	specifies a string of characters, each of which is treated as an interface command.
length	specifies the number of characters in the command string. (An error occurs if the number is less than 0.)

Examples

For Pascal:

```
VAR
 commands : STRING[4];
   length : INTEGER;
      err : INTEGER;
    hHpib : INTEGER;

commands := '?)/4'; {Specifies unlisten
            and listen addresses 9, 15, and 20.}
  length := 4;
     err := HpibSend (hHpib,7,commands,length);
     err := HpibTrigger (hHpib,7);
            {Triggers devices at addresses 9, 15, and 20.}
```

For C:

```
char    *commands;
int     length;
int     error;
HANDLE  hHpib;

commands = "?)/4"; /*Specifies unlisten
           and listen addresses 9, 15, and 20.*/
length = 4;
error = HpibSend(hHpib,7L,commands,length);
   .
   .
error = HpibTrigger(hHpib,7L);
        /*Triggers devices at addresses 9, 15, and 20.*/
```

For Visual BASIC:

```
isc& = 7
commands$ = "?)/4"
'Specifies unlisten, then listen addresses 9, 15, and 20.
length% = 4
errnum% = HpibSend(hHpib%,isc&,commands$,length%)
errnum% = HpibTrigger(hHpib%,isc&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Triggers devices at addresses 9, 15, and 20.
```

Bus Activity

- ATN is set.
- Command bytes are sent.

Comments

See appendix B in *Using the HP-IB Interface and Command Library* for a list of HP-IB commands and the corresponding data characters.

All bytes are sent with ATN set. The EOL sequence is not appended, nor is EOI set.

Possible errors are NOERR, ETIME, ESEL, ECTRL, ENOOPEN, EHANDLE, and ERANGE.

HpibSetWaitHook

HpibSetWaitHook sets up a function that the Windows DLL calls when it is waiting for devices to finish handshaking the HP-IB bus. If this function returns zero, the library continues normally. If it returns a non-zero value, the library halts as though a timeout error had occurred, but returns with the error value that the hook function returned.

HpibSetWaitHook has two main uses:

- An HP-IB transaction can be aborted manually even if the transaction was working properly.
- This hook function can call a PeekMessage loop to yield to windows, allowing the window to run normally even during long HP-IB transactions.

Syntax

`HpibSetWaitHook (`*hHpib*`,`*select_code*`,`*FARPROC*`)`

hHpib	specifies the handlep1179Xretbyrn HpibOpen.
select_code	specifies the interface select code.
FARPROC	points to the Abort function that the Windows DLL will call.

Examples

Before HpibSetWaitHook is called, a far pointer to the procedure should be created using MakeProcInstance. For example:

```
FARPROC lpfnMyAbortProc ;
:
lpfnMyAbortProc = MakeProcInstance (MyAbortProc, hInst) ;
hpiberror = HpibSetWaitHook(hHpib, select_code, lpfnMyAbortProc) ;
```

In its simplest form, the abort proc looks like this:

```
int FAR PASCAL MyAbortProc (void)
    {
    if /* you want to abort the HP-IB transaction */
        return MY_ERROR ;
    else
        return 0 ;
    }
```

If you want your abort proc to yield to windows, use the following function:

```
void FAR PASCAL PeekYield (void)
    {
    MSG msg ;
    if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
        {
        TranslateMessage ((LPMSG) &msg) ;
        DispatchMessage ((LPMSG) &msg) ;
        }
    return ;
    }
```

Comments

This function is supported with C languages only.

The error values you use should be in the range 0x20 through 0xFF. An error value of 2, for example, is the same as ESEL, and the program would think that an invalid select code was given. You can take advantage of this restriction by setting up your own timeout routines instead of using HpibTimeout(), and returning ETIME if a transaction times out.

The abort proc should be exported in your .DEF file. Or, if you are using Microsoft C 6.0, you can use the _export keyword.

Caution

If your abort proc yields to Windows, or calls any function that yields, make sure that you cannot close the application when an Hpib function is called. You should *not* be able to close your application when a .DLL function is being executed. Make sure that you do not call any functions in the HP-IB DLL if the DLL is currently yielding.

HpibSpoll

This command performs a serial poll of a specified device. It sets a variable representing the device's response byte.

Syntax

`HpibSpoll (`*hHpib*`,`*device_address*`,`*response_REF*`)`

hHpib specifies the handle returned by HpibOpen.

device_address specifies the bus address of the device to be polled.

response_REF variable into which the response byte is placed.

Examples

For Pascal:

```
VAR
 response : INTEGER;
      err : INTEGER;
    hHpib : INTEGER;

err := HpibSpoll (hHpib,723,response); {Performs a serial poll
       on device 723 and puts the response byte in RESPONSE.}
```

For C:

```
int     response;
int     error;
HANDLE  hHpib;

error = HpibSpoll(hHpib,723L,&response); /*Perform a serial poll
        on device 723 and put the response byte in response.*/
```

For Visual BASIC:

```
device& = 723
errnum% = HpibSpoll(hHpib%,device&,response%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Perform a serial poll on device 723,
'put the response byte in response%.
```

Bus Activity

- ATN is set.
- UNL is sent.
- MLA is sent.
- TAD is sent.
- OSA is sent if specified.
- SPE is sent.
- ATN is cleared.
- Poll byte is read.
- ATN is set.
- SPD is sent.
- UNT is sent.

Comments

If a device is requesting service, it stops requesting service when its response byte is read.

Some devices are not capable of responding to a serial poll, in which case polling may result in an error. Consult the instrument manual to determine if an instrument can respond to a serial poll and how its response byte is interpreted.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

HpibStatus

This command determines the current interface status regarding a particular condition. It sets a variable representing that status.

Syntax

`HpibStatus` (*hHpib*, *select_code*, *condition*, *status_REF*)

hHpib specifies the handle returned by HpibOpen.

select_code specifies the interface select code.

condition specifies the condition being checked, from 0 to 11. The possible conditions are:

Value	Description
0	Is the interface currently in the remote state? (always no)
1	What is the current state of the SRQ line?
2	What is the current state of the NDAC line?
3	Is the interface currently system controller?
4	Is the interface currently active controller?
5	Is the interface currently addressed as talker?
6	Is the interface currently addressed as listener?
7	What is the interface's bus address?
8	What is the state of the ATN line?
9	What is the address status of the interface?
10	What is on the DIO lines now?
11	What is the bus status of the interface?
12	What interface is installed?

status_REF variable into which the condition's status is placed. It can have the following values:

Conditions 0 to 6 and 8

Value	Meaning
0	Clear or No
1	Set or Yes

Condition 7

Value	Meaning
0 to 30	Address of card

Condition 9*

Bit	Value	Meaning
0	1	ulpa
1	2	TADS
2	4	LADS
3	8	TPAS
4	16	LPAS
5	32	ATN
6	64	LLO
7	128	REM

Condition 10

Value	Meaning
0 to 255	Value of the data lines on the bus

Condition 11*

Bit	Value	Meaning
0	1	REN
1	2	IFC
2	4	SRQ
3	8	EOI
4	16	NRFD
5	32	NDAC
6	64	DAV
7	128	ATN

Condition 12

Value	Meaning
0	no interface
1	HP 82990 (old) or HP 27209
2	HP 82335

* The actual value returned from conditions 9 and 11 will be the sum of the values of all true conditions. For example, the value returned if bits 2 and 3 were true would be 12.

To check whether a specific condition is true, use the AND operand in your language. For example, to check if DAV is true, you could call `HpibStatus(7L,11,&result)`, then check whether (result AND 32) = 32, then (DAV is set). Make sure you are using the binary AND in your language and not the logical AND.

Examples

For Pascal:

```
VAR
 select : INTEGER;
 status : INTEGER;
    err : INTEGER;
  hHpib : INTEGER;

 select := 1;
    err := HpibStatus (hHpib,7,select,status);
           {Determine if SRQ is set.}
```

For C:

```
int     select;
int     status;
int     error;
HANDLE  hHpib;

select = 1;
error = HpibStatus(hHpib,7L,select,&status);
        /*Determine if SRQ is set.*/
```

For Visual BASIC:

```
isc& = 7
select% = 1
errnum% = HpibStatus(hHpib%,isc&,select%,status%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
'Determine if SRQ is set.
```

Bus Activity

None.

Comments

Possible errors are NOERR, ESEL, ENOOPEN, EHANDLE, and ERANGE.

Status conditions 9 through 11 are rarely used.

HpibTakectl

This command takes active control from the currently active controller on the bus back to the controller holding the HP 82335 HP-IB interface.

Syntax

`HpibTakectl (`*hHpib*`,`*select_code*`,`*priority*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
priority	specifies the priority of the request. This parameter can take one of three values:

1	Wait until the active controller passes control back to me. It will wait until it receives control or until it times out as specified by the HpibTimeout function.
2	Assert SRQ with bits 1 and 6 set, then wait until the active controller passes control back to me. It will wait until either it receives control or until it times out as specified by the HpibTimeout function.
3	Assert the Interface Clear (IFC) line. Asserting the IFC line immediately makes the HP 82335 the active controller. The HP 82335, however, *must* be the system controller to be able to assert the IFC line. If it is not the system controller, an ECTRL error will result.

Examples

For Pascal:

```
error  : INTEGER ;
hHpib  : INTEGER ;

error := HpibTakectl(hHpib,7, 1)
if error <> NOERR then writeln('an error occurred...');
```

For C:

```
int    error ;
HANDLE hHpib ;
error = HpibTakectl(hHpib,7L, 1)
if (error != NOERR) /*Do error handling*/
```

For Visual BASIC:

```
dev& = 7 : priority% = 1
errnum% = HpibTakectl(hHpib%,dev&,priority%)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

Bus activity is controlled by the active controller until HpibTakectl is finished.

Comments

The Windows DLL defaults to address 30. If necessary, you can change this using the HpibControl command.

It may take awhile for the device that has active control to pass control back to the Windows DLL. You may want to increase your timeout value using HpibTimeout before calling HpibTakectl, and decrease it after the HpibTakectl call.

Possible errors are NOERR, ETIME, ESEL, ERANGE, ENOOPEN, EHANDLE, and ECTRL.

HpibTimeout

This command sets an interface timeout value in seconds for I/O operations that do not complete (for example, the printer runs out of paper).

The default is timeout disabled.

Syntax

`HpibTimeout (`*hHpib*`,`*select_code*`,`*timeout*`)`

hHpib	specifies the handle returned by HpibOpen.
select_code	specifies the interface select code.
timeout	specifies the length of the timeout period. A value of 0.0 disables the timeout, while a negative value results in an error.

Examples

For Pascal:

```
VAR
 timeout_val : DOUBLE;
         err : INTEGER;
       hHpib : INTEGER;

timeout_val := 1.23; {Timeout after 1.23 seconds.}
        err := HpibTimeout (hHpib,7,timeout_val);
```

For C:

```
int       error;
double    timeout_val;
HANDLE    hHpib;

timeout_val = 1.23; /*Timeout after 1.23 seconds.*/
error = HpibTimeout(hHpib,7L,timeout_val);
```

For Visual BASIC:

```
isc& = 7
timeout.val# = 2.0 'Timeout = 2 seconds.
errnum% = HpibTimeout(hHpib%,isc&,timeout.val#)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

None.

Comments

Timeout is effective for any interface operation that transfers data or commands.

A timeout error occurs only if timeout is enabled (that is, the timeout is set to a positive value).

Timeout should be established early in your program. It provides a way to recover from I/O operations that are not completed.

The timeout value is a real number specified in seconds, which gets rounded to the nearest 1/16 second. To timeout after 5 seconds, set timeout to 5.0. To timeout after 0.5 second, set timeout to 0.5. Note that a timeout of 0.0 effectively disables any timeouts. The maximum allowable timeout is 4096 seconds.

If a transfer times out, the incompleted transfer function returns the value 4, which corresponds to the ETIME error.

Possible errors are NOERR, ESEL, ENOOPEN, EHANDLE, and ERANGE.

HpibTrigger

This command triggers one or more devices.

Syntax

`HpibTrigger (`*hHpib*`,`*device_address*`)`
`HpibTrigger (`*hHpib*`,`*select_code*`)`

hHpib specifies the handle returned by HpibOpen.

device_address specifies a device address.

select_code specifies the interface select code.

Examples

For Pascal:

```
VAR
  err : INTEGER;
hHpib : INTEGER;

err := HpibTrigger (hHpib,723);
```

For C:

```
int     error;
HANDLE  hHpib;

error = HpibTrigger(hHpib,723L);
```

For Visual BASIC:

```
dev& = 723
errnum% = HpibTrigger(hHpib%,dev&)
if errnum% <> NOERR then MsgBox(HpibErrStr$(errnum%))
```

Bus Activity

If a device address is specified:

- ATN is set.
- MTA is sent.
- UNL is sent.
- LAD is sent.
- OSA is sent if specified.
- GET is sent.

If a select code is specified:

- ATN is set.
- GET is sent.

Comments

Only one device can be triggered at a time if a device address is specified.

If a select code is specified, all devices on the bus that are addressed to listen (with HpibSend, for example) are triggered.

Possible errors are NOERR, ETIME, ECTRL, ENOOPEN, EHANDLE, and ESEL.

A

Error Descriptions

This appendix describes the Windows Command Library errors.

Error Number	Mnemonic	Description
0	NOERR	No error occurred.
1	EUNKNOWN	Unknown error occurred. Check for malfunctioning equipment or incorrect hardware configuration.
2	ESEL	Invalid select code or device address was specified. This error would most likely occur under these conditions: ■ The first parameter of a call should have been a valid select code, but a device address or an invalid select code was specified instead. ■ A device address was expected, but a select code was given or a primary address outside the range 0 to 31 was specified. ■ The device address of the HP-IB interface was specified as a parameter in commands such as HpibSpoll, HpibRemote, or HpibClear.

Error Number	Mnemonic	Description
3	ERANGE*	A command parameter was specified outside its allowable range. This error can occur under these conditions: ■ HpibEntera, HpibEnters. The specified length must be positive. ■ HpibEnterab, HpibEnterb. The specified length must be positive. The swap size must be from 1 to 8. ■ HpibEol. The specified length must be from 0 to 8. ■ HpibControl. The specified value must be from 5 to 7. If 7 is selected, the valid address values are 0 to 30. ■ HpibOutputa, HpibOutputs. The specified length must be positive. ■ HpibOutputab, HpibOutputb. The specified length must be positive. The swap size must be from 1 to 8. ■ HpibPpollc. The configuration value must be from 0 to 15. ■ HpibSend. The length must be positive. ■ HpibStatus. The status specified was outside the range 0 to 12. ■ HpibTimeout. The specified timeout value must be greater than or equal to 0.
4	ETIME	The time specified by HpibTimeout has elapsed since the last byte was transferred.
5	ECTRL	The HP-IB interface must be the active controller or the system controller.

Error Number	Mnemonic	Description
6	EPASS	Obsolete.
7	ENUM	Either no digit or an improperly formed number was received during real number input using HpibEnter or HpibEntera. In this case, 0 is returned as the data value.
8	EADDR	Improper talker or listener addressing occurred. An attempt was made to input or output data when the interface was not addressed to listen or talk. This error is likely to occur if a select code was specified instead of a device address, and the interface was not properly addressed to talk or listen.
9	EFILE	A file error has occurred while reading, writing, or creating a file. This error could indicate either a disk full condition or a file does not exist for HpibOutputf.
10	EOPEN	An error occurred during a call to HpibOpen. The first parameter could be out of range, or another application may have already allocated the card.
11	ENOOPEN	A card has not been opened.
12	ECLOSE	An error occurred during a call to HpibClose.
13	EHANDLE	An invalid handle was received.

* Potential conflict for C languages. See the following paragraph.

For C languages, ERANGE is defined to a different value by the MATH.H file. If MATH.H is included by a program, the compiler gives a warning and sets ERANGE to the last value defined. In this case, change one of the define statements so the names are different. For example, you could change two lines in the HPIB.H Command Library file to

```
#define ERNGE 3
case ERNGE:    return (" Value out of range ");
```

and then use ERNGE for the Command Library error name (instead of ERANGE).

Index

F

H

Customer Reorder Number
82335-90007

Printed in U.S.A. E1291

82335-90607 Manufacturing Number