GPIB-232CT
User Manual

MicroGPIB®
RS-232 to IEEE-488 Controller

July 1992 Edition

Part Number 320114-01

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
(800) IEEE-488 (toll-free U.S. and Canada)
Technical support fax: (512) 794-5678

Branch Offices:
Australia 03 879 9422, Belgium 02 757 00 20, Canada 519 622 9310,
Denmark 45 76 73 22, France 1 48 65 33 70, Germany 089 714 5093,
Italy 02 48301892, Japan 03 3788 1921, Netherlands 01720 45761,
Norway 03 846866, Spain 91 896 0675, Sweden 08 984970,
Switzerland 056 45 58 80, U.K. 0635 523 545

MicroGPIB Products

National Instruments has developed the MicroGPIB product line to offer a series of high performance, low cost IEEE-488 support items, packaged in small all-metal cases, capable of being rack mounted.

Limited Warranty

The GPIB-232CT is warranted against defects in materials and workmanship for a period of two years from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

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

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

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against

National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

FCC/DOC Radio Frequency Interference Compliance

This equipment generates and uses radio frequency energy and, if not installed and used in strict accordance with the instructions in this manual, may cause interference to radio and television reception. This equipment has been tested and found to comply with (1) the limits for a Class A computing device, in accordance with the specifications in Subpart J of Part 15 of U.S. Federal Communications Commission (FCC) Rules, and (2) the limits for radio noise emissions from digital apparatus set out in the Radio Interference Regulations of the Canadian Department of Communication (DOC). These regulations are designed to provide reasonable protection against interference from the equipment to radio and television reception in commercial areas.

There is no guarantee that interference will not occur in a particular installation. However, the chances of interference are much less if the equipment is used according to this instruction manual.

If the equipment does cause interference to radio or television reception, which can be determined by turning the equipment on and off, one or more of the following suggestions may reduce or eliminate the problem.

- Operate the equipment and the receiver on different branches of your AC electrical system.
- Move the equipment away from the receiver with which it is interfering.
- Relocate the equipment with respect to the receiver.
- Reorient the receiver's antenna.
- Be sure that the equipment is plugged into a grounded outlet and that the grounding has not been defeated with a cheater plug.

If necessary, consult National Instruments or an experienced radio/television technician for additional suggestions. The following booklet prepared by the FCC may also be helpful: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock Number 004-000-00345-4.

Preface

The *GPIB-232CT User Manual* describes the features, functions, and operation of the Product. The GPIB-232CT is one of National Instruments family of MicroGPIB products. These products are high-performance, low-cost IEEE-488 support items, packaged in small all-metal cases, capable of being rackmounted.

Organization of the *GPIB-232CT User Manual*

The *GPIB-232CT User Manual* is organized as follows:

Chapter 1, *Description of the GPIB-232CT*, contains general information about the National Instruments GPIB-232CT, the IEEE-488 bus, and the RS-232 port. This chapter also lists all components and accessories as well as provides electrical, environmental, and physical specifications.

Chapter 2, *The GPIB-232CT Modes of Operation*, explains the two modes of operation of the GPIB-232CT.

Chapter 3, *Installation and Configuration in S Mode*, contains the S mode installation and configuration steps.

Chapter 4, *Programming in S Mode*, explains how to program the GPIB-232CT when operating in S mode.

Chapter 5, *S Mode Functions*, gives a detailed description of each S mode function. The functions are arranged in alphabetical order and each contains the syntax, purpose, and examples.

Chapter 6, *Installation and Configuration in G Mode*, contains the G mode installation and configuration steps.

Chapter 7, *Programming in G Mode*, explains how to program the GPIB-232CT when operating in G mode.

Chapter 8, *G Mode Functions*, gives a detailed description of each G mode function. The functions are arranged in alphabetical order and each contains the syntax, purpose, and examples.

Appendix A, *Multiline Interface Messages*, contains an ASCII chart, and a list of the corresponding GPIB messages.

Appendix B, *Status Information*, contains explanations of status bits, GPIB error codes, and serial port error codes.

Appendix C, *The Serial Connection*, explains the RS-232 serial standard and contains information for building a serial cable.

Appendix D, *Operation of the GPIB*, describes the operation of the GPIB.

Appendix E, *Common Questions*, provides answers to common questions that frequently arise.

Appendix F, *Parallel Polling*, explains the use and operation of Parallel Polls.

Appendix G, *Setting Switches*, gives additional details about setting switches on the GPIB-232CT.

Appendix H, *Sample Programs*, provides some S mode and G mode sample programs.

Conventions Used in This Manual

Throughout this manual, the following conventions are used to distinguish elements of text:

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, syntax examples, and names of variables.
`boldface`	Bold text in this manual denotes a signal name, a front panel LED indicator, or onscreen module responses.

<>	Angle brackets enclose the ASCII character symbols for certain keystrokes. For example, <CR> for carriage return and <LF> for linefeed.

Abbreviations

The following abbreviations are units of measure that are used in the text of this manual.

A	ampere
C	Celsius
°	degree
>	greater than
•	greater than or equal to
hex	hexadecimal
Hz	hertz
in.	inch
K	1,024
kbytes/sec	1000 bytes per second
<	less than
•	less than or equal to
m	meter
M	1,048,576
Mbytes/sec	1,000,000 bytes per second
mA	milliampere
mm	millimeter
oz.	ounce
%	percent
sec	second
V	volt
VAC	volts alternating current
VDC	volts direct current

Customer Communication

We appreciate communicating with the people who use our products. If you encounter any technical problems, you can use the following toll-free number between the hours of 8:00 a.m. and 5:30 p.m. (central time) to reach the National Instruments applications engineering department:

(512) 794-0100
(800) IEEE-488 (toll-free U.S. and Canada)

For your documentation comments, we have included a *User Comment Form* at the back of the manual. Please mail it to the address printed at the bottom of the form.

Contents

Figures

Tables

Chapter 1
Description of the GPIB-232CT

Introduction

The GPIB-232CT, shown in Figure 1-1, is a high-performance serial-to-GPIB interface. It provides a computer with an RS-232 port, a means of controlling, talking, and listening on the GPIB. The GPIB-232CT is also capable of interfacing RS-232 instruments and peripherals to the GPIB.

The GPIB-232CT has all the software and logic required to implement the physical and electrical specifications of the IEEE-488 and the RS-232 standards. It is capable of both interpreting and executing high-level commands that you send to it over the serial port, and performing GPIB-to-RS-232 protocol conversion.

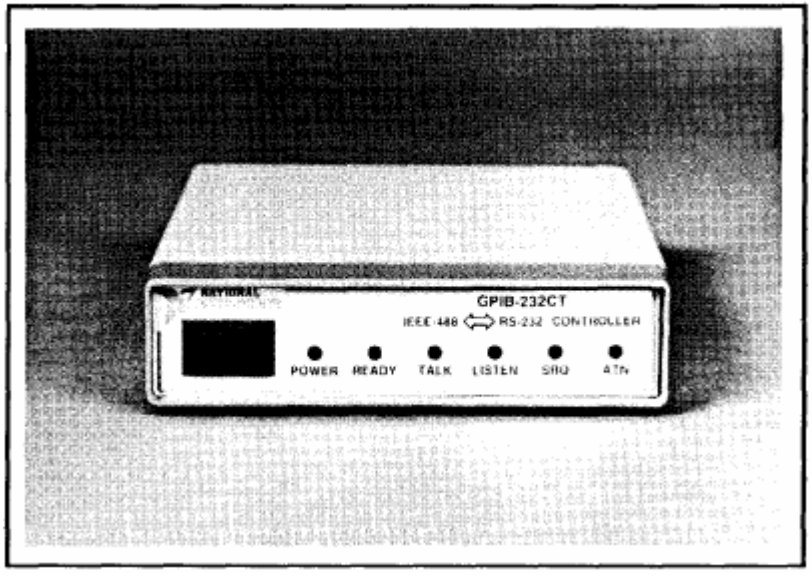


Figure 1-1. The GPIB-232CT

What Your Kit Contains

Your kit should contain the following components:

Component	Part Number
One of the following boxes: • GPIB-232CT (64K RAM - 115 VAC) • GPIB-232CT (256K RAM - 115 VAC) • GPIB-232CT (64K RAM - 230 VAC) • GPIB-232CT (256K RAM - 230 VAC)	 776173-01 776173-02 776173-31 776173-32
GPIB-232CT User Manual	320114-01
GPIB-CT IBCL Reference Manual	320132-01

Optional Equipment

Component	Part Number
Rackmount Kit:	
Single (1 unit)	180480-01
Dual (2 units)	180480-02
RS-232 Shielded Cables, Compatible with IBM PC	
DTE to DTE - 2 m	180459-02
DTE to DTE - 4 m	181459-04
Double-Shielded GPIB Cables:	
GPIB Type X2 Cable – 1 m	763061-01
GPIB Type X2 Cable – 2 m	763061-02
GPIB Type X2 Cable – 4 m	763061-03

GPIB-232CT Specifications

Tables 1-1 through 1-3 specify the electrical, environmental, and physical characteristics of the GPIB-232CT.

Table 1-1. Electrical Characteristics

Characteristic	**Specification**
Power Supply Unit	Wall mount type, 115 VAC or 230 VAC, 50/60 Hz input, 9 VDC @ 1A max output
Voltage	9 VDC unregulated
Current	640 mA typical; 1,500 mA max

Table 1-2. Environmental Characteristics

Characteristic	**Specification**
Operating Temperature	0° to 50° C
Storage Temperature	0° to 70° C
Relative Humidity	10% to 95% noncondensing conditions
Noise Emissions	FCC Class A Verified

Table 1-3. Physical Characteristics

Characteristic	**Specification**
Case Size	1.6 in. by 5.7 in. by 8.4 in. (40.6 mm by 144.8 mm by 213.4 mm)
Case Material	All metal enclosure
Rack Mounting	Single or dual kits available
Weight	28 oz. (without power supply unit)

The GPIB-232CT Front Panel

The front panel of the GPIB-232CT is shown in Figure 1-2. The power switch and six Light Emitting Diodes (LEDs) are mounted on the GPIB-232CT front panel.

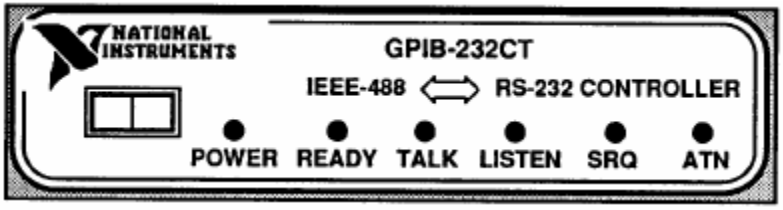


Figure 1-2. The GPIB-232CT Front Panel

The LEDs show the current status of the GPIB-232CT at all times. Table 1-4 describes each LED.

Table 1-4. LED Descriptions

LED	Indication
POWER	Indicates that power to the unit has been applied and the ON/OFF switch is in the ON position.
READY	Indicates that the power-on self-test has passed successfully and the unit is ready to operate.
TALK	Indicates that the GPIB-232CT is configured as a GPIB Talker.
LISTEN	Indicates that the GPIB-232CT is configured as a GPIB Listener.
SRQ	Indicates that the GPIB signal line SRQ* is asserted (low).
ATN	Indicates that the GPIB signal line ATN* is asserted (low).

Note: * indicates that the signal is active low (negative logic).

The GPIB-232CT Rear Panel

The rear panel of the GPIB-232CT is shown in Figure 1-3. The power cable, RS-232 cable, and GPIB cable are shown connected to the rear panel of the GPIB-232CT.

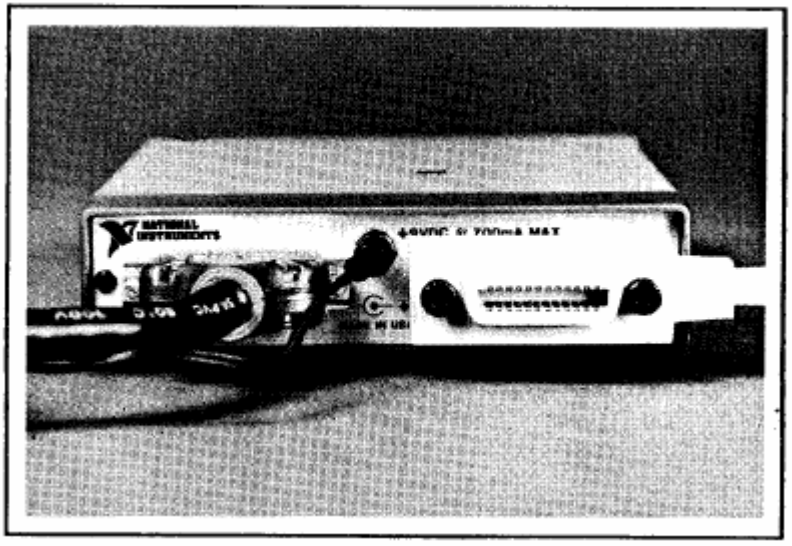


Figure 1-3. The GPIB-232CT Rear Panel

The RS-232 Port

The RS-232 port on the GPIB-232CT is configured as a DTE (Data Terminal Equipment) and uses a standard 25-pin shielded D-subminiature female connector with screwlock assemblies. The RS-232 connector will accept standard 25-pin D-subminiature male connectors. A diagram of the serial connector and the signals supported is shown in Figure 1-4. For more information on the RS-232 signals, refer to Appendix C, *The Serial Connection*.

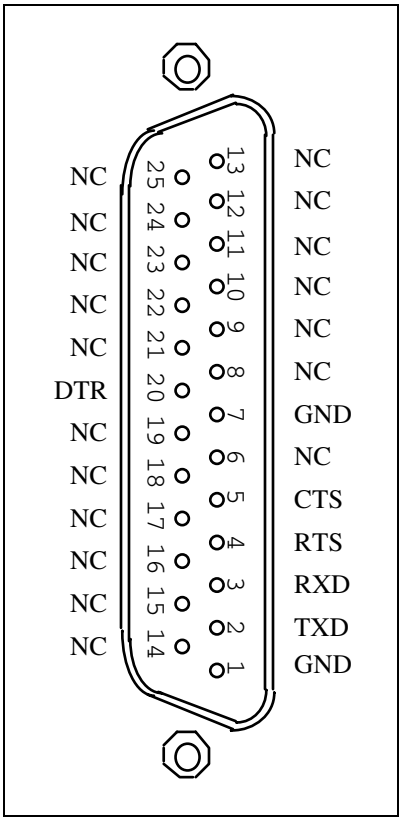


Figure 1-4. The RS-232 Connector and Signal Designations

The GPIB Port

The GPIB connector is a standard 24-pin shielded AMP Champ female connector with metric screwlock hardware. A diagram of the GPIB connector and the signals supported is shown in Figure 1-5 (a * suffix indicates that the signal is active low).

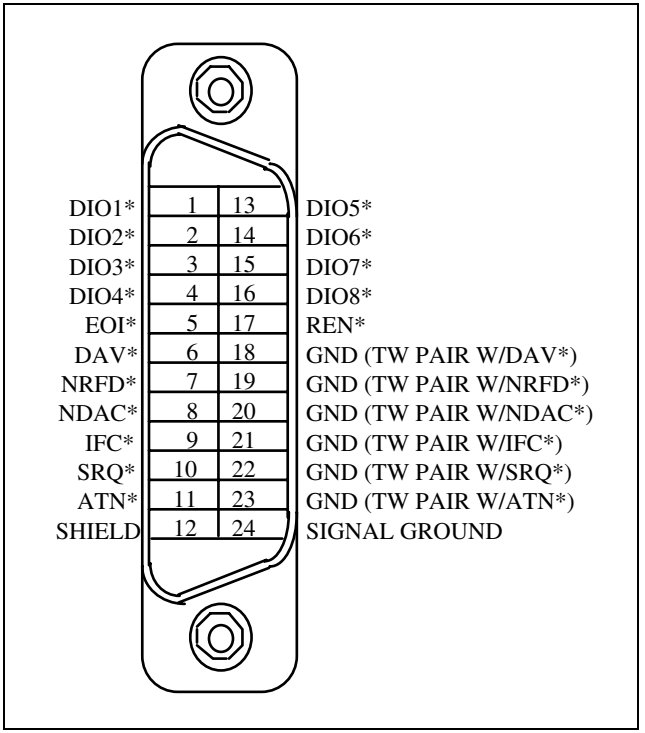


Figure 1-5. The GPIB Connector and Signal Designations

Chapter 2
The GPIB-232CT Modes of Operation

Introduction

The GPIB-232CT is capable of operating in one of two modes: **S mode** or **G mode**. This chapter helps you determine the mode you will use and the chapters of the manual that apply to that mode.

Choosing Between S Mode or G Mode

You will be connecting the GPIB-232CT to both a serial device and a GPIB device. The GPIB-232CT's mode of operation is determined by the role the serial device plays in your setup.

Is the serial device a Controller? Is it going to manage the GPIB, address devices, and perform other GPIB Controller functions? If so, use the GPIB-232CT in the **S mode**.

Or, is the serial device going to be only a Talker and/or Listener, where a GPIB device(s) will manage the bus and send and receive data to and from the serial device? If so, use the GPIB-232CT in the **G mode**.

Figure 2-1 shows the GPIB-232CT operating in the **S mode** where a personal computer is controlling a GPIB plotter.

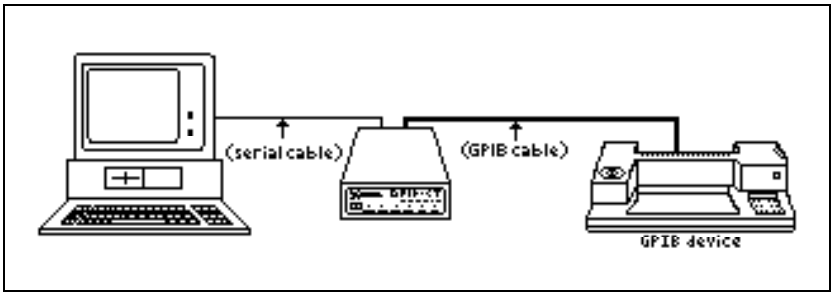


Figure 2-1. Personal Computer Controlling a GPIB Plotter

Figure 2-2 shows the GPIB-232CT operating in the **G mode** where the GPIB-232CT enables a serial printer to be programmed from a GPIB Controller.

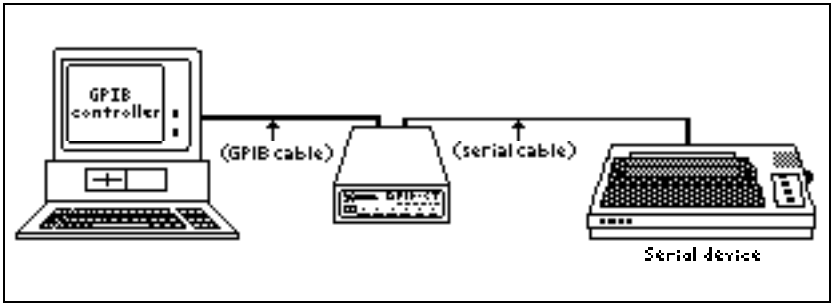


Figure 2-2. A Serial Printer Connected to a GPIB Controller

Now, turn to Chapter 3 for **S mode** installation and configuration, then to Chapters 4 and 5 to begin programming your GPIB-232CT in **S mode**.

Or, turn to Chapter 6 for **G mode** installation and configuration, then to Chapters 7 and 8 to begin programming your GPIB-232CT in **G mode**.

Chapter 3
Installation and Configuration in S Mode

If you plan to operate in S mode, use this chapter to install and configure the GPIB-232CT. Then read Chapters 4 and 5 to learn about its programming messages.

Installation

There are five basic steps to installing the GPIB-232CT.

1. Inspect the GPIB-232CT for damage that may have been caused in shipment.
2. Verify the voltage requirement.
3. Configure the operating parameters.
4. Connect the cables.
5. Power on the unit.

These steps are described in more detail in the following sections.

Step 1: Inspection

Before you install the GPIB-232CT, inspect the shipping container and its contents for damage. If damage appears to have been caused in shipment, file a claim with the carrier. Retain the packing material for possible inspection and/or for reshipment.

If the equipment appears to be damaged, do not attempt to operate it. Contact National Instruments for instructions.

Step 2: Verify the Voltage Requirement

The GPIB-232CT is shipped from the factory with either a 115V or 230V wall-mount supply. Verify that the voltage on the supply matches the voltage that is supplied in your area.

Caution: Operating the unit at any voltage other than the one specified could damage the unit.

Step 3: Configure the Operating Parameters

The GPIB-232CT is shipped from the factory configured to operate in S mode. The serial port is configured at 9600 baud, 1 stop bit, no parity, and 7 data bits. If you wish to change any of the GPIB-232CT parameters, you must open the unit and set the configuration switches. To change the configuration switches follow these steps:

1. Disconnect power to the unit and disconnect any cables that may be connected to the unit.

2. Unscrew the two screws on the opposite sides of the rear panel.

3. Grab the rear panel bezel and pull it straight away from the rest of the unit. The card should slide out the back of the enclosure.

4. Locate the configuration DIP switch (U22) on the printed wire board.

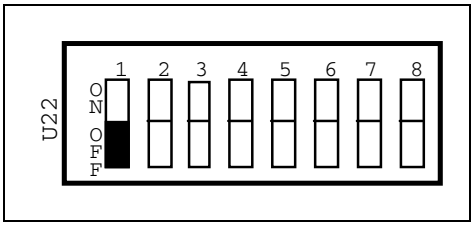
5. Set the switches for the desired mode of operation. Refer to the following section, *Set Configuration Switches*.

   Caution: Most of the circuitry in the GPIB-232CT uses advanced CMOS technology and can be damaged by static electricity. Avoid touching any of the components and take any necessary CMOS handling precautions.

6. Close the unit and re-insert the screws removed in Step 2.

Set Configuration Switches

The DIP switch at location U22 on the printed wire board is used to configure the serial port parameters of the GPIB-232CT while in S mode. The DIP switch, shown in Figure 3-1, has eight configuration switches.



ON indicates that the switch is pressed on the "ON" side.

OFF indicates that the switch is pressed on the "OFF" side.

Figure 3-1. S Mode Switch Settings

Tables 3-1 and 3-2 show the possible configurations of the eight switches and what the configurations indicate.

Table 3-1. Configuration Parameters for Switches 1 through 5

Switch	Position	Indication
1	OFF	Configures the GPIB-232CT to operate in S mode.
	ON	Configures the GPIB-232CT to operate in G mode.
2	OFF	7 bits/character.
	ON	8 bits/character.
3	OFF	1 stop bit/character.
	ON	2 stop bits/character.
4	OFF	Parity generation/checking disabled.
	ON	Parity generation/checking enabled.
5	OFF	Odd parity.
	ON	Even parity.

Table 3-2. Configuration Parameters for Switches 6 through 8

Switches 6	7	8	Indication
OFF	OFF	OFF	300 baud
OFF	OFF	ON	600 baud
OFF	ON	OFF	1200 baud
OFF	ON	ON	2400 baud
ON	OFF	OFF	4800 baud
ON	OFF	ON	9600 baud
ON	ON	OFF	19200 baud
ON	ON	ON	38400 baud

Default Settings for Configuration Switch

Figure 3-2 shows the GPIB-232CT's factory default setting for switch U22. Switch 1 is OFF; this indicates that the GPIB-232CT will be operating in S mode. Switch 2 is OFF; this indicates that the GPIB-232CT will be using 7 bits/character for its serial data transfers. Switch 3 is OFF; this indicates 1 stop bit per character. Switches 4 and 5 are both OFF; this indicates that parity is disabled. Switches 6 through 8 are in the ON OFF ON positions, respectively, indicating that the serial port will be operating at 9600 baud.

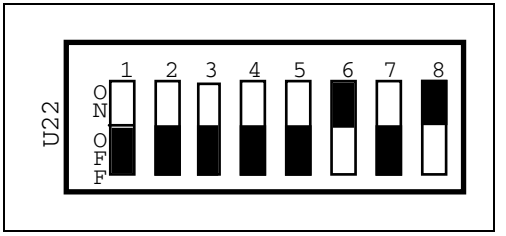


Figure 3-2. Factory Default Settings for Switch U22

Note: Switch U20 is not used by S or G mode, but can be used for user applications in IBCL mode.

Step 4: Connect the Cables

Connect the cables as follows:

1. Connect the serial cable to the GPIB-232CT and securely fasten the holding screws. Connect the other end of the cable to your serial device. Be sure to use only shielded serial cable, and obey all RS-232 cabling restrictions.

2. Connect the GPIB cable to the GPIB-232CT and tighten the thumb screws on the connector. Connect the other end to your GPIB system. Be sure to obey all IEEE-488 cabling restrictions, and use only double-shielded GPIB cable.

3. Connect the power jack of the wall-mount power supply to the power receptacle on the back panel of the GPIB-232CT, then plug the supply into an AC outlet of the correct voltage.

Step 5: Power on the Unit

Power on your GPIB-232CT by using the front panel rocker switch. The **POWER** LED should come on immediately. The **READY** indicator should come on after the GPIB-232CT has passed its power-on self-test, indicating the unit is ready for operation.

If the **READY** indicator does not come on within seven seconds after the unit is powered on, recheck all connections and switch settings and retry the power-on sequence. If the **READY** light still fails to come on, contact National Instruments for further instructions.

Chapter 4
Programming in S Mode

This chapter describes how to program the GPIB-232CT in S mode using programming messages and data strings. It describes programming messages, their format, and how they are processed, along with the functions and function arguments that make up the programming messages.

Programming Messages

You program the GPIB-232CT by sending it programming messages (which are ASCII strings) and data strings by way of its serial port.

Programming Message Format

A programming message consists of a function name, one or more arguments (optional), followed by a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>).

You can enter programming messages in any combination of uppercase and lowercase letters.

Example of a Programming Message

The following line of BASIC code:

```
PRINT #1,"clr 3,4"
```

contains the function name `clr` and the arguments `3` and `4`. This programming message tells the GPIB-232CT to clear the devices at GPIB addresses 3 and 4. `PRINT #1` is the BASIC command to send characters to the serial port after the serial port has been opened with the `"OPEN COM..."` statement. In this example, BASIC automatically sends a <CR>, so it is not necessary to include it here.

Example of a Programming Message with Data String

The following lines of BASIC code:

```
PRINT #1, "wrt 2"
PRINT #1, "IN;CI;"
```

contain the function name `wrt`, the argument `2`, and the data string `"IN;CI;"`. This programming message is telling the GPIB-232CT to write to the device at primary address 2.

`"IN;CI;"` is the data string that contains the data `wrt` will send out on the GPIB. In this case, a <CR> is automatically sent by BASIC following each print string, so, again, it is not necessary to include it here.

The `cmd` and `wrt` programming messages are followed by a data string that can contain 7- or 8-bit data.

How Messages are Processed

The GPIB-232CT processes a programming message on a line-by-line basis. The GPIB-232CT buffers the entire message, interprets the function name and arguments, then executes the message.

The data portions of the `wrt` and `cmd` functions are not processed on a line-by-line basis. The data immediately following a wrt and a cmd are sent directly to the GPIB.

The GPIB-232CT recognizes <CTRL> h (hex 8) in a programming message as a backspace and erases the previous character. The GPIB-232CT recognizes <CTRL> h in a data string as a data byte and does not erase the previous character.

Function Names

The function names have been selected to indicate each function's purpose, thereby making your programs easy to understand. However, if you wish to reduce some overhead in your program and do not mind giving up these advantages, you can use only as much of the function name as is necessary to distinguish it from other functions. This abbreviated form of the function

name is shown in **boldface** in the function tables and in the syntax portions of the function descriptions.

Function Argument Delimiters

When you type in a function, separate the first argument from the function name with at least one space. Separate each additional argument with at least one space or a comma.

In the syntax portions of the function descriptions in Chapter 5, the information within the square brackets (`[]`) are optional. If you want to include optional information, do not type the brackets, only the information inside the brackets.

Abbreviations for Arguments

The function descriptions in Chapter 5 use abbreviations for some arguments. They are as follows:

`addr`	a GPIB address	
`alist`	one or more `addrs`	
`bool`	a boolean value:	1 = true, on, or enable
		0 = false, off, or disable

GPIB Address

Each device on the GPIB has a GPIB address. The GPIB-232CT's address is 0 at power on and can be changed using the `caddr` function. Refer to the manuals of your GPIB devices to learn their addresses. You will need to know these when you begin to program the GPIB-232CT.

Only the lower five bits of each GPIB address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary address. For example, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. A plus sign (+) separates the primary address from the secondary address. The listen address is 32 (primary address plus 32), the

talk address is 64 (primary address plus 64), and the secondary address is 2 or 98, which are equivalent. The next paragraph explains the `\x` notation.

`0+2` or `0+98` or `32+98` or `0+\x62`

Numeric String Argument

Another type of argument is a numeric string. A numeric string represents an integer, which you can express using decimal, octal, or hexadecimal digits. To specify an octal integer, precede it with a backslash (`\`). To specify a hexadecimal integer, precede it with a backslash x (`\x`) or backslash X (`\X`).

Each of the following numeric strings represents the decimal integer value 112:

```
 112
\160
\x70
```

The GPIB address argument described in the preceding discussion, *GPIB Address*, consisted of one or two numeric strings.

Status Information

The function descriptions in Chapter 5 explain that the GPIB-232CT *records* specific status and error information. This means that it stores that information in its memory so that the status information is available to you when you request it.

The function descriptions also explain that the GPIB-232CT *returns* to you certain information. This means that the GPIB-232CT sends information to you over the serial port. You then read this information from your serial port.

Serial Port Error Handling

The GPIB-232CT continuously monitors the serial port for transmission errors. If it encounters an error in the serial data, the GPIB-232CT records the error. You can program the GPIB-232CT to ignore serial port errors using the `spign` function.

GPIB Read and Write Termination Method (END and EOS)

You program the GPIB-232CT to Talk in order to send data messages over the GPIB, and to Listen in order to receive data messages from the GPIB.

The IEEE-488 specification defines two ways that GPIB Talkers and Listeners can identify the last byte of data messages—`END` and `EOS`. The two methods permit a Talker to send data messages of any length without the Listener(s) knowing in advance the number of bytes in the transmission.

`END` message	The Talker asserts the EOI (End Or Identify) signal while the last data byte is being transmitted. The Listener stops reading when it detects a data byte accompanied by EOI.
`EOS` character	The Talker sends an EOS (end-of-string) character at the end of its data string. The Listener stops receiving data when it detects the EOS character. Either a 7-bit ASCII character or a full 8-bit binary byte can be used.

The two methods can be used individually or in combination. It is important that the Listener be configured to detect the end of a transmission.

The GPIB-232CT always terminates GPIB `rd` operations on the `END` message. Using the `eos` and `eot` functions, you can change the other default GPIB read and write termination methods.

S Mode - Default Settings

Tables 4-1 and 4-2 list power-on characteristics of the GPIB-232CT and the functions you can use to change those characteristics.

Table 4-1. Serial Port Characteristics

Characteristic	Power-on Value	Function
Echo bytes to serial port	no	`echo`
Ignore serial port errors	yes	`spign`
Send XON/XOFF	no	`xon`
Recognize XON/XOFF	no	`xon`

Table 4-2. GPIB Characteristics

Characteristic	Power-on Value	Function
Primary/secondary address	pad=0, sad=none	`caddr`
End-of-string modes	none	`eos`
Send `END` on writes	yes	`eot`
IST bit setting	0	`ist`
GPIB-232CT is System Controller	yes	`rsc`
I/O timeout	10 sec	`tmo`
Serial poll timeout	.1 sec	`tmo`

List of S Mode Functions by Group

The GPIB-232CT functions are divided into three main groups: GPIB functions, Serial Port functions, and General Use functions.

GPIB Functions

The GPIB functions are divided into subgroups as shown in the following tables. The subgroups are listed with the most frequently used groups first. Often, the I/O and bus management functions are the only ones you need.

Table 4-3. I/O Functions

Function	Purpose
RD count, address	Read data
WRT count, address list data	Write data

Table 4-4. Bus Management Functions

Function	Purpose
CLR address list	Clear specified device(s)
LOC address list	Go to Local
TRG address list	Trigger selected device(s)

Table 4-5. GPIB Initialization Functions

Function	Purpose
CADDR address	Change the IEEE-488 address of the GPIB-232CT
EOS modes, eoschar	Change or disable GPIB end-of-string termination mode
EOT on/off	Enable or disable END termination message on GPIB write operations
ONL on/off	Place the GPIB-232CT online/ offline
RSC on/off	Request System Control
TMO values	Change or disable time limits

Table 4-6. Serial Poll Functions

Function	Purpose
RSP address list	Conduct (request) a serial poll
RSV status byte	Request service and/or set or change the serial poll status byte

Table 4-7. Low-Level Controller Functions

Function	Purpose
CAC mode	Become Active Controller
CMD count commands	Send IEEE-488 commands
GTS mode	Go from Active Controller to Standby
PCT address	Pass Control
SIC time	Send interface clear
SRE on/off	Set/clear remote enable

Table 4-8. Parallel Poll Functions

Function	Purpose
IST on/off	Set or clear individual status bit for use in GPIB-232CT response to Parallel Polls
PPC values	Parallel Poll Configure
PPU address list	Parallel Poll Unconfigure
RPP	Conduct (request) a Parallel Poll

Serial Port Functions

Table 4-9. Serial Port Initialization Functions

Function	Purpose
ECHO on/off	Echo characters received from serial port
SPIGN on/off	Ignore serial port errors
XON modes	Change serial port XON/XOFF protocol

General Use Functions

Table 4-10. General Use Functions

Function	Purpose
IBCL	Switch to IBCL operating system
ID	Identify system
STAT modes	Return GPIB-232CT status
WAIT mask	Wait for selected event(s)

Alphabetical List of S Mode Functions

The following is an alphabetical list of all functions.

Table 4-11. GPIB-232CT Functions

Function	Purpose
CAC mode	Become Active Controller
CADDR address	Change the IEEE-488 address of the GPIB-232CT
CLR address list	Clear specified device(s)
CMD count commands	Send IEEE-488 commands
ECHO on/off	Echo characters received from serial port
EOS modes, eoschar	Change or disable GPIB end-of-string termination mode
EOT on/off	Enable or disable END termination message on GPIB write operations
GTS mode	Go from Active Controller to Standby
IBCL	Switch to IBCL operating system
ID	Identify system
IST set/clear	Set or clear individual status bit for use in GPIB-232CT response to Parallel Polls
LOC address list	Go to Local
ONL on/off	Place the GPIB-232CT online/offline

(continues)

Table 4-11. GPIB-232CT Functions (continued)

Function	Purpose
PCT address	Pass Control
PPC values	Parallel Poll Configure
PPU address list	Parallel Poll Unconfigure
RD count, address	Read data
RPP	Conduct (request) a Parallel Poll
RSC on/off	Request System Control
RSP address list	Conduct (request) a serial poll
RSV status byte	Request service and/or set or change the serial poll status byte
SIC time	Send interface clear
SPIGN on/off	Ignore serial port errors
SRE on/off	Set remote enable
STAT modes	Return GPIB-232CT status
TMO values	Change or disable time limits
TRG address list	Trigger selected device(s)
WAIT mask	Wait for selected event(s)
WRT count, address list data	Write data
XON modes	Change serial port XON/XOFF protocol

Chapter 5
S Mode Functions

This chapter contains descriptions of S mode functions that you use to program the GPIB-232CT. These functions are in alphabetical order and are formatted to provide you an easily usable reference.

Points to Remember

- The programming examples for each function description are in Microsoft BASIC Version 3.0.

- In the syntax portion of the function descriptions, arguments enclosed in square brackets ([]) are optional. Do not enter the brackets as part of your argument.

- Terminate each programming message with a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>). The terminator is denoted by a <CR> in the syntax portions of the function descriptions. In the programming examples, the BASIC `PRINT #` statement automatically sends a carriage return at the end of the string, so a carriage return is not placed there explicitly.

- To send more than one programming message per `PRINT` statement, embed a <CR> (denoted by `CHR$(13)`) or a <LF> (denoted by `CHR$(10)`) in the statement. For example, to send the two programming messages "send interface clear" (`SIC`) and "send remote enable" (`SRE`), you could use either of these two sequences:

```
PRINT #1,"sic"
PRINT #1,"sre 1"
```

  or

```
PRINT #1,"sic"+CHR$(13)+"sre 1"
```

- For all examples, the communications port has been assigned to file number 1 (`#1`) by the BASIC `OPEN "COM..."` statement.

- The I/O and bus management functions should meet most of your needs. In the descriptions that follow, these functions are marked with an asterisk (*). These are the most frequently used functions.

- It is only necessary for you to send enough characters of the function name to distinguish it from other functions. Those characters are shown in **boldface** in the syntax portion of each function description.

cac - Become Active Controller

Type: Low-level Controller function

Syntax: **cac** [bool]<CR>

Purpose: You use cac to change the GPIB-232CT from Standby Controller to Active Controller when the I/O and bus management functions do not meet the needs of your device. cac gives you more precise control over the GPIB than the I/O and bus management functions.

Remarks: If the argument bool is 0, the GPIB-232CT takes control immediately; that is, it takes control asynchronously. If the argument bool is 1, the GPIB-232CT takes control after any handshake that is in progress completes; that is, it takes control synchronously.

If you call cac without an argument, the GPIB-232CT returns to you the current Controller status, which is 0 if the GPIB-232CT is not the Active Controller and 1 if the GPIB-232CT is the Active Controller.

If you call cac with an argument and the GPIB-232CT is not CIC, the GPIB-232CT records the ECIC error.

The power-on Controller status of the GPIB-232CT is Idle Controller.

See Also: gts and sic.

Examples:

```
1. PRINT #1,"cac 0"  'Take control immediately.

2. PRINT #1,"cac 1"  'Take control synchronously.

3. PRINT #1,"CAC"    'Are we the Active Controller?

   response: 1<CR><LF>           (...yes, we're CAC)
```

caddr - Change the GPIB Address of the GPIB-232CT

Type: Initialization function

Syntax: **cad**dr [addr]<CR>

Purpose: You use caddr at the beginning of your program to change the GPIB address of the GPIB-232CT.

Remarks: The argument addr is a device address that specifies the new GPIB address for the GPIB-232CT. addr consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+). Both addresses are expressed as numeric strings.

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

If you specify a primary address without a secondary address, secondary addressing is disabled.

If you call caddr without an argument, the GPIB-232CT returns to you its current GPIB address.

The address assigned by this function remains in effect until you call caddr again, call onl, or you turn off the GPIB-232CT.

The power-on default is 0 with secondary addressing disabled.

Examples:

```
1. PRINT #1,"caddr 0+22"     'Give GPIB-232CT a
                             'primary address of 0
                             'and a secondary
                             'address of 22.

2. PRINT #1,"CADDR 1"        'Change GPIB-232CT
                             'primary address to 1
                             'and disable secondary
                             'addressing.

3. PRINT #1,"CADDR"          'Return current GPIB-
                             '232CT address.
```

response: 1<CR><LF>

clr - Clear Specified Device *

Type: Bus Management function

Syntax: **`cl`**`r [alist]`<CR>

Purpose: You use `clr` to reset the internal or device functions of the specified devices. For example, a multimeter might require that you send it either the GPIB Device Clear or Selected Device Clear command to change its function, range, and trigger mode back to its default setting. Use `clr` to do this.

Remarks: The argument `alist` is a list of `addrs` separated by commas or spaces. `addrs` are device addresses that specify the GPIB addresses you wish to clear.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

`0+2` or `0+98` or `32+98` or `0+\x62`

If you call `clr` with `alist`, the GPIB-232CT clears only the devices specified in `alist` (Selected Device Clear).

If you call `clr` without `alist`, the GPIB-232CT clears all devices (Device Clear).

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

See Also: Appendix D for more information on clearing devices, and Appendix B for more error information.

Examples:

1.
```
PRINT #1,"clr 14+30,16+12,18,3+26,6"
                        'Selectively clear 5
                        'devices.
```

2.
```
PRINT #1,"CLR"          'Issue Device Clear to all
                        'devices.
```

cmd - Send GPIB Commands

Type: Specialized Controller function

Syntax: **cm**d [#count]<CR>
commands<CR>

Purpose: You use cmd when the I/O and bus management functions do not meet the needs of your device. cmd gives you precise control over the GPIB. For example, in applications that require command sequences not sent by other functions, you can use cmd to transmit any sequence of interface messages (commands) over the GPIB.

Remarks: The argument count is a numeric string preceded by a number sign (#). #count specifies the number of GPIB command bytes (interface messages) to send, which is a number between 1 and 255. The number of command bytes must not include the carriage return (<CR>) or linefeed (<LF>) that you include to indicate the end of the programming message.

The argument commands is a list of GPIB commands. These commands are represented by their ASCII character equivalents. For example, the GPIB Untalk (UNT) command is the ASCII character underscore (_).

If you call cmd without #count, the GPIB-232CT recognizes the end of the command string when it sees a <CR> or an <LF>. #count is required only if the command string contains a <CR> or an <LF> character. However, a <CR> or an <LF> in the command string would be unusual since neither of these is a defined GPIB command.

The GPIB commands, or interface messages, are listed in Appendix A. They include device talk and listen addresses, secondary addresses, messages, device clear and trigger instructions, and other management messages.

Do not use cmd to send programming instructions to devices. Use rd and wrt to send or receive device programming instructions and other device dependent information.

The `cmd` operation terminates when:

- The GPIB-232CT successfully transfers all commands.
- The GPIB-232CT detects an error (GPIB-232CT is not CIC).
- The I/O time limit is exceeded.
- The Take Control (TCT) command is in your command string and is sent to the GPIB.
- The Interface Clear (IFC) message is received from the System Controller (not the GPIB-232CT).

After `cmd` terminates, the GPIB-232CT records the number of command bytes it actually sent. If one of the events described above occurs, the count may be less than expected.

If you specify `#count` and enter more than `#count` command bytes, the excess command bytes up to the <CR><LF> are discarded.

If you call `cmd` and the GPIB-232CT is not CIC, the GPIB-232CT records the ECIC error.

If the GPIB-232CT is CIC but not Active Controller, it takes control and asserts ATN before sending the command bytes. It remains Active Controller afterward.

See Also: Appendix A to convert hex values to ASCII characters.

Examples:

```
1. PRINT #1,"CMD"          'Program device at address
                           '11 to listen and GPIB-
                           '232CT at address 0 to
                           'talk.
   PRINT #1,"+@"           'Device listen address is
                           '43 or ASCII + and GPIB-
                           '232CT talk address is 64
                           'or ASCII @.
   PRINT #1,"WRT"          'Write the string "ABCDE"
   PRINT #1,"ABCDE"        'to device at address 11.

2. PRINT #1,"cmd"+CHR$(13)+"_?W"+CHR$(9)
                           'Pass control to device 23
                           '(CHR$(9)=TCT command).
```

echo - Echo Characters Received from Serial Port

Type: Serial Port function

Syntax: **ec**ho [bool]<CR>

Purpose: You use echo when a terminal is connected to the GPIB-232CT and you wish to display what you type on the screen of the terminal.

Remarks: If the argument bool is 1, characters received from the serial port are echoed back to the serial port. If the argument bool is 0, characters are not echoed. If the argument bool is 1 and echoing was previously disabled, characters will not be echoed until this command has been completely processed, that is, the *next* programming message will be echoed.

If you call echo without an argument, the GPIB-232CT returns the current setting.

Examples:

The following examples show commands as you would enter them at a terminal.

1. echo 1<CR> 'Turn on character echoing.

2. ECHO 0<CR> 'Disable character echoing.

3. echo<CR> 'What is the current echo status?

   response: 0<CR><LF> (character echo is

eos - Change/Disable GPIB EOS Termination Mode

Type: Initialization function

Syntax: **eos** [[R][X][B] eoschar]<CR>

or

eos D<CR>

Purpose: You use eos at the beginning of your program if you wish to use an eos mode when you transfer data to and from the GPIB. eos tells the GPIB-232CT when to stop reading information from the GPIB. eos also enables the GPIB-232CT to tell other devices that it is finished writing information to the GPIB. eos defines a specific character, end-of-string (EOS), to be recognized as a string terminator.

Remarks: The arguments R, X, B, and D specify GPIB termination methods. They enable or disable the corresponding eos mode. If a particular letter is specified, the corresponding eos mode is enabled. If it is not specified, the corresponding eos mode is disabled.

eoschar is a numeric string that represents a single ASCII character. For example, 10 represents the ASCII linefeed character.

Table 5-1. Data Transfer Termination Methods

Description	Letter
REOS - terminate read when EOS is detected.	R
XEOS - set EOI with EOS on write functions.	X
BIN - compare all 8 bits of EOS byte rather than low 7 bits (all read and write functions).	B
DISABLE - disable all eos modes.	D

Methods `R` and `B` determine how GPIB read operations performed by the GPIB-232CT terminate. If Method `R` alone is chosen, reads terminate when the low seven bits of the byte that is read match the low seven bits of the EOS character. If Methods `R` and `B` are chosen, a full 8-bit comparison is used.

Methods `X` and `B` together determine when GPIB write operations performed by the GPIB-232CT send the END message. If Method `X` alone is chosen, the END message is sent automatically with the EOS byte when the low seven bits of that byte match the low seven bits of the EOS character. If Methods `X` and `B` are chosen, a full 8-bit comparison is used.

Note: Defining an EOS byte for the GPIB-232CT does not cause the GPIB-232CT to insert that byte into the data string when performing GPIB writes. To send the EOS byte, you must include it in the data string that you send following the `wrt` programming message.

By default, no `eos` modes are enabled.

If you call `eos` with B alone as an argument, the GPIB-232CT records the EARG error.

If you call `eos` without an argument, the GPIB-232CT returns to you the current `eos` settings.

The assignment made by this function remains in effect until you call `eos` again, call `onl`, or you turn off the GPIB-232CT.

See Also: Chapter 4, the section entitled *GPIB Read and Write Termination Method*.

Examples:

```
1. PRINT #1,"eos R,B,10" 'Terminate read when <LF>
                         'is detected; compare all
                         '8 bits; do not send EOI
                         'with <LF>.
   PRINT #1,"rd 10 5"    'Read 10 bytes from device
                         '5 into serial port
                         'buffer.
   RESP$=INPUT$(10,#1)   'Input 10 bytes from
                         'serial port buffer.
   LINE INPUT #1,COUNT$  'Input string that
                         'indicates number of bytes
                         'actually read from GPIB.
   PRINT COUNT$;" bytes were read from GPIB"
                         'Print number of bytes
                         'that were read from
                         'the GPIB.

2. PRINT #1,"EOS X,13"   'Send EOI with <CR> on
                         'wrt; do not terminate
                         'when <CR> is detected
                         'on rd; compare 7 bits.
   PRINT #1,"wrt #10 5"  'GPIB-232CT sends EOI with
                         '<CR> (CHR$(13)) to tell
                         'Listeners that this is
                         'the last byte of data.
   PRINT #1,"012345678"

3. PRINT #1,"eos"        'What are the current EOS
                         'settings?
```

response: X,13<CR><LF>

eot - Enable/Disable END Message on GPIB Writes

Type: Initialization function

Syntax: **`eot`** `[bool]<CR>`

Purpose: You use `eot` at the beginning of your program if you wish to change how the GPIB-232CT terminates GPIB writes. Using `eot`, you tell the GPIB-232CT to automatically send or not send the GPIB END message with the last byte that it writes to the GPIB.

Remarks: If the argument `bool` is 1, the GPIB-232CT automatically sends the END message with the last byte of each `wrt`. If the argument `bool` is 0, END is not sent. The power-on default is 1.

If you call `eot` without an argument, the GPIB-232CT returns to you a 1 to indicate END termination is currently enabled, or a 0 to indicate END termination is currently disabled.

The assignment made by `eot` remains in effect until you call `eot` again, call `onl`, or you turn off the GPIB-232CT.

The GPIB-232CT sends the END message by asserting the GPIB EOI signal during the last byte of a data transfer. `eot` is used primarily to send variable length data.

See Also: Chapter 4, the section entitled *GPIB Read and Write Termination Method*.

Examples:

```
1. PRINT #1,"eot 0"      'Disable END termination.

2. PRINT #1,"EOT 1"      'Send END with last byte.
   PRINT #1,"WRT 3"      'Write data to device at
                         'address 3.
   PRINT #1,"ABCDE"      'The EOI line is
                         'automatically asserted
                         'when the last byte (the
                         'letter E) is sent to tell
                         'the Listeners it is the
                         'last byte of data.

3. PRINT #1,"eot"        'What is the current EOT
                         'setting?

   response: 1<CR><LF> (END termination is
                            currently enabled)
```

gts - Go from Active Controller to Standby

Type: Specialized Controller function

Syntax: **g**`ts [bool]`<CR>

Purpose: You use `gts` to change the GPIB-232CT from Active Controller to Standby Controller. You use `gts` when the I/O and bus management functions do not meet the needs of your device. For example, you use `gts` if you wish to let two external devices to talk to each other directly. The GPIB-232CT can selectively participate in the handshake of the data transfer and hold off the handshake when it detects the END message. The GPIB-232CT can then take control synchronously without possibly corrupting the transfer.

Remarks: If the argument `bool` is 1, shadow handshaking is enabled. If the argument `bool` is 0, shadow handshaking is not performed.

If you call `gts` without an argument, the GPIB-232CT returns to you the current Controller status: `CSB,0` if the GPIB-232CT is in Standby without shadow handshaking; `CSB,1` if the GPIB-232CT is in Standby with shadow handshaking; `CAC` if the GPIB-232CT is CIC but is not in Standby, that is, it is the Active Controller; and `CIDLE` if the GPIB-232CT is not the CIC, that is, is an IDLE Controller.

`gts` causes the GPIB-232CT to go to the Controller Standby state and to unassert the ATN signal if it is initially the Active Controller. `gts` permits GPIB devices to transfer data without the GPIB-232CT participating in the transfer.

If you enable shadow handshaking, the GPIB-232CT participates in the data handshake as an Acceptor without actually reading the data. It monitors the transfers for the END (EOI or end-of-string character) message and holds off subsequent transfers. This mechanism allows the GPIB-232CT to take control synchronously on a subsequent operation such as `cmd` or `rpp`.

Before performing a `gts` with a shadow handshake, you should call `eos` to establish the proper end-of-string character or to disable the EOS detection if the end-of-string character used by the Talker is not known.

If you call `gts` with an argument and the GPIB-232CT is not CIC, the GPIB-232CT records the ECIC error.

See Also: `cac`.

Examples:

```
1. PRINT #1,"gts 0"       'GTS without shadow
                          'handshaking.

2. PRINT #1,"GTS 1"       'GTS with shadow
                          'handshaking.

3. PRINT #1,"gts"         'What is standby status?

   response: CSB,1<CR><LF> (GPIB-232CT is
                            in standby status
                            with shadow
                            handshaking)
```

ibcl - Enter ibcl Operating System

Type: General Use function

Syntax: **ib**cl<CR>

Purpose: You use ibcl if you wish to enter the interface bus control language operating system.

Remarks: After entering ibcl, the user has access to most of the same GPIB words (commands) as in S mode. In addition to the GPIB words, there are other predefined words that the user could use to write complex programs and custom applications.

See Also: The *GPIB-CT IBCL Reference Manual* included with your package.

Examples:

1.
```
PRINT #1,"ibcl"        'Enter the ibcl operating
                       'system.
```

   response: <CR><LF>
   ok<CR><LF>

2.
```
PRINT #1,": init 5 "+CHR$(34)+" IN;"+CHR$(34)+
          " wrt,"      'Define a word,
                       'called 'init' to initialize
                       'an IBM 7320 'Color Plotter.
```

   response: ok<CR><LF>

3.
```
PRINT #1,"init"        'Executed the word,
                       'defined in Example 2.
```

   response: ok<CR><LF>

```
4. PRINT #1,"bye"          'Exit ibcl and return to
                           'the NI610 operating
                           'system.
```

response:*

```
          END,CMPL,REM,CIC,LACS
          NGER
          NSER
          165
```

*Assuming you had continuous status reporting enabled when you called `ibcl`.

id - Identify System

Type: General Use function

Syntax: **id**<CR>

Purpose: You use id if you wish to know the revision level of your software, or if you wish to know how much RAM is installed in your GPIB-232CT.

Remarks: The identification is returned in three strings. The first two strings identify the company product model, the software revision level, and a copyright notice. The third string identifies the number of bytes of RAM in the GPIB-232CT.

Example:

```
1. PRINT #1,"id"          'Get system
                          'identification.
```

```
response:    GPIB-232CT Rev. A.0<CR><LF>
             (c)1988 National Instruments<CR><LF>
             64K bytes RAM<CR><LF>
```

ist - Set or Clear Individual Status Bit

ist: Parallel Poll function

Syntax: **is**t [bool]<CR>

Purpose: You use ist when the GPIB-232CT participates in a parallel poll that is conducted by another device that is Active Controller.

Remarks: If the argument bool is 1, the GPIB-232CT's individual status bit is set to 1. If the argument bool is 0, the GPIB-232CT's individual status bit is cleared. The power-on default is 0.

If you call ist without an argument, the GPIB-232CT returns the value of its individual status bit.

See Also: ppc, and Appendix F, *Parallel Polling*.

Examples:

```
1. PRINT #1,"ist 1"      'Set ist to 1.

2. PRINT #1,"IST 0"      'Clear ist to 0.

3. PRINT #1,"ist"        'What is ist set to?

   response: 0<CR><LF> (ist is currently 0)
```

loc - Go to Local *

Type: Bus Management function

Syntax: `loc [alist]`<CR>

Purpose: You use `loc` to put a device in local program mode. In this mode you can program the device from its front panel. Since a device must usually be placed in remote program mode before it can be programmed from the GPIB, the GPIB-232CT automatically puts the device in remote program mode. You then use `loc` to return devices to local program mode.

Remarks: The argument `alist` is a list of `addrs` separated by commas or spaces. `addrs` are device addresses that specify the GPIB addresses of the devices you wish to return to local mode.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

`0+2` or `0+98` or `32+98` or `0+\x62`

If you call `loc` with `alist`, the GPIB-232CT places the specified device(s) in local mode using the Go To Local (GTL) command.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If you call `loc` without `alist`, and the GPIB-232CT is System Controller, the GPIB-232CT returns all devices to local mode by unasserting REN and asserting it again. If you call `loc` without `alist` and the GPIB-232CT is not System Controller, the GPIB-232CT records the ESAC error.

See Also: Appendix B for more error information.

Examples:

1.
```
PRINT #1,"loc 6+22,4+23,7"
                        'Put 3 devices in local
                        'mode.
```

2.
```
PRINT #1,"LOC"          'Put all devices in local
                        'mode.
```

onl - Place the GPIB-232CT Online/Offline

Type: Initialization function

Syntax: **o**`nl [bool]`<CR>

Purpose: You use `onl` to disable communications between the GPIB-232CT and the GPIB, or to reinitialize the GPIB-232CT characteristics to their default values.

Remarks: If the argument `bool` is 1, the GPIB-232CT places itself online. If the argument `bool` is 0, the GPIB-232CT places itself offline. By default, the GPIB-232CT powers up online, is in the Idle Controller state, and configures itself to be the System Controller.

If you call `onl` without an argument, the GPIB-232CT returns the current status of the GPIB-232CT, which is 0 if the GPIB-232CT is offline and 1 if the GPIB-232CT is online.

Placing the GPIB-232CT offline can be thought of as disconnecting its GPIB cable from the other GPIB devices.

Placing the GPIB-232CT online allows the GPIB-232CT to communicate over the GPIB, and also restores all GPIB-232CT settings to their power-on values.

See Also: Tables 4-1 and 4-2 for the GPIB-232CT power-on settings.

Examples:

```
1. PRINT #1,"onl 1"       'Put the GPIB-232CT online
                          'and restore its power-on
                          'settings.

2. PRINT #1,"ONL 0"       'Put the GPIB-232CT
                          'offline to prevent
                          'it from communicating
                          'with the 'GPIB.
```

pct - Pass Control

Type: Specialized Controller function

Syntax: **pc**t addr<CR>

Purpose: You use pct to pass Controller-In-Charge (CIC) authority from the GPIB-232CT to some other device.

Remarks: The argument addr is the address of the device you wish to pass control to. addr consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+). Both addresses are expressed as numeric strings.

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary addresses. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

pct passes CIC authority from the GPIB-232CT to the device specified by addr. The GPIB-232CT automatically goes to Controller Idle State. It is assumed that the target device has Controller capability.

If you call pct with an argument and the GPIB-232CT is not CIC, it records the ECIC error.

If you call pct without an argument, the GPIB-232CT records the EARG error.

Example:

```
1. PRINT #1,"pct 7+18"  'Pass control to device
                        'with primary address 7
                        'and secondary address 18.
```

ppc - Parallel Poll Configure

Type: Parallel Poll function

Syntax: **ppc** addr,ppr,s [addr,ppr,s]...<CR>

Purpose: You use ppc to configure specified devices to respond to parallel polls in a certain manner.

Remarks: addr specifies the GPIB address of the device to be enabled or disabled for parallel polls. addr consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+). Both addresses are expressed as numeric strings.

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary addresses. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

The argument ppr is an integer string between 1 and 8 specifying the data line on which to respond.

The argument s is either 0 or 1 and is interpreted along with the value of the device's individual status bit to determine whether to drive the line true or false.

Each group of addr,ppr,s can be separated by either a comma or space, just as any list of arguments.

If you call ppc without an argument, the GPIB-232CT records the EARG error.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System

Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

The GPIB-232CT takes the arguments `ppr` and `s` and constructs the appropriate parallel poll enable (PPE) message for each `addr` specified.

When `addr` is the address of the GPIB-232CT, the GPIB-232CT programs itself to respond to a parallel poll by setting its local poll enable (LPE) message to the value specified.

See Also: `ist`, `ppu`, `rpp`, and Appendix F, *Parallel Polling*.

Example:

```
1. PRINT #1,"PPC 18+23,8,0 23+10,7,1"
                          'Configure 2 devices for
                          'parallel poll.
   PRINT #1,"RPP"         'Conduct a Parallel poll
                          'of 2 devices configured
                          'above.

   response:192<CR><LF>       (both devices
                               responded positively)

   INPUT #1,PPR%          'Assign parallel poll
                          'response to integer
                          'variable.
```

ppu - Parallel Poll Unconfigure

Type: Parallel Poll function

Syntax: **`ppu`** `[alist]`<CR>

Purpose: You use `ppu` if you are performing parallel polls and you wish to prevent certain devices from responding.

Remarks: The argument `alist` is a list of `addrs` that are separated by commas or spaces. `addrs` are device addresses that specify the GPIB addresses of the device or devices to be disabled from parallel polls.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

`0+2` or `0+98` or `32+98` or `0+\x62`

If you call `ppu` with `alist`, the GPIB-232CT unconfigures from parallel polls only those devices specified in `alist`.

If you call `ppu` without `alist`, the GPIB-232CT unconfigures all devices from parallel polls.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If the address of the GPIB-232CT is included in `alist`, the GPIB-232CT disables itself from responding to parallel polls.

See Also: `ist`, `ppc`, `rpp`, and Appendix F, *Parallel Polling*.

Examples:

1.
```
PRINT #1,"ppu 14"    'Send the PPU command to
                     'device 14.
```

2.
```
PRINT #1,"PPU"       'Send the PPU command to
                     'all devices.
```

rd - Read Data *

Type: I/O function

Syntax: **rd** `#count [addr]`<CR>

Purpose: You use `rd` to read data from the GPIB.

Remarks: The argument `#count` is a numeric string preceded by a number sign (#). `#count` specifies the number of bytes to read. `count` must not contain a comma. It can specify a number between 1 and 65535.

The argument `addr` is a device address that specifies the address of the device to be addressed as a Talker. `addr` consists of a primary address and a secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

`0+2` or `0+98` or `32+98` or `0+\x62`

The GPIB-232CT reads data from the GPIB until the specified byte count is reached, the GPIB END message is received with a data byte, the EOS byte is received, or a timeout occurs.

Because you may not know for certain the number of bytes actually read from the GPIB, the GPIB-232CT returns the received GPIB data to you in the following manner. First, the GPIB-232CT returns to you all bytes it read from the GPIB. Next, it sends null bytes until the total number of bytes returned to you matches your requested count. Finally, it returns a numeric string representing the number of bytes that it actually read from the GPIB.

For example, if you send the GPIB-232CT the programming message `"rd 10"`<CR>, it reads data from the GPIB until it receives 10 bytes of data, the END message, or an `eos` byte. Let's say the GPIB-232CT receives END with the fourth data byte. The GPIB-232CT then returns to you the 4 data bytes, followed by 6 null bytes, followed by an ASCII 4 and <CR><LF>. A null byte is decimal 0. You should always read back count bytes of data from the serial port, then look at the remaining bytes to determine how many of the count bytes were read from the GPIB. Refer to the example at the end of this description.

The GPIB-232CT aborts the GPIB read and records the EABO error if, at any time during the GPIB read, the time limit set for I/O functions expires. This limit is 10 seconds unless you use `tmo` to change it.

If the GPIB-232CT is CIC, `rd` will cause the GPIB-232CT to address itself to listen if it is not already addressed. If you specify the address of the Talker, the GPIB-232CT will also address that device to talk. If you do not specify the address of the Talker, the GPIB-232CT will assume that the Talker has already been addressed.

The GPIB-232CT then places itself in Standby Controller state with ATN off and remains there after the read operation is completed.

If you specify an address, the GPIB-232CT must be CIC to perform the addressing.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you call `rd` with the address argument, and you previously passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If the GPIB-232CT is not CIC and you do not specify the Talker address, the GPIB-232CT assumes it will be addressed by the Controller, then proceeds.

If you call `rd` without an argument, the GPIB-232CT records the EARG error.

See Also: `eos`, `eot`, and `tmo`.

Example:

```
1. PRINT #1,"rd #10 3"    'Read up to 10 bytes from
                          'the GPIB device at
                          'address 3.
   RESP$=INPUT$(10,#1)    'Input 10 bytes from
                          'serial port buffer.
   INPUT #1,COUNT%        'Input ASCII string
                          'representing number of
                          'bytes read from the GPIB.
                          'COUNT% is number of bytes
                          'read from GPIB; remaining
                          'bytes in RESP$ can be
                          'ignored.
```

rpp - Request (Conduct) a Parallel Poll

Type: Parallel Poll function

Syntax: **rp**p<CR>

Purpose: You use `rpp` if you wish to conduct a parallel poll to obtain information from several devices at the same time.

Remarks: `rpp` causes the GPIB-232CT to conduct a parallel poll of previously configured devices by sending the IDY message (ATN and EOI both asserted) and reading the response from the GPIB data lines. The GPIB-232CT pulses the IDY message for greater than or equal to 2 microseconds and expects valid responses within that time. It remains Active Controller afterward.

The GPIB-232CT returns the Parallel Poll Response (PPR) following the poll in the form of a numeric string representing the decimal value of the response.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

See Also: `ist`, `ppc`, `ppu`, and Appendix F, *Parallel Polling*.

Example:

```
1. PRINT #1,"ppc 13,1,0 15,3,0"+CHR$(13)+"rpp"
                        'Configure 2 devices for
                        'parallel polls and poll
                        'them.

   response: 5<CR><LF>      (both devices
                             responded positively)

   INPUT #1,PPR%        'Get parallel poll
                        'response from serial
                        'port buffer and assign
                        'it to integer variable
                        'PPR%.
   PRINT #1,"ppu"       'Unconfigure all devices
                        'from parallel polls.
```

rsc - Request System Control

Type: Initialization function

Syntax: **`rsc`** `[bool]<CR>`

Purpose: You use `rsc` if some other device in your GPIB system should be System Controller.

Remarks: If the argument `bool` is 1, the GPIB-232CT configures itself to be the GPIB System Controller. If the argument `bool` is 0, the GPIB-232CT configures itself as not System Controller.

If you call `rsc` without an argument, the GPIB-232CT returns to you its System Controller status, which is 0 if the GPIB-232CT is not currently System Controller or 1 if the GPIB-232CT is System Controller.

As System Controller the GPIB-232CT can send the Interface Clear (IFC) and Remote Enable (REN) messages to GPIB devices. If some other Controller asserts Interface Clear, the GPIB-232CT cannot respond unless it is configured as not System Controller.

In most applications, the GPIB-232CT will be System Controller. In some applications, the GPIB-232CT will never be System Controller. In either case, `rsc` is used only if the computer is not going to be System Controller while the program executes. The IEEE-488 standard does not specifically allow schemes in which System Control can be passed from one device to another; however, `rsc` could be used in such a scheme.

The GPIB-232CT configures itself to be System Controller at power-on.

See Also: `sic` and `sre`.

Examples:

```
1. PRINT #1,"rsc 1"       'Enable GPIB-232CT to be
                          'System Controller.

2. PRINT #1,"rsc 0"       'Disable system control.

3. PRINT #1,"rsc"         'What is the current
                          'System Controller
                          'status?

   response: 0<CR><LF>      (GPIB-232CT is not the
                             System Controller)
```

rsp - Request (Conduct) a Serial Poll

Type: Serial Poll function

Syntax: **`rsp`** `alist<CR>`

Purpose: You use `rsp` if you wish to conduct a serial poll to obtain device-specific status information from one or more devices.

Remarks: The argument `alist` is a list of `addrs` that are separated by commas or spaces. `addrs` are device addresses that specify the GPIB addresses you wish to poll.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary addresses. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

`0+2` or `0+98` or `32+98` or `0+\x62`

`rsp` serially polls the specified devices to obtain their status bytes. If bit 6 (the hex 40 or RQS bit) of a device's response is set, its status response is positive; that is, that device is requesting service. Before `rsp` completes, all devices are unaddressed.

The interpretation of each device's response, other than the RQS bit, is device specific. For example, the polled device might set a particular bit in the response byte to indicate that it has data to transfer, and another bit to indicate a need for reprogramming. Consult the device documentation for interpretation of the response byte.

Each device's serial poll response byte is returned as a numeric string giving the decimal value of the byte, followed by <CR> and <LF>. If a device does not respond in the timeout period, the GPIB-232CT returns string -1 and records the EABO error. The time limit is set to 1/10 second unless you called `tmo` to change it. Each response corresponds directly to an address you specify, therefore, there are exactly as many lines of responses, including -1, as the number of addresses you specify.

If you call `rsp` and the GPIB-232CT is not CIC, it attempts to become CIC. If it cannot become CIC, it records the ECIC error. Refer to Appendix B for more information.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If you call `rsp` without an argument, the GPIB-232CT records the EARG error.

See Also: `tmo` for timeout information.

Example:

```
1. PRINT #1,"rsp 1+28,5,9"     'Poll 3 devices.

   response:   42<CR><LF>     (device 9 did not
               30<CR><LF>      respond within the
               -1<CR><LF>      timeout period)

   DIM SPR%(2)             'Read 3 responses from
   FOR I=0 to 2            'serial port buffer.
   LINE INPUT #1,SPR%(I)   'Store each serial poll
   IF SPR%(I) = -1 THEN GOSUB 1000
   NEXT I                  'Response in the array.
                           '1000 is an error routine.
   REM Code will now interpret poll responses.
```

rsv - Request Service/Set or Change Serial Poll Status Byte

Type: Serial Poll function

Syntax: **`rsv`** `[spbyte]<CR>`

Purpose: You use `rsv` if the GPIB-232CT is not the GPIB Controller and you wish to request service from the Controller using the Service Request (SRQ) signal. The GPIB-232CT will provide a user defined status byte when the Controller serially polls it.

Remarks: The argument `spbyte` is a numeric string specifying the decimal value of the new GPIB-232CT serial poll response byte.

The serial poll response byte is the status byte the GPIB-232CT provides when serially polled by another device that is CIC. If bit 6 (hex 40 RQS bit) is also set, the GPIB-232CT additionally requests service by asserting the SRQ line.

If you call `rsv` without an argument, the GPIB-232CT returns a numeric string containing the decimal value of its serial poll status byte.

Examples:

```
1. PRINT #1,"rsv \x46"    'Request service with
                          'serial poll response = 6.

2. PRINT #1,"rsv"         'What is the current
                          'serial poll status byte?

   response: 70<CR><LF>       (The current status
                              byte = decimal 70 or
                              hex 46)
```

sic - Send Interface Clear

Type: Specialized Controller function

Syntax: **si**c [time]<CR>

Purpose: You use sic if the initialization, I/O, or bus management functions do not meet the needs of your device, and you wish to have more precise control over the GPIB. sic makes the GPIB-232CT CIC and initializes the GPIB. sic is not a function you will use frequently because in most cases the first I/O or bus management function you call will do this automatically.

Remarks: The argument time is a numeric string specifying any number of seconds between .0001 and 3600, which corresponds to time limits between 100 microseconds and 1 hour. time must not contain a comma.

If you call sic without an argument, IFC is sent for 500 microseconds. The action of asserting IFC for at least 100 microseconds initializes the GPIB and makes the interface board become CIC. When needed, sic is generally used at the beginning of a program to make the GPIB-232CT CIC and is used when a bus fault condition is suspected.

The IFC signal resets only the GPIB interface functions of bus devices and not the internal device functions. Device functions are reset with the clr programming message. To determine the effect of these messages, consult the device documentation.

If you are in a debugging environment, you may want to vary the amount of time IFC is asserted. For example, you may set time to 10 seconds to allow you to check on a bus analyzer that IFC is actually being asserted. Otherwise, you do not need to include the time argument.

The GPIB-232CT records the ESAC error if you have disabled its System Controller capability with the rsc function. It records the EARG error if you specify a time outside the range .0001 to 3600.

See Also: `clr` and Appendix D.

Examples:

```
1. PRINT #1,"sic"          'Send interface clear for
                           '500 microseconds.

2. PRINT #1,"SIC .01"      'Send interface clear for
                           '10 milliseconds.
```

spign - Ignore Serial Port Errors

Type: Serial Port function

Syntax: **spi**gn [bool]<CR>

Purpose: You use spign at the beginning of your program if you wish to change the effect that serial port errors have on how the GPIB-232CT processes programming messages and data. This function tells the GPIB-232CT to ignore or not to ignore the occurrence of serial port errors. By default, the GPIB-232CT ignores serial port errors.

Remarks: If the argument bool is 0, the GPIB-232CT will not ignore serial port errors. When bool is 0, the GPIB-232CT does not execute programming messages that contain serial port errors. A list of serial port errors are given in Appendix B. Also, if a serial port error occurs with any byte contained in a cmd or wrt data string, the GPIB-232CT discards that data byte and all remaining bytes in the string.

The serial port errors include parity, overrun, framing, and overflow errors.

If the argument bool is 1, the GPIB-232CT executes all programming messages and sends all data, even if serial port errors occur as the messages and data bytes are received.

If you call spign without an argument, the GPIB-232CT returns to you the current setting.

See Also: cmd and wrt.

Examples:

1.
```
PRINT #1,"spign 0"   'Do not execute programming
                     'messages or process data
                     'that contain serial port
                     'errors.
```

2.
```
PRINT #1,"spign 1"   'Execute all programming
                     'messages and send all
                     'data, even if serial port
                     'errors occur.
```

sre - Set (or clear) Remote Enable

Type: Specialized Controller function

Syntax: **sre** [bool]<CR>

Purpose: You use sre if the I/O and bus management functions do not meet the needs of your device. sre gives you more precise control over the GPIB. Use sre to turn the Remote Enable signal on and off. sre is not a function you will use frequently because in most cases, the first I/O or bus management function you call will set remote enable automatically.

Remarks: If the argument bool is 1, the GPIB-232CT asserts the Remote Enable (REN) signal. If the argument bool is 0, the GPIB-232CT unasserts REN.

Many GPIB devices have a remote program mode and a local program mode. It is usually necessary to place devices in remote mode before programming them from the GPIB. A device enters the remote mode when the REN line is asserted and the device receives its listen address.

Use cmd to send a device its listen address after using sre. Use loc to return the device to local program mode.

If you call sre with an argument and the GPIB-232CT is not System Controller, the GPIB-232CT records the ESAC error.

If you call sre without an argument, the GPIB-232CT returns its current remote status: 1=remote, 0=local.

See Also: rsc, cmd, and loc.

Examples:

```
1. PRINT #1,"SRE 1"      'Set REN.

2. PRINT #1,"sre 0"      'Unassert REN.
```

stat - Return GPIB-232CT Status

Type: General Use function

Syntax: **st**`at [[c] n]`<CR>
or
st`at [c] s`<CR>
or
st`at [c] n s`<CR>

Purpose: You use `stat` to obtain the status of the GPIB-232CT to see if certain conditions are currently present. You use `stat` most often to see if the previous operation resulted in an error.

Remarks: You should use `stat` frequently in the early stages of your program development when your device's responses are likely to be unpredictable. The GPIB-232CT responds with status information in a form depending on the mode or combination of modes you chose. `n` indicates that the status information will be returned as numeric strings. `s` indicates that the status information will be returned in symbolic format, that is, as mnemonic strings. `c` specifies that the status will be returned after each programming message, eliminating the need to call `stat` after each programming message.

Normally, you use `s` only when you are debugging your code and you want to print the mnemonic for each piece of status information.

The status information returned by the GPIB-232CT contains four pieces of information: the GPIB-232CT status, a GPIB-error code, a serial-error code, and a count. The GPIB-232CT returns a <CR><LF> following each piece of the response.

Status represents a combination of GPIB-232CT conditions. Internally in the GPIB-232CT, status is stored as a 16-bit integer. Each bit in the integer represents a single condition. A bit value of 1 indicates that the corresponding condition is in effect; a bit value of zero indicates that the condition is not in effect. Since more than one GPIB-232CT condition can exist at one time, more than one bit can be set in status. The

highest order bit of status, also called the sign bit, is set when the GPIB-232CT detects either a GPIB error or a serial port error. Consequently, when status is negative, an error condition exists, and when status is positive, no error condition exists.

GPIB-error represents a single GPIB error condition present.

Serial-error represents a single serial error condition present.

count is the number of bytes transferred over the GPIB by the last `rd`, `wrt`, or `cmd` function.

Table 5-2. GPIB Status Conditions

Numeric Value (n)	**Symbolic Value (s)**	**Description**	**Bit**
-32768	ERR	Error detected	15
16384	TIMO	Timeout	14
8192	END	EOI or EOS detected	13
4096	SRQI	SRQ detected while CIC	12
2048	—	Reserved	11
1024	—	Reserved	10
512	—	Reserved	9
256	CMPL	Operation completed	8
128	LOK	Lockout state	7
64	REM	Remote state	6
32	CIC	Controller-In-Charge	5
16	ATN	Attention asserted	4

(continues)

Table 5-2. GPIB Status Conditions (continued)

Numeric Value (n)	Symbolic Value (s)	Description	Bit
8	TACS	Talker active	3
4	LACS	Listener active	2
2	DTAS	Device trigger state	1
1	DCAS	Device clear state	0

Table 5-3. GPIB Error Conditions

Numeric Value (n)	Symbolic Value (s)	Description
0	NGER	No GPIB error condition to report
1	ECIC	Command requires GPIB-232CT to be CIC
2	ENOL	Write detected no Listeners
3	EADR	GPIB-232CT not addressed correctly
4	EARG	Invalid argument or arguments
5	ESAC	Command requires GPIB-232CT to be SC
6	EABO	I/O operation aborted
7-10	—	Reserved
11	ECAP	No capability for operation

(continues)

Table 5-3. GPIB Error Conditions (continued)

Numeric Value (n)	Symbolic Value (s)	Description
12-16	—	Reserved
17	ECMD	Unrecognized command

Table 5-4. Serial Port Error Conditions

Numeric Value (n)	Symbolic Value (s)	Description
0	NSER	No serial port error condition to report
1	EPAR	Serial port parity error
2	EORN	Serial port overrun error
3	EOFL	Serial port receive buffer overflow
4	EFRM	Serial port framing error

A detailed description of the conditions under which each bit in status is set or cleared can be found in Appendix B.

In general, the GPIB-232CT updates the first three status variables at the end of each programming message. It updates the fourth status variable, `count`, after a `cmd`, `rd`, or `wrt` function. The errors reported correspond to the previous programming message. For example, if you call `wrt` and then `stat s`, any errors returned to you correspond to errors in the `wrt` programming message, not `stat`. However, if status is returned in continuous mode, the status information corresponds to the current programming message. For example, suppose you call `stat c s` to set up continuous

status reporting. After reading the status information returned for the `stat` call, you call `wrt`. The GPIB-232CT then returns the status information that corresponds to the `wrt` message.

Refer to the following examples for ways in which to make use of the status information.

When you wish to begin continuous status reporting, send the `stat c s`, `stat c n`, or `stat c n s` programming message. Status information will be immediately returned indicating the current status conditions. When you call `stat` with both `s` and `n` the numeric status is always returned first.

If you call `stat` without an argument, continuous status reporting is disabled.

Notice that when you send several programming messages to the GPIB-232CT, it buffers them and processes each one without any delay in between. However, if you enable continuous status reporting and check the status of each programming message before sending the next, the GPIB-232CT waits for each subsequent programming message to arrive at the serial port before processing it. This slows down the overall performance of your program. If speed is a primary concern, disable continuous status reporting.

Examples:

1.
```
10   PRINT #1,"stat n"     'Get GPIB-232CT status.
20   REM GPIB-232CT responds with:
30   REM 340<CR><LF>0<CR><LF>0<CR><LF>0<CR><LF>
40   REM Now read status into variables.
50   INPUT#1,STATUS%,GPIBERR%,SPERR%,COUNT%
60   REM Go to error routine at 500 if error.
70   IF STATUS% < 0 THEN GOTO 500
80   REM Go to SRQ service routine if SRQ is
90   REM asserted.
100  IF (STATUS% AND &H1000) THEN GOTO 400
     ..
400  REM
410  REM Place code here to service SRQ.
```

```
420  REM
     ..
500  REM Print GPIB-error and serial-error values
510  REM to determine what errors occurred.
520  PRINT "GPIB-error = ";GPIBERR%
530  PRINT "Serial-error = ";SPERR%
540  STOP
```

2.
```
10   PRINT #1, "stat s"
20   REM If it has just read 3 bytes from the
30   REM GPIB, the GPIB-232CT responds with:
40   REM    CMPL,REM,ATN,LACS<CR><LF>NGER<CR><LF>
50   REM    NSER<CR><LF>3<CR><LF>
```

3. The following list illustrates what appears on the screen when you are programming the GPIB-232CT from a terminal. Programming messages you enter are in regular type. GPIB-232CT responses are in **boldface**. The statements in parentheses are comments.

```
stat c s n (Enable continuous status reporting.)
344
0
0
3
CMPL,REM,ATN,TACS          (Status returned.)
NGER
NSER
3
wrt 10
ABCDE                      (Write the string ABCDE)
296                        (to device 10.)
0                          (Status returned.)
0
5
CMPL,CIC,TACS
NGER
NSER
5
```

tmo - Change or Disable Time Limit

Type: Initialization function

Syntax: **tm**o [timeio][,timesp]<CR>

Purpose: You use tmo at the beginning of your program to change the time limits in effect on the GPIB-232CT. The time limits prevent the GPIB-232CT from hanging indefinitely when waiting for critical events to occur.

Remarks: The arguments timeio and timesp are numeric strings. timeio specifies the amount of time in seconds the GPIB-232CT waits for an I/O operation (rd, wrt, cmd) or the wait function to complete. timesp specifies the amount of time in seconds each device is given in which to respond to a serial poll. The power-on timeouts are 10 seconds for timeio and 1/10 of a second for timesp.

timeio and timesp can be any decimal number between .00001 and 3600, which corresponds to time limits between 10 microseconds and 1 hour. 10,.1 specifies a time of 10 seconds for I/O operations and 1/10 of a second for serial poll response. timeio and timesp can also be 0, which disables either timeout accordingly. Neither timeio nor timesp can contain commas.

The timeio time limit is in effect for the cmd, rd, and wrt functions. If the GPIB-232CT cannot complete any of these functions within the period of time set by timeio, it aborts the function and records the EABO error. Bytes that were transferred before the timeout are not affected.

The timeio time limit is also the maximum amount of time the wait function waits when you call it with the TIMO bit set in the wait mask.

The timesp time limit is in effect only for the rsp function. If a polled device fails to respond within the amount of time indicated by timesp, the GPIB-232CT returns an error flag.

If you want to change only the timeout value for serial polls, a comma must precede the serial poll timeout value.

If you call `tmo` without an argument, the GPIB-232CT returns a numeric string representing the current timeout settings. It records the EARG error if you specify a time value outside the range .00001 to 3600.

The assignment made by this function remains in effect until you call `tmo` again, call `onl`, or turn off the GPIB-232CT.

See Also: `rsp`.

Examples:

```
1. PRINT #1,"tmo 30"      'Set timeout for I/O
                          'operations to 30 seconds.

2. PRINT #1,"tmo"         'Print current timeout
                          'settings.

   response: 30,.1<CR><LF>

3. PRINT #1,"tmo ,1"      'Set serial poll timeout
                          'for one second; leave I/O
                          'timeout unchanged.
```

trg - Trigger Selected Device(s) *

Type: Bus Management function

Syntax: **tr**`g alist`<CR>

Purpose: You use `trg` to trigger the specified devices. The instructions for each GPIB device explain when you should trigger them and what effect the trigger has.

Remarks: The argument `alist` is a list of `addrs` separated by commas or spaces. `addrs` are device addresses that specify the GPIB addresses you wish to trigger.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary addresses. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

`0+2` or `0+98` or `32+98` or `0+\x62`

If you call `trg` without an argument, the EARG error is posted.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

Example:

```
1. PRINT #1,"trg 2+10,4,5+7"   'Trigger 3 devices.
```

wait - Wait for Selected Event

Type: General Use function

Syntax: **wa**`it mask`<CR>

Purpose: You use `wait` to monitor selected GPIB events and to delay any further GPIB-232CT activity until any of them occur.

Remarks: The argument `mask` is a numeric string that specifies the events to wait for. The numeric string represents a bit mask containing a subset of the same bit assignments as the status word described in the `stat` function. Each bit is set or cleared to wait or not to wait, respectively, for the corresponding event to occur. The numeric string can be expressed as decimal, octal, or hexadecimal.

After receiving the `wait` programming message, the GPIB-232CT monitors GPIB activity. When any event corresponding to the bits set in `mask` occurs, the GPIB-232CT returns status information indicating its current status. If continuous status reporting has been enabled, status will be reported in the format requested. If continuous status has not been enabled, status will be returned in numeric format.

You could use `wait`, for example, if you wish to wait until a device requests service before you perform a serial poll. In this case, you send the wait programming message with `mask=4096`, then wait for status information to be returned. You then check that status to see if the SRQI bit is set in the returned status indicators.

To prevent the GPIB-232CT from waiting indefinitely for SRQ to be asserted, set the SRQI and TIMO bits by setting the mask equal to `4096 + 16384`. This will cause the wait to terminate either on SRQI or TIMO, whichever occurs first.

Table 5-5. Wait Mask Values

Decimal Value	Mnemonic	Description	Hex Value	Bit
—	—	Reserved	—	15
16384	TIMO	Timeout	4000	14
8192	END	EOI or EOS detected	2000	13
4096	SRQI	SRQ detected while CIC	1000	12
—	—	Reserved	—	11
—	—	Reserved	—	10
—	—	Reserved	—	9
—	—	Reserved	—	8
128	LOK	Lockout state	80	7
64	REM	Remote state	40	6
32	CIC	Controller-In-Charge	20	5
16	ATN	Attention asserted	10	4
8	TACS	Talker active	8	3
4	LACS	Listener active	4	2
2	DTAS	Device trigger state	2	1
1	DCAS	Device clear state	1	0

If `mask=0` the function completes immediately and returns the current status.

If the TIMO bit is 0 or the `timeio` time limit is set to 0 with `tmo`, timeouts for this function are disabled. You should disable timeouts only when you are certain the selected event will occur; otherwise, the GPIB-232CT waits indefinitely for the event to occur.

If you call `wait` without an argument, the GPIB-232CT records the EARG error.

See Also: `stat` and `tmo`.

Examples:

1.
```
PRINT #1,"wait \x5000"
                        'Wait for TIMO or SRQI.
INPUT#1,STATUS%,GPIBERR%,SPERR%,COUNT%
                        'Get status info.
IF (STATUS% AND &H4000) <> 0 THEN GOTO 1000
                        'If TIMO bit is set we
                        'timed out before getting
                        'SRQI.  Go to an error
                        'routine at line 1000.
IF(STATUS% AND &H1000) <> 0 THEN GOTO 200
                        'If SRQI bit set, go to
                        'routine to conduct a
                        'serial poll.
```

2.
```
PRINT #1,"wait 4"     'Wait indefinitely to
                      'become LACS.
INPUT#1,STATUS%,GPIBERR%,SPERR%,COUNT%
                      'Get status info.
PRINT #1,"rd 10"      'Now that GPIB-232CT is
                      'addressed to listen, read
                      '10 bytes from the GPIB.
RESP$=INPUT$(10,#1)   'Input 10 bytes from
                      'serial port buffer.
INPUT #1,CNT%         'Input number of valid
                      'bytes in CNT$.
```

wrt - Write Data *

Type: I/O Function

Syntax: **wr**t [#count][alist]<CR>
data<CR>

Purpose: You use wrt to send data over the GPIB.

Remarks: The argument count is a numeric string preceded by a number sign (#count). The string specifies a number between 1 and 65535 and must not contain a comma. #count specifies the number of data bytes to send. The number of data bytes must not include the carriage return that indicates the end of the programming message.

The argument data is a string of 8-bit characters that are transferred without any translation to the GPIB.

The argument alist is a list of addrs separated by commas or spaces. addrs are addresses that specify the GPIB addresses of the Listener (or Listeners, if more than one address is given).

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits can be in the range from 0 through 30 for both the primary and the secondary addresses. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

When #count is not specified, the GPIB-232CT recognizes the end of the data string when it encounters a carriage return

or a linefeed. `#count` is required when your data string contains embedded carriage return or linefeed characters.

If you specify an address list, the GPIB-232CT must be CIC to perform the addressing. If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-232CT sends Interface Clear (IFC) to make itself CIC.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If you do not give an `alist` and the GPIB-232CT is not CIC, it assumes it will be addressed by the Controller. If you do not give an `alist` and the GPIB-232CT is CIC, it addresses itself as Talker and assumes the Listeners are already addressed.

The first part of this programming message, up to <CR>, is buffered, meaning the GPIB-232CT will not act upon it until it receives <CR>. The string that follows the first line is piped to the GPIB, non-buffered. This allows you to send a string larger than the GPIB-232CT's internal buffer with one programming message.

The GPIB-232CT aborts the GPIB write if it receives Device Clear or Selected Device Clear and its Listen Address. The GPIB-232CT aborts the GPIB write and records the EABO error if, at any time during the GPIB write, the time limit set for I/O functions expires. This limit is 10 seconds unless you use `tmo` to change it. The GPIB-232CT also aborts the `wrt` and records the ENOL error if there are no addressed Listeners when it begins to send data.

See Also: `tmo` for timeout information, Appendix B for more error information, and `spign` for serial port error handling information.

Examples:

1.
```
PRINT #1,"wrt #50 9+97" 'Write 50 bytes to
FOR I = 1 TO 50         'device at primary
PRINT #1,CHR$(A(I));    'address 9 and
NEXT I                  'secondary address 97.
PRINT #1,CHR$(13);      'Send carriage return.
```

2.
```
PRINT #1,"wrt 2"     'Write the data bytes ABCDE
PRINT #1,"ABCDE"     'at device at address 2.
```

xon - Change Serial Port XON/XOFF Protocol

Type: Serial Port function

Syntax: **x**`on [booltx][,boolrx]`<CR>

Purpose: You use `xon` at the beginning of your program to configure the GPIB-232CT to communicate over the serial port using the same XON/XOFF protocol as your serial device.

Remarks: The argument `booltx` specifies whether to enable the XON/XOFF protocol when sending data out on the serial port. If the argument `booltx` is a 1, the GPIB-232CT monitors its serial receive buffer for XON/XOFF characters as it sends data over the serial port. If it receives the XOFF character (decimal 19 or <CTRL>s), it will immediately stop sending data. When it receives the XON character (decimal 17 or <CTRL>q), it begins sending data again.

If you want to send a data string that may contain a <CTRL>s or <CTRL>q, you must disable `booltx`.

The argument `boolrx` specifies whether to enable the XON/XOFF protocol when receiving data over the serial port. If the argument `boolrx` is a 1, and the GPIB-232CT is receiving data over the serial port, it sends XOFF over the serial port (if its serial receive buffer is almost full). This tells the sender to stop sending data. When the GPIB-232CT serial port receive buffer has room to safely receive more bytes, the GPIB-232CT sends XON over the serial port. This tells the sender to begin sending data again.

You should use XON/XOFF when your computer or terminal does not recognize the hardware handshake protocol, and you are transferring very large amounts of data (greater than the serial port buffer size). Without it, you could be in danger of overflowing either the GPIB-232CT or your computer's internal buffer.

When would you want to enable the protocol in one case and not the other? Some computers use XON/XOFF protocol when transmitting data but not when receiving data. In this

case you might configure the GPIB-232CT using the second example.

Notice that the comma must precede the argument in this case.

The power-on default is that XON/XOFF for both cases is disabled.

If you call `xon` without an argument, the GPIB-232CT returns to you the current settings (1=protocol enabled, 0=protocol disabled).

Examples:

```
1. PRINT #1,"XON 1,1"     'Enable GPIB-232CT
                          'XON/XOFF protocol for
                          'TX and RX.

2. PRINT #1,"XON 0,1"     'Disable protocol on TX;
                          'enable protocol on RX.

3. PRINT #1,"XON"         'Return current settings.

   response: 0,1<CR><LF>
                          (transmit protocol
                          disabled, receive protocol
                          enabled)

4. PRINT #1, "XON ,0"     'Disable protocol on RX,
                          'keep current setting on
                          'TX.
```

Chapter 6
Installation and Configuration in G Mode

If you plan to operate in G mode, use this chapter to install and configure the GPIB-232CT. Then read Chapters 7 and 8 to learn about the programming messages.

Installation

There are five basic steps to installing the GPIB-232CT.

1. Inspect the GPIB-232CT for damage that may have been caused in shipment.
2. Verify the voltage requirement.
3. Configure the operating parameters.
4. Connect the cables.
5. Power on the unit.

These steps are described in more detail in the following sections.

Step 1: Inspection

Before you install the GPIB-232CT, inspect the shipping container and its contents for damage. If damage appears to have been caused in shipment, file a claim with the carrier. Retain the packing material for possible inspection and/or reshipment.

If the equipment appears to be damaged, do not attempt to operate it. Contact National Instruments for instructions.

Step 2: Verify the Voltage Requirement

The GPIB-232CT is shipped from the factory with either a 115V or 230V wall-mount supply. Verify that the voltage on the supply matches the voltage that is supplied in your area.

Caution: Operating the unit at any voltage other than the one specified could damage the unit.

Step 3: Configure the Operating Parameters

The GPIB-232CT is shipped from the factory configured to operate in S mode. To configure the GPIB-232CT to operate in G mode, follow these steps:

1. Disconnect power to the unit and disconnect any cables that may be connected to the unit.

2. Unscrew the two screws on the opposite sides of the rear panel.

3. Grab the rear panel bezel and pull it straight away from the rest of the unit. The card should slide out the back of the enclosure.

4. Locate the configuration DIP switch (U22) on the printed wire board.

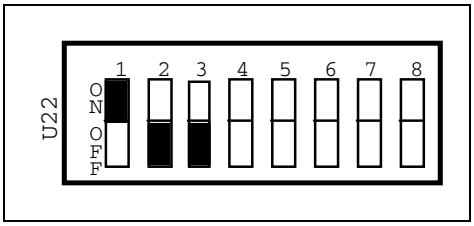
5. Set the switches for the desired mode of operation. Refer to the following section, *Set Configuration Switches*.

   Caution: Most of the circuitry in the GPIB-232CT uses advanced CMOS technology and can be damaged by static electricity. Avoid touching any of the components and take any necessary CMOS handling precautions.

6. Close the unit and re-insert the screws removed in Step 2.

Set Configuration Switches

The DIP switch at location U22 on the printed wire board is used to configure the GPIB address of the GPIB-232CT while in G mode. The DIP switch, shown in Figure 6-1, has eight configuration switches.



ON indicates that the switch is pressed on the "ON" side.

OFF indicates that the switch is pressed on the "OFF" side.

Figure 6-1. G Mode Switch Settings

Notice that to put the GPIB-232CT into G mode, Switch 1 of U22 should be ON and Switches 2 and 3 should be in the OFF position.

Connecting the serial device to the GPIB-232CT converts the serial device into a GPIB device. Each GPIB device has a GPIB address. The GPIB-232CT uses two GPIB addresses—the one you select by setting the configuration switches, and the second address equal to the address you select plus one. The first becomes the GPIB-232CT's address. The second becomes the serial device's address.

The GPIB addresses of the GPIB-232CT and the serial device must be different from the addresses of the other GPIB devices in your system. Determine the GPIB addresses of the other GPIB devices; then, using the list on the following page, select two consecutive addresses that are not already in use.

Configure the GPIB-232CT primary address by setting the switches as shown in Table 6-1. The corresponding talk and listen addresses are also shown.

Table 6-1. Primary Address Configurations

Switches 4	5	6	7	8	Primary Address	Listen Address Dec/ASCII		Talk Address Dec/ASCII	
OFF	OFF	OFF	OFF	OFF	0	32	SP	64	@
OFF	OFF	OFF	OFF	ON	1	33	!	65	A
OFF	OFF	OFF	ON	OFF	2	34	"	66	B
OFF	OFF	OFF	ON	ON	3	35	#	67	C
OFF	OFF	ON	OFF	OFF	4	36	$	68	D
OFF	OFF	ON	OFF	ON	5	37	%	69	E
OFF	OFF	ON	ON	OFF	6	38	&	70	F
OFF	OFF	ON	ON	ON	7	39	'	71	G
OFF	ON	OFF	OFF	OFF	8	40	(	72	H
OFF	ON	OFF	OFF	ON	9	41	)	73	I
OFF	ON	OFF	ON	OFF	10	42	*	74	J
OFF	ON	OFF	ON	ON	11	43	+	75	K
OFF	ON	ON	OFF	OFF	12	44	,	76	L
OFF	ON	ON	OFF	ON	13	45	-	77	M
OFF	ON	ON	ON	OFF	14	46	.	78	N
OFF	ON	ON	ON	ON	15	47	/	79	O
ON	OFF	OFF	OFF	OFF	16	48	0	80	P
ON	OFF	OFF	OFF	ON	17	49	1	81	Q

(continues)

Table 6-1. Primary Address Configurations (continued)

Switches					Primary Address	Listen Address Dec/ASCII		Talk Address Dec/ASCII	
4	**5**	**6**	**7**	**8**					
ON	OFF	OFF	ON	OFF	18	50	2	82	R
ON	OFF	OFF	ON	ON	19	51	3	83	S
ON	OFF	ON	OFF	OFF	20	52	4	84	T
ON	OFF	ON	OFF	ON	21	53	5	85	U
ON	OFF	ON	ON	OFF	22	54	6	86	V
ON	OFF	ON	ON	ON	23	55	7	87	W
ON	ON	OFF	OFF	OFF	24	56	8	88	X
ON	ON	OFF	OFF	ON	25	57	9	89	Y
ON	ON	OFF	ON	OFF	26	58	:	90	Z
ON	ON	OFF	ON	ON	27	59	;	91	[
ON	ON	ON	OFF	OFF	28	60	<	92	\
ON	ON	ON	OFF	ON	29	61	=	93	]
ON	ON	ON	ON	OFF	30	62	>	94	^

Example Settings for Configuration Switch

Figure 6-2 shows an example setting of the configuration switch. Switches 1, 2, and 3 are ON, OFF, and OFF, respectively, which indicates that the GPIB-232CT is operating in G mode. Switches 4 through 8 are OFF, OFF, ON, OFF, and ON, respectively. This configures the GPIB-232CT at GPIB address 5 and the serial device at address 6.

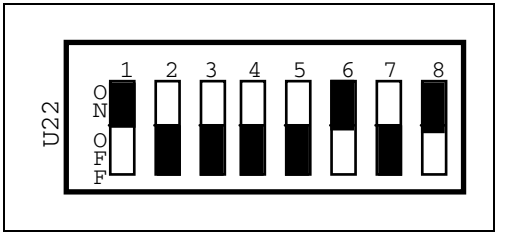


Figure 6-2. Example Settings for Switch U22

Note: Switch U20 is not used by S or G mode, but can be used for user applications in IBCL mode.

Step 4: Connect the Cables

Connect the cables as follows:

1. Connect the serial cable to the GPIB-232CT and securely fasten the holding screws. Connect the other end of the cable to your serial device. Be sure to use only shielded serial cable, and obey all RS-232 cabling restrictions.

2. Connect the GPIB cable to the GPIB-232CT and tighten the thumb screws on the connector. Connect the other end to your GPIB system. Be sure to obey all IEEE-488 cabling restrictions, and use only double-shielded GPIB cable.

3. Connect the power jack of the wall-mount power supply to the power receptacle on the back panel of the GPIB-232CT, then plug the supply into an AC outlet of the correct voltage.

Step 5: Power on the Unit

Power on your GPIB-232CT by using the front panel rocker switch. The **POWER** LED should come on immediately. The **READY** indicator should come on after the GPIB-232CT has passed its power-on self-test, indicating the unit is ready for operation.

If the **READY** indicator does not come on within seven seconds after the unit is powered on, recheck all connections and switch settings and retry the power-on sequence. If the **READY** light still fails to come on, contact National Instruments for further instructions.

Chapter 7
Programming in G Mode

This chapter shows how to program the GPIB-232CT in G mode using programming messages. It describes programming messages, their format, and how they are processed, along with the functions and function arguments that make up the programming messages.

This chapter also explains how to communicate with your serial device through the GPIB-232CT.

Programming Messages

You program the GPIB-232CT by sending it programming messages (which are ASCII strings) by way of its GPIB port.

Programming Message Format

The programming message consists of a function name, one or more arguments (optional), followed by a carriage return (<CR>), or a linefeed (<LF>), or a carriage return followed by a linefeed <CR><LF>. Carriage return and linefeed can be expressed in BASIC as `CHR$(13)` and `CHR$(10)`, respectively.

Example of a Programming Message

In the following lines of BASIC code:

```
WRT$ ="eos x,10"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
```

`WRT$` contains the programming message in which `eos` is the function name, `x` and `10` are the arguments, and `CHR$(13)` is the terminating carriage return. This programming message tells the GPIB-232CT to assert the EOI line when it sends the end-of-string character linefeed. The second line of the example is a GPIB-PC function call provided as part of a National Instruments product that allows a personal computer to control the GPIB from Microsoft BASIC. This function outputs the string in `WRT$` to the device GPIB-232CT.

You can enter programming messages in any combination of uppercase and lowercase letters.

How Messages are Processed

The GPIB-232CT processes each programming message on a line-by-line basis. The GPIB-232CT buffers the entire message, interprets the function name and arguments, then executes the message.

Function Names

The function names have been selected to indicate each function's purpose, thereby making your programs easy to understand. However, if you wish to reduce some overhead in your program and do not mind giving up these advantages, you can use only as much of the function name as is necessary to distinguish it from other functions. This abbreviated form of the function name is shown in **boldface** in the function tables and in the syntax portions of the function descriptions.

Function Argument Delimiters

When you type in a function, separate the first argument from the function name with at least one space. Separate each additional argument with at least one space or comma.

In the syntax portions of the function descriptions in Chapter 8, the square brackets (`[]`) that enclose some arguments indicate that those arguments are optional. Do not enter the brackets as part of your arguments.

Abbreviation for Argument

The term `bool` is an abbreviation used for an argument in the function descriptions. The value for `bool` is: 1 = true, on, or enable; or 0 = false, off, or disable.

Status Information

The function descriptions in Chapter 8 explain that the GPIB-232CT *records* specific status and error information. This means that it stores that information in its memory so that it is available when you request it.

The function descriptions also explain that the GPIB-232CT *returns* to you certain information. This means that the GPIB-232CT sends information to you over the GPIB.

Communicating with the GPIB-232CT and the Serial Device

The GPIB-232CT knows the type of GPIB data it is processing by using dual addressing. With dual addressing, the GPIB-232CT recognizes two different GPIB addresses. The first is the GPIB-232CT's address. The second is the serial device's address.

Address of the GPIB-232CT

The address of the GPIB-232CT is the primary address you set on the configuration switch, with secondary addressing disabled.

Address of the Serial Device

The address of the serial device is the GPIB-232CT primary address plus 1, with secondary addressing disabled. However, if you select a primary address of 30 with the configuration switch, the serial device will be at address 0.

The two primary addresses recognized by the GPIB-232CT are referred to as the GPIB-232CT address and the serial device address.

Addressing Terminology

When the GPIB-232CT receives the serial device address, the data it sends and receives is referred to as serial data.

When the GPIB-232CT receives its own address, the data it receives is referred to as programming messages; the data it sends is referred to as status information.

The GPIB-232CT and Serial Device as Listener

When the GPIB-232CT receives its own listen address, it examines the data received over the GPIB, treats it as a programming message or messages, and takes actions based on that data.

When the GPIB-232CT receives the serial device listen address, it forwards the data received over the GPIB to the serial port without examining the data for meaning.

For example, let's say you have a serial printer connected to the GPIB-232CT and you wish to send a data file from your computer over the GPIB to the printer. Ordinarily, when the printer buffer is full, the printer sends the XOFF character; when the printer is ready to receive more characters, it sends the XON character. So, before you send your file to the printer, you must tell the GPIB-232CT to watch for XON/XOFF characters from the printer.

Do this by first addressing the GPIB-232CT to listen by sending its listen address. Then, send it the programming message `xon ,1`. The GPIB-232CT interprets this programming message and acts upon it without sending any data on to the serial device.

Next, address the serial device to listen by sending to the GPIB-232CT the serial device listen address. Then, send the data over the GPIB. The GPIB-232CT sends that data on to the printer without examining that data for meaning. The LISTEN LED on the GPIB-232CT front panel is lit when either the GPIB-232CT is addressed to listen or the serial device is addressed to listen.

The GPIB-232CT and Serial Device as Talker

When the GPIB-232CT receives its own talk address, it sends out status information.

When the GPIB-232CT receives the serial device talk address, it sends all data received from the serial device to the GPIB.

For example, let's say your serial device is programmed to perform some calculations and you want it to return data to you.

First, address the serial device to talk by sending to the GPIB-232CT the serial device talk address. Next, perform a GPIB read of 100 bytes. The GPIB-232CT retrieves 100 bytes from its serial port receive buffer and sends them to you.

Now, find out if the serial device has sent more bytes to the GPIB-232CT by asking the GPIB-232CT to send you status information. Do this by sending the GPIB-232CT its listen address and the programming message `stat n`. Then, send the GPIB-232CT its talk address and perform a GPIB read of 20 bytes. The GPIB-232CT sends you its status information, terminated by the GPIB END message.

This status information contains the number of bytes remaining in the serial port receive buffer. This information will help you decide how much data to continue to collect from the serial device.

If the GPIB-232CT receives its talk address but has nothing to send, it will respond to GPIB reads with a carriage return and a linefeed, accompanied by END. If the GPIB-232CT receives the serial device talk address but has no data in its serial port receive buffer to send, it will wait for data from the serial device to fill the request.

The **TALK** LED on the GPIB-232CT's front panel is lit when either the GPIB-232CT is addressed to talk or the serial device is addressed to talk.

GPIB Read and Write Termination (END and EOS)

You program the GPIB-232CT and the serial device to Talk in order to send status information and serial data over the GPIB. You program the GPIB-232CT and the serial device to Listen in order to receive programming messages and serial data from the GPIB.

The IEEE-488 specification defines two ways that GPIB Talkers and Listeners can identify the last byte of data messages: END and EOS. The two methods permit a Talker to send data messages of any length without the Listener(s) knowing in advance the number of bytes in the transmission.

END message	The Talker asserts the EOI (End or Identify) signal simultaneously with transmission of the last data byte. The Listener stops reading when it detects a data message accompanied by EOI, regardless of the value of the byte.
EOS character	The Talker uses a special character at the end of its data string. The Listener stops receiving data when it detects that character. Either a 7-bit ASCII character or a full 8-bit binary byte can be used.

The two methods can be used individually or in combination. It is important that the Listener be configured to detect the end of a transmission.

When the GPIB-232CT receives its own talk or listen address, no EOS modes are in effect. When talking, the GPIB-232CT asserts EOI with the last byte of its response.

When the GPIB-232CT receives the serial device talk address, the EOS modes in effect are those that you select using the `eos` function.

Serial Port Transmission

The GPIB-232CT checks the data received from the serial device for errors, while buffering data. If a serial port error occurs, the GPIB-232CT records the appropriate error code in its status area. To determine if a serial port error has occurred, use `stat` to request the GPIB-232CT status

information. After the serial error code has been reported, it will be reset automatically so that no further action to the GPIB-232CT is necessary. You can also determine if a serial port error has occurred by performing a serial poll of the GPIB-232CT and checking the serial poll response byte to see if its ESDR bit is set. Refer to the section on serial poll responses at the end of this chapter.

G Mode - Default Settings

Tables 7-1 and 7-2 list power-on characteristics of the GPIB-232CT and the functions you can use to change those characteristics.

Table 7-1. GPIB Characteristics

Characteristic	**Power-on Value**	**Function**
End-of-string modes	none	`eos`
Allow GPIB-232CT to assert SRQ	no	`srqen`

Table 7-2. Serial Port Characteristics

Characteristic	**Power-on Value**	**Function**
Echo bytes to serial port	no	`echo`
Enable serial port communication	yes	`onl`
Baud rate	9600	`spset`
Parity	none	`spset`
Data bits	8	`spset`
Stop bits	1	`spset`

(continues)

Table 7-2. Serial Port Characteristics (continued)

Characteristic	Power-on Value	Function
Send XON/XOFF	no	`xon`
Recognize XON/XOFF	no	`xon`
Report serial errors	no	`spign`

List of G Mode Functions by Group

Tables 7-3 through 7-5 contain the programming messages that are sent to the GPIB-232CT from a GPIB Talker to configure the GPIB-232CT.

The GPIB-232CT functions are divided into three groups: GPIB Configuration functions, Serial Port Configuration functions, and General Use functions.

GPIB Configuration Functions

Table 7-3. GPIB Configuration Functions

Function	Purpose
`eos` modes, `eoschar`	Change or disable GPIB end-of-string termination mode
`srqen` mask	Set conditions for asserting SRQ

Serial Port Configuration Functions

Table 7-4. Serial Port Configuration Functions

Function	Purpose
`echo` on/off	Echo characters received from serial port
`spset` modes	Change serial port parameters
`xon` modes	Change serial port XON/XOFF protocol
`spign` on/off	Ignore serial port errors

General Use Functions

Table 7-5. General Use Functions

Function	Purpose
`id`	Identify system
`onl` on/off	Place the GPIB-232CT online/ offline
`stat` options	Return GPIB-232CT status

List of G Mode Functions in Alphabetical Order

Table 7-6 is an alphabetical list of G mode functions.

Table 7-6. GPIB-232CT Functions

Function	Purpose
`echo` on/off	Echo characters received from serial port
`eos` modes, eoschar	Change or disable GPIB end-of-string termination mode
`id`	Identify system
`onl` on/off	Place the GPIB-232CT online/offline
`spign` on/off	Ignore serial port errors
`spset` modes	Change serial port parameters
`srqen` mask	Set conditions for asserting SRQ
`stat` options	Return GPIB-232CT status
`xon` modes	Change serial port XON/XOFF protocol

Operation of the GPIB-232CT with a Serial Device

The serial device attached to the GPIB-232CT looks like any other GPIB device and, as such, is configured to respond in certain ways to GPIB commands.

Serial Poll

The GPIB-232CT will respond to a serial poll of the serial device by placing a serial poll response byte on the data lines.

Table 7-7 contains the meanings of the bits in the serial poll response byte.

Table 7-7. Serial Poll Response Byte

Bit	Mnemonic	Meaning
0	—	Not Used
1	BF	Serial port receive buffer is full
2	GERR	GPIB error - EARG, ECAP, ECMD
3	SERR	Serial error - EPAR, EORN, EFRM
4	BNE	Serial port receive buffer not empty and serial device not addressed to talk
5	EOS	EOS character received and GPIB-232CT not addressed to talk
6	RSV	Request service
7	—	Not Used

Service Request

The GPIB-232CT can be programmed to assert Service Request (SRQ) in a variety of cases. After power-on, the GPIB-232CT defaults to never asserting service request. You can program the GPIB-232CT to assert SRQ under the following circumstances:

- When the GPIB-232CT serial port receive buffer is full.
- When a GPIB-error occurs, that is, EARG, ECMD, or ECAP as reported by `stat`.

- When a serial-error occurs, that is, EPAR, EORN, or EFRM as reported by `stat`.
- When the serial device is not addressed as a Talker, and
  - any byte is received from the serial device, or
  - the EOS byte is received from the serial device.

Note: You must use the `srqen` function to enable the GPIB-232CT to assert SRQ under the conditions listed above.

Parallel Poll

The GPIB-232CT responds to a parallel poll by driving the line indicated by the PPE command true if the value of its individual status bit matches the sense bit of the PPE command and false otherwise.

Trigger

This has no effect on the GPIB-232CT.

Local

This has no effect on the GPIB-232CT.

Control

In G mode, the GPIB-232CT can act only as a Talker/Listener. It does not make sense for the GPIB-232CT to be passed control, since all programming instructions must be sent to it from another GPIB device. However, the GPIB circuitry in the GPIB-232CT will allow it to be passed control, in which case the GPIB-232CT will immediately assert ATN. This is an error condition that can lock up your system and should therefore be avoided.

Device Clear

When the GPIB-232CT receives the universal Device Clear (DCL) command or when it receives its listen addresses and the Selected Device Clear (SDC) command, it clears both its status buffer and its serial port receive buffer.

Chapter 8
G Mode Functions

This chapter contains descriptions of the G mode functions that you use to program the GPIB-232CT. These functions are in alphabetical order and are formatted to provide you with an easily usable reference.

Points to Remember

- The program examples within the function descriptions are written in Microsoft BASIC Version 3.0, using the GPIB-PC function calls of National Instruments.

- The GPIB-PC function call:

  ```
  WRT$="SPSET"+CHR$(13)
  CALL IBWRT(GPIB232%,WRT$)
  ```

  automatically sends to the GPIB-232CT its talk address, and the programming message `spset`, followed by a carriage return. If you are not using the National Instruments GPIB-PC software, be sure your program properly addresses the GPIB-232CT and the serial device when writing to and reading from them.

- In the function syntax descriptions, arguments shown in square brackets (`[]`) are optional. Do not enter the brackets as part of your argument.

- Terminate each programming message with a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>). This is denoted by <CR> in the syntax portions of the function descriptions and by `CHR$(13)` in the BASIC examples.

- To send more than one programming message per GPIB write operation, embed a <CR> or an <LF> in the data string you send. For example, to send the two programming messages "change serial port parameters" and "change serial port XON/XOFF protocol," you could either send two separate strings to the GPIB in two separate GPIB writes:

```
WRT$="SPSET 1200,n,8"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
WRT2$="XON 1,1"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
```

or, you could put both messages in one string and send it to the GPIB-232CT in one GPIB write:

```
WRT$="SPSET 1200,n,8"+CHR$(13)+"XON
1,1"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
```

- It is necessary for you to send only enough characters of the function name to distinguish it from other functions. These characters are shown in **boldface** in the syntax portion of the function descriptions.

echo - Echo Characters Received from Serial Port

Type: Serial Port Configuration function

Syntax: **ec**`ho [bool]`<CR>

Purpose: You use `echo` when a terminal is connected to the GPIB-232CT and you wish to display what you type on the screen of the terminal.

Remarks: If the argument `bool` is 1, characters received from the serial port are echoed back to the serial port. If the argument `bool` is 0, characters are not echoed.

If you call `echo` without an argument, the GPIB-232CT returns the current setting.

By default, echoing is disabled.

In a debugging environment where the success of your communication with the serial device is unclear, you could connect a terminal to the GPIB-232CT instead of connecting the serial device to the GPIB-232CT. Now the data that the GPIB-232CT normally sends to the serial device is displayed on the terminal screen. Also, you can type characters on the terminal to send to the GPIB-232CT, just as your serial device sends characters to the GPIB-232CT. With `echo` enabled, the GPIB-232CT echoes back to the terminal what you type, so that you can verify that what the GPIB-232CT receives is exactly what you type.

Examples:

1.
```
WRT$="echo 1"+CHR$(13)          'Enable character
CALL IBWRT(GPIB232%,WRT$)       'echoing.
```

2.
```
WRT$="ECHO 0"+CHR$(13)          'Disable character
CALL IBWRT(GPIB232%,WRT$)       'echoing.
```

3.
```
WRT$="echo"+CHR$(13)            'What is the
CALL IBWRT(GPIB232%,WRT$)       'current echo
CALL IBRD(GPIB232%,RESP$)       'status?
                                'RESP$ contains
                                '0<CR><LF>
                                '(character echo is
                                'disabled).
```

eos - Change/Disable GPIB EOS Termination Mode

Type: GPIB Configuration function

Syntax: **eos** [[X][B]eoschar]<CR>

or

eos D<CR>

Purpose: You use eos to enable the GPIB-232CT to add the GPIB END message to the data string sent by the serial device when the data string contains the specified end-of-string character.

Remarks: eos applies only when the GPIB-232CT has received the serial device talk address and is sending serial data to the GPIB. It does not apply when the GPIB-232CT reads serial data or programming messages from the GPIB. The arguments X, B, and D specify GPIB data transfer termination methods.

The argument eoschar is a numeric string representing a single ASCII character that is to be the EOS byte. The arguments X and B are used to enable the corresponding EOS mode. The argument D disables all EOS modes.

Table 8-1. Data Transfer Termination Methods

Description	Letter
XEOS - set EOI with EOS when sending data from serial device.	X
BIN - compare all 8 bits of EOS byte rather than low 7 bits.	B
DISABLE - disable all EOS modes.	D

If Methods `X`, or `X` and `B` are chosen, the GPIB-232CT automatically sends the END message along with `eoschar` when performing GPIB writes of serial data. That is, when the GPIB-232CT receives `eoschar` over the serial port and sends it on to the GPIB, it will also assert EOI along with that byte. When `X` alone is chosen, END is sent with the EOS byte when the low seven bits of that byte match the low seven bits of `eoschar`. When `X` and `B` are chosen, a full 8-bit comparison is used.

If `B` is the only mode chosen, the EARG error is posted.

If `D` is chosen, all EOS modes are disabled.

By default, all EOS modes are disabled.

If you call `eos` without an argument, the GPIB-232CT returns to you the current `eos` settings.

The assignment made by this function remains in effect until you call `eos` again, you call `onl`, or you turn the GPIB-232CT off.

See Also: Chapter 7, the section entitled *GPIB Read and Write Data Termination*, and *Operation of the GPIB-232CT as a Serial Device;* and the `srqen` function in this chapter.

Examples:

```
1. WRT$="EOS X,13"+CHR$(13)      'Send EOI with <CR>.
   CALL IBWRT(GPIB232%,WRT$)     'Compare 7 bits.

2. WRT$="eos"+CHR$(13)           'What are the
   CALL IBWRT(GPIB232%,WRT$)     'current EOS
                                 'settings?
   CALL IBRD(GPIB232%,RESP$)     'RESP$ contains
                                 'X,13<CR><LF>.
```

id - Identify System

Type: General Use function

Syntax: **id**<CR>

Purpose: You use `id` if you wish to know the revision level of your software, or if you wish to know how much RAM is installed in your GPIB-232CT.

Remarks: The identification is returned in three strings. The first two strings identify the company product model, the software revision level, and a copyright notice. The third string identifies the number of bytes of RAM in the GPIB-232CT.

Examples:

1.
```
WRT$="id"+CHR$(13)     'Get system
                       'identification.
CALL IBWRT(GPIB232%,WRT$)
```

response: **GPIB-232CT, Rev. A.0<CR><LF>**
(c)1988 National Instruments<CR><LF>
64K bytes RAM<CR><LF>

onl - Place the GPIB-232CT Online/Offline

Type: General Use function

Syntax: **o**`nl [bool]`<CR>

Purpose: You use `onl` to disable communication between the GPIB-232CT and the serial port, or to re-initialize the GPIB-232CT characteristics to their default values.

Remarks: If the argument `bool` is 1, the GPIB-232CT places itself online. If the argument `bool` is 0, the GPIB-232CT places itself offline. By default, the GPIB-232CT powers up online.

If you call `onl` without an argument, the GPIB-232CT returns the current status of the GPIB-232CT, which is 0 if the GPIB-232CT is offline and 1 if the GPIB-232CT is online.

Placing the GPIB-232CT offline can be thought of as disconnecting its serial cable from the serial device. When placed offline, no data is sent out the GPIB-232CT serial port and data coming in to the GPIB-232CT serial port is not saved until `onl` is called with `bool=1`.

Placing the GPIB-232CT online enables data to be sent and received over its serial port. Placing the GPIB-232CT online also restores all GPIB-232CT settings to their power-on values.

See Also: Tables 7-1 and 7-2 for the GPIB-232CT power-on settings.

Examples:

```
1. WRT$="onl 1"+CHR$(13)          'Put the GPIB-232CT
   CALL IBWRT(GPIB232%,WRT$)      'online, and restore
                                  'its power-on
                                  'settings.

2. WRT$="ONL 0"+CHR$(13)          'Put the GPIB-232CT
   CALL IBWRT(GPIB232%,WRT$)      'offline.
```

spign - Ignore Serial Port Errors

Type: Serial Port function

Syntax: **spi**gn [bool]<CR>

Purpose: You use spign at the beginning of your program if you wish to change the effect that serial port errors have on the storage of a character received with a serial error. This function tells the GPIB-232CT to ignore or not to ignore the occurrence of serial port errors. By default, the GPIB-232CT ignores serial port errors.

Remarks: If the argument bool is 0, the GPIB-232CT will not ignore serial port errors. When bool is 0, the GPIB-232CT does not store characters that contain serial errors. A list of serial port errors are listed in Appendix B.

The serial port errors include parity, overrun, framing, and overflow errors.

If the argument bool is 1, the GPIB-232CT stores all serial port errors occur. Also, no serial error code will be reported by the stat function.

Examples:

1.
```
WRT $,"spign 0"+CHR$(13)     'Do not ignore
CALL IBWRT(GPIB232%,WRT$)    'serial port errors.
```

2.
```
WRT $,"spign 1"+CHR$(13)     'Ignore serial port
CALL IBWRT(GPIB232%,WRT$)    'errors.
```

spset - Change Serial Port Parameters

Type: Serial Port Configuration function

Syntax: **sps**et [baud] [parity] [databits] [stopbits]<CR>

Purpose: You use spset at the beginning of your program to set up the GPIB-232CT serial port characteristics (baud rate, parity, data bits, and stop bits) to those required by your serial device.

Remarks: The argument baud is a numeric string specifying the baud rate (300, 600, 1200, 2400, 4800, 9600, 19200, 38400).

The argument parity is a character specifying the parity (e for even, o for odd, n for none).

The argument databits is a character specifying the number of data bits (7 or 8).

The argument stopbits is a character specifying the number of stop bits (1 or 2).

Until you call spset the following characteristics are in effect:

```
9600 n 8 1
```

meaning the baud rate is 9600 bits per second (bps), parity is disabled, there are 8 data bits and 1 stop bit.

If you must reconfigure the GPIB-232CT's serial port, do so only when communication with the serial port is not taking place.

If you call spset without an argument, the GPIB-232CT returns to you its current serial port configuration.

Examples:

1.
```
REM Set up the serial port of GPIB-232CT to keep
REM its current baud rate, current parity, and
REM to use 7 data bits and 2 stop bits.
WRT$="spset 7 2"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
```

2.
```
REM What are the current GPIB-232 serial port
REM settings?
WRT$="SPSET"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
REM RESP$ will contain 19200,E,7,2<CR><LF>
REM (19200 baud, even parity, 7 data bits,
REM 2 stop bits).
CALL IBRD(GPIB232%,RESP$)
```

3.
```
REM Set the GPIB-232CT serial port to 1200 baud,
REM no parity, 8 data bits, and 1 stop bit.
WRT$="spset 1200 n 8 1"+CHR$(13)
CALL IBWRT(GPIB232%,WRT$)
```

srqen - Enable/Disable Setting of SRQ

Type: GPIB Configuration function

Syntax: **`srq`**`en [mask]`<CR>

Purpose: You use `srqen` when you wish to allow the GPIB-232CT to assert SRQ under the conditions described in Chapter 7 in the section entitled *Operation of the GPIB-232CT as a Serial Device*.

Remarks: When the argument `mask` is 0, the GPIB-232CT will never assert SRQ. When the argument `mask` is > 0, the GPIB-232CT will assert SRQ under the conditions represented by each bit in the `mask`. The `mask` bits are as follows:

Table 8-2. SRQ Mask Bits

Bit	Hex Value	Decimal Value	Mne-monic	Description
0	1	1	—	Not used
1	2	2	BF	Serial port receive buffer full and serial device not addressed to talk
2	4	4	GERR	GPIB error - (see `stat`)
3	8	8	SERR	Serial error - (see `stat`)
4	10	16	BNE	Serial port receive buffer not empty and serial device not addressed to talk
5	20	32	EOS	EOS character received and serial device not addressed to talk
6	40	64	—	Not used
7	80	128	—	Not used

To determine the `mask` value you want, add up the bit values of each of the conditions on which you want SRQ to be asserted. For example, if you want SRQ asserted on GPIB errors and serial port errors you will call `srqen` with a `mask` of 12 (4 for GERR and 8 for SERR).

The power on default of `srqen` is disabled, that is, SRQ will never be asserted.

If you call `srqen` without an argument, the GPIB-232CT returns a decimal string that indicates the decimal value of the current setting.

See Also: `eos`.

Examples:

1.
```
WRT$="srqen 0"+CHR$(13)      'Never assert SRQ.
CALL IBWRT(GPIB232%,WRT$)
```

2.
```
WRT$="srqen 4"+CHR$(13)      'Assert SRQ when
CALL IBWRT(GPIB232%,WRT$)    'ENOL occurs.
```

stat - Return GPIB-232CT Status

Type: General Use function

Syntax: **st**at [[c] n]<CR>
or
stat [c] s<CR>
or
stat [c] n s<CR>

Purpose: You use stat to obtain the status of the GPIB-232CT to see if certain conditions are currently present. You use stat most often to see if the previous operation resulted in an error.

Remarks: The GPIB-232CT returns status information to you in a form depending on the mode or combination of modes you chose. n indicates that the status information will be returned as numeric strings. s indicates that the status information will be returned in symbolic format, that is, as mnemonic strings. c specifies that the status will be returned after each programming message, eliminating the need to call stat after each programming message.

Normally, you use s or symbolic format only when you are debugging your code and you want to print the mnemonic for each piece of status information.

The status information returned by the GPIB-232CT contains four pieces of information: the GPIB-232CT status, a GPIB-error code, a serial-error code, and a count. The GPIB-232CT returns a <CR><LF> following each piece of the response. It asserts EOI with the final <LF> that comes after count.

Status represents a combination of GPIB-232CT conditions. Internally in the GPIB-232CT, status is stored as a 16-bit integer. Each bit in the integer represents a single condition. A bit value of 1 indicates the corresponding condition is in effect; a bit value of 0 indicates the condition is not in effect. Since more than one GPIB-232CT condition can exist at one time, more than one bit can be set in status. The highest order bit of status, also called the sign bit, is set when the GPIB-232CT detects either a GPIB error or a serial port error.

Consequently, when status is negative, an error condition exists; when status is positive, no error condition exists.

GPIB-error represents a single GPIB error condition present.

serial-error represents a single serial error condition present.

count is the number of bytes currently contained in the GPIB-232CT's serial port receive buffer.

Table 8-3. GPIB-232CT Status Conditions

Numeric Value (n)	**Symbolic Value (s)**	**Description**	**Bit**
-32768	ERR	Error detected	15
16384	—	Reserved	14
8192	—	Reserved	13
4096	—	Reserved	12
2048	—	Reserved	11
1024	—	Reserved	10
512	—	Reserved	9
256	CMPL	Operation completed	8
128	—	Reserved	7
64	—	Reserved	6
32	—	Reserved	5
16	—	Reserved	4
8	—	Reserved	3

(continues)

Table 8-3. GPIB-232CT Status Conditions (continued)

Numeric Value (n)	Symbolic Value (s)	Description	Bit
4	—	Reserved	2
2	—	Reserved	1
1	—	Reserved	0

Table 8-4. GPIB Error Conditions

Numeric Value (n)	Symbolic Value (s)	Description
0	NGER	No GPIB error condition to report
1	—	Reserved
2	—	Reserved
3	—	Reserved
4	EARG	Invalid argument or arguments
5	—	Reserved
6	—	Reserved
7-10	—	Reserved
11	ECAP	No capability for operation
12-16	—	Reserved
17	ECMD	Unrecognized command

Table 8-5. Serial Error Conditions

Numeric Value (n)	Symbolic Value (s)	Description
0	NSER	No serial port error condition to report
1	EPAR	Serial port parity error
2	EORN	Serial port overrun error
3	EOFL	Serial port receive buffer overflow
4	EFRM	Serial port framing error

A detailed description of the conditions under which each bit in status is set or cleared can be found in Appendix B.

The GPIB-232CT updates `status` and `count` at the end of each programming message. It updates `GPIB-error` and `serial-error` whenever a new error occurs. `GPIB-error` and `serial-error` are cleared only after you have requested status.

Refer to the following examples for ways in which to use the status information.

When you wish to begin continuous status reporting, send the `stat c s`, `stat c n`, or `stat c n s` programming message. When you call `stat` with both `n` and `s` modes specified, the numeric status is always returned first.

If you call `stat` without an argument, continuous status reporting is disabled.

Examples:

1.
```
 10  REM Tell GPIB-232CT to send us numeric
 20  REM status.
 30  WRT$="stat n"+CHR$(13)
 40  CALL IBWRT(GPIB232%,WRT$)
 50  REM Now read the status from the
 60  REM GPIB-232CT.
 70  STATUS$=SPACE$(10) : GPIBERR$=SPACE$(10)
 80  SPERR$=SPACE$(10) : COUNT$=SPACE$(10)
 90  CALL IBRD(GPIB232%,STATUS$)
100  REM Read up to 10 bytes of status.
110  REM The GPIB-232CT returns 4 pieces of
120  REM status, one for each IBRD.  We are set
130  REM up to terminate read on linefeed, which
140  REM is what terminates each piece of status.
150  STATUS% = VAL(STATUS$)
160  CALL IBRD(GPIB232%,GPIBERR$)  'Read GPIB-error.
170  CALL IBRD(GPIB232%,SPERR$)    'Read serial-error.
180  CALL IBRD(GPIB232%,COUNT$)    'Read count.
190  REM Call error routine at 500 if error occurred.
200  IF STATUS% < 0 THEN GOTO 500
   ...
500  REM Print GPIB-error, and serial-error
510  REM values to determine what errors occurred.
520  PRINT "GPIB-error = ";GPIBERR$
530  PRINT "serial-error = ";SPERR$
540  STOP
```

2.
```
 10  REM Turn on continuous status reporting,
 20  REM in numeric format.
 30  WRT$="stat c n"
 40  CALL IBWRT(GPIB232%,WRT$)
 50  REM If we have 3 bytes in the serial port
 60  REM buffer, a typical response would be:
 70  REM 262<CR><LF>0<CR><LF>0<CR><LF>3<CR><LF>
 80  REM Read the GPIB-232CT status; read 30
 90  REM bytes or until EOI is received.
100  RD$=SPACE$(30)
110  CALL IBRD(GPIB232%,RD$)
120  REM Print the status information.
130  PRINT "GPIB-232CT status is: ";RD$
```

3.
```
 10  REM Turn on continuous status reporting,
 20  REM in symbolic format.
 30  WRT$="stat c s"
 40  CALL IBWRT(GPIB232%,WRT$)
 50  REM Read the GPIB-232CT status; read 50
 60  REM bytes or until EOI is received.
 70  RD$=SPACE$(50)
 80  CALL IBRD(GPIB232%,RD$)
 90  REM Print the status information.
100  PRINT "GPIB-232CT status is: ";RD$
```

```
Printed information is:

GPIB-232CT status is : CMPL
NGER
NSER
3
```

xon - Change Serial Port XON/XOFF Protocol

Type: Serial Port Configuration function

Syntax: **x**`on [booltx][,boolrx]`<CR>

Purpose: You use `xon` at the beginning of your program when your serial device uses the XOFF/XOFF protocol. `xon` allows the GPIB-232CT to communicate over the serial port using the same protocol as your serial device.

Remarks: The argument `booltx` specifies whether to enable the XON/XOFF protocol when sending data out on the serial port. When `booltx` is enabled, the GPIB-232CT monitors its serial receive buffer for XON/XOFF characters as it sends data over the serial port. If it receives the XOFF character (decimal 19 or <CTRL>s), it immediately stops sending data. When it receives the XON character (decimal 17 or <CTRL>q), it begins sending data again.

The argument `boolrx` specifies whether to enable the XON/XOFF protocol when receiving data over the serial port. When `boolrx` is enabled, as the GPIB-232CT receives data over the serial port, it sends XOFF over the serial port (if its serial port recive buffer is almost full). This tells the sender to stop sending data. When the GPIB-232CT has room to safely receive more bytes, the GPIB-232CT sends XON over the serial port. This tells the sender to begin sending data again.

Use XON/XOFF if your serial device does not recognize the hardware handshake protocol, and you are transferring large amounts of data at high speeds. Without it, you are in danger of overflowing the serial port receive buffers of the GPIB-232CT and the serial device. Refer to your serial device's documentation to see if it uses a hardware handshake.

Why would you want to enable the protocol in one case and not the other? Some serial devices use XON/XOFF protocol when transmitting data but not when receiving. In this case, you might configure the GPIB-232CT using the second example.

The power-on default is that XON/XOFF for both cases is disabled.

If you call `xon` without an argument, the GPIB-232CT returns to you the current settings (1=protocol enabled, 0=protocol disabled).

Examples:

```
1. WRT$="XON 1,1"+CHR$(13)      'Enable GPIB-232CT
   CALL IBWRT(GPIB232%,WRT$)    'XON/XOFF protocol
                                'for TX and RX.

2. WRT$="XON 0,1"+CHR$(13)      'Disable protocol on
   CALL IBWRT(GPIB232%,WRT$)    'TX; enable protocol
                                'on RX.

3. WRT$="XON"+CHR$(13)          'Return current settings.
   CALL IBWRT(GPIB232%,WRT$)
   CALL IBRD(GPIB232%,RESP$)    'RESP$ contains
                                '0,1<CR><LF>
                                '(transmit protocol
                                'disabled, receive
                                'protocol enabled).
```

Appendix A
Multiline Interface Messages

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

For more information on these messages, refer to the IEEE-488 Std. 488-1978, *IEEE Standard Digital Interface for Programmable Instrumentation*.

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US		3F	077	63	?	UNL

Message Definitions

DCL	Device Clear	MSA	My Secondary Address
GET	Group Execute Trigger	MTA	My Talk Address
GTL	Go To Local	PPC	Parallel Poll Configure
LLO	Local Lockout	PPD	Parallel Poll Disable
MLA	My Listen Address		

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

PPE	Parallel Poll Enable	SPE	Serial Poll Enable
PPU	Parallel Poll Unconfigure	TCT	Take Control
SDC	Selected Device Clear	UNL	Unlisten
SPD	Serial Poll Disable	UNT	Untalk

Interface Message Reference List

Mnemonic	Message	Interface Function(s)
LOCAL MESSAGES RECEIVED (by interface functions)		
gts	go to standby	C
ist	individual status qualifier	PP
lon	listen only	L, LE
[lpe]	local poll enable	PP
ltn	listen	L, LE
lun	local unlisten	L, LE
nba	new byte available	SH
pon	power on	SH, AH, T, TE, L, LE, SR, RL, PP, C
rdy	ready	AH
rpp	request parallel poll	C
rsc	request system control	C
rsv	request service	SR
rtl	return to local	RL
sic	send interface clear	C
sre	send remote enable	C
tca	take control asynchronously	C
tcs	take control synchronously	AH, C
ton	talk only	T, TE
REMOTE MESSAGES RECEIVED		
ATN	attention	SH, AH, T, TE, L, LE, PP, C
DAB	data byte	(via L, LE)
DAC	data accepted	SH
DAV	data valid	AH
DCL	device clear	DC
END	end	(via L, LE)
GET	group execute trigger	DT
GTL	go to local	RL
IDY	identify	L, LE, PP
IFC	interface clear	T, TE, L, LE, C
LLO	local lockout	RL
MLA	my listen address	L, LE, RL
[MLA]	my listen address	T
MSA or [MSA]	my secondary address	TE, LE
MTA	my talk address	T, TE
[MTA]	my talk address	L
OSA	other secondary address	TE
OTA	other talk address	T, TE
PCG	primary command group	TE, LE, PP

Interface Message Reference List (continued)

Mnemonic	Message	Interface Function(s)
REMOTE MESSAGES RECEIVED (continued)		
PPC	parallel poll configure	PP
[PPD]	parallel poll disable	PP
[PPE]	parallel poll enable	PP
PPRn	parallel poll response n	(via C)
PPU	parallel poll unconfigure	PP
REN	remote enable	RL
RFD	ready for data	SH
RQS	request service	(via L, LE)
[SDC]	selected device clear	DC
SPD	serial poll disable	T, TE
SPE	serial poll enable	T, TE
SRQ	service request	(via C)
STB	status byte	(via L, LE)
TCT or [TCT]	take control	C
UNL	unlisten	L, LE
REMOTE MESSAGES SENT		
ATN	attention	C
DAB	data byte	
DAC	data accepted	AH
DAV	data valid	SH
DCL	device clear	(via C)
END	end (via T)	
GET	group execute trigger	(via C)
GTL	go to local	(via C)
IDY	identify	C
IFC	interface clear	C
LLO	local lockout	(via C)
MLA or [MLA]	my listen address	(via C)
MSA or [MSA]	my secondary address	(via C)
MTA or [MTA]	my talk address	(via C)
OSA	other secondary address	(via C)
OTA	other talk address	(via C)
PCG	primary command group	(via C)
PPC	parallel poll configure	(via C)
[PPD]	parallel poll disable	(via C)
[PPE]	parallel poll enable	(via C)
PPRn	parallel poll response n	PP
PPU	parallel poll unconfigure	(via C)
REN	remote enable	C
RFD	ready for data	AH

Interface Message Reference List (continued)

Mnemonic	Message	Interface Function(s)
REMOTE MESSAGES SENT (continued)		
RQS	request service	T, TE
[SDC]	selected device clear	(via C)
SPD	serial poll disable	(via C)
SPE	serial poll enable	(via C)
SRQ	service request	SR
STB	status byte	(via T, TE)
TCT	take control	(via C)
UNL	unlisten	(via C)
UNT	untalk	(via C)

Appendix B
Status and Message Information

This appendix describes the status and error information that the GPIB-232CT records as it executes each programming message. Items that apply to S mode are marked with an S. Items that apply to G mode are marked with a G. The number preceding each description is the numeric value of that bit in the status word or of the error code.

Status Bits

The following paragraphs describe the conditions represented by the bits in status.

ERR S/G -32768

The ERR bit is set in `status` following any call that results in an error; the particular error can be determined by examining the `GPIB-error` and `serial-error` values. The ERR bit is cleared following any call that does not result in an error.

By examining this bit, you can check for an error condition after each call. An error made early in your application program may not become apparent until a later instruction. At that time, the error can be more difficult to locate.

TIMO S 16384

The TIMO bit specifies whether a timeout has occurred. The TIMO bit is set in the status word following a call to `wait` if the TIMO bit of the wait `mask` parameter is also set and if the wait has exceeded the time limit value that is set by the `tmo` call. The TIMO bit is also set following a call to any of the I/O functions (for example, `rd`, `wrt`, and `cmd`), if a timeout occurs during a call. The TIMO bit is cleared in the status word in all other circumstances.

END S 8192

The END bit specifies whether the END or EOS message has been received. The END bit is set in the status word following a `rd` function if the END or EOS message was detected during the read. While the GPIB-232CT is performing a shadow handshake as a result of the `gts` function, any other function call can return a status word with the END bit set if the END or EOS message occurred before or during that call. The END bit is cleared in the status word at the start of any subsequent programming message.

SRQI S 4096

The SRQI bit specifies whether a device is requesting service. This bit is set in the status word whenever the SRQ line is asserted. The bit is cleared whenever the GPIB SRQ line is unasserted.

CMPL S/G 256

The CMPL bit specifies that the operation relating to this status information is complete. This bit is always set, and is useful in identifying the status word from other responses.

LOK S 128

The LOK bit specifies whether the GPIB-232CT is in a lockout state. The LOK bit is set whenever the GPIB-232CT detects the Local Lockout (LLO) message has been sent either by the GPIB-232CT or by another Controller. The LOK bit is cleared when the Remote Enable (REN) GPIB line becomes unasserted either by the GPIB-232CT or by another Controller.

REM S 64

The REM bit specifies whether the GPIB-232CT is in remote state. The REM bit is set whenever the Remote Enable (REN) GPIB line is asserted and the GPIB-232CT detects its listen address has been sent either by the GPIB-232CT or by another Controller. The REM bit is cleared whenever REN becomes unasserted, or when the GPIB-232CT as a Listener detects the Go to Local (GTL) command has been sent either by the GPIB-232CT or by another Controller, or when the `LOC` function is called while the LOK bit is cleared in status.

CIC S 32

The CIC bit specifies whether the GPIB-232CT is the Controller-In-Charge. The CIC bit is set whenever `sic` is called while the GPIB-232CT is System Controller, or when another Controller passes control to the GPIB-232CT. The CIC bit is cleared whenever the GPIB-232CT detects Interface Clear (IFC) from some other device that is System Controller, or when the GPIB-232CT passes control to another device.

ATN S 16

The ATN bit specifies the state of the GPIB Attention (ATN) line. The ATN bit is set whenever the GPIB ATN line is asserted and cleared when the ATN line is unasserted.

TACS S 8

The TACS bit specifies whether the GPIB-232CT has been addressed as a Talker. The TACS bit is set whenever the GPIB-232CT detects that its talk address (and secondary address, if enabled) has been sent either by the GPIB-232CT itself or by another Controller. The TACS bit is cleared whenever the GPIB-232CT detects the Untalk (UNT) command, a talk address other than its own, its own listen address, or Interface Clear (IFC).

LACS S 4

The LACS bit specifies whether the GPIB-232CT has been addressed as a Listener. The LACS bit is set whenever the GPIB-232CT detects that its listen address (and secondary address, if enabled) has been sent either by the GPIB-232CT itself or by another Controller. The LACS bit is also set whenever the GPIB-232CT shadow handshakes as a result of the `gts` function. The LACS bit is cleared whenever the GPIB-232CT detects that the Unlisten (UNL) command, its own talk address, Interface Clear (IFC), or `gts` is called without shadow handshake.

DTAS S 2

The DTAS bit specifies whether the GPIB-232CT has detected a device trigger command. The DTAS bit is set whenever the GPIB-232CT as a Listener, detects the Group Execute Trigger (GET) command has been sent by another Controller. The DTAS bit is cleared in status at the start of any subsequent programming message.

DCAS S 1

The DCAS bit specifies whether the GPIB-232CT has detected a device clear command. The DCAS bit is set whenever the GPIB-232CT detects the Device Clear (DCL) command has been sent by itself or by another Controller, or whenever the GPIB-232CT as a Listener detects the Selected Device Clear (SDC) command has been sent by itself or by another Controller. The DCAS bit is cleared in status at the start of any subsequent programming message.

In addition to the previously described conditions, the following situations also affect the bits in status:

- A call to the `onl` function clears the following bits:

  - END
  - LOK
  - REM
  - CIC
  - TACS
  - LACS
  - DTAS
  - DCAS

- A call to `onl` affects bits other than those listed here according to the rules explained for each bit.

GPIB Error Codes

When the ERR bit is set in status, a GPIB error or a serial port error has occurred. The error code is indicated by `GPIB-error` or `serial-error`.

The following paragraphs describe the `GPIB-errors` in detail.

NGER S/G 0

The GPIB-232CT reports this error when GPIB-232CT detected no GPIB errors as a result of the last operation.

ECIC S 1

The GPIB-232CT records this error when you call a function that requires that the GPIB-232CT be CIC and it is not CIC.

In cases when the GPIB-232CT should always be the Controller-In-Charge, the remedy is to be sure to call `sic` to send Interface Clear before attempting any of these calls, and to avoid sending the command byte TCT (hex 09, Take Control). In multiple Controller-In-Charge situations, the remedy is to always be certain that the CIC bit appears in status before attempting these calls. If it is not, you can call `wait` (CIC) to delay further processing until control is passed to the GPIB-232CT.

ENOL S 2

The most common cause of this error is that the GPIB-232CT attempted to write to the GPIB when no Listeners were addressed.

The remedy is to be sure that the proper listen address is in the `alist` argument string, to use `cmd` to properly address the Listeners, or to be sure some other controller has addressed the Listeners before you call `wrt`.

This error may occur more rarely in situations in which the GPIB-232CT is not the Controller-In-Charge and the Controller asserts ATN before the write call in progress has ended. The remedy is either to reduce the write byte count to that which is expected by the Controller, or to resolve the situation on the Controller's end.

EADR S 3

The GPIB-232CT records this error when it is not addressed to listen or to talk before read and write calls when it is not the Controller-In-Charge. The remedy is to be sure that the Controller addresses the GPIB-232CT to talk or listen before attempting the `wrt` or `rd`.

The GPIB-232CT also records this error during the function `gts` when the shadow-handshake feature is requested and the GPIB ATN line is already unasserted. In this case, the shadow handshake is not possible and the error is recorded to notify you of that fact. `gts` should almost never be called except immediately after a `cmd` call. (`cmd` causes ATN to be asserted.)

EARG S/G 4

The GPIB-232CT records this error when you pass an invalid argument to a function call. The following are some examples:

- `tmo` called with a value not in the range .00001 to 3600.
- `sic` called with a value not in the range .0001 to 3600.
- `eos` called with meaningless termination method identifiers.
- `caddr` called with the value 31.
- `ppc` called with illegal parallel poll configurations.

If your programming message contains more than one argument and you get this error, the GPIB-232CT discards all arguments and does not perform the function.

This also can be caused by a transmission error that corrupts the argument portion of the programming message or that corrupts the <CR> or <LF> that terminates the programming message. Use `stat` and check `serial-error` to determine if a transmission error has occurred.

ESAC S 5

The GPIB-232CT records this error when `sic` or `sre` is called when the GPIB-232CT does not have System Controller capability. The remedy is to give the GPIB-232CT that capability by calling `rsc`. (At power on, the GPIB-232CT assumes itself to be the System Controller.)

EABO S 6

The GPIB-232CT records this error when I/O has been cancelled. By far the most common cause of this error is a timeout condition.

To remedy a timeout error, if I/O is actually progressing but times out anyway, lengthen the timeout period with `tmo`. More frequently, however, the I/O is stuck (the Listener is not continuing to handshake or the Talker has stopped talking), or the byte count in the call that timed out was more than the other device was expecting. Be sure that both parties to the transfer understand what byte count is expected; or if possible, have the Talker use the END message to assist in early termination.

ECAP S/G 11

The GPIB-232CT records this error when your programming message contains an S mode function and the GPIB-232CT is configured for G mode, or your programming message contains a G mode function and the GPIB-232CT is configured for S mode.

ECMD S/G 17

The GPIB-232CT records this error when your programming message received by the GPIB-232CT does not contain a recognizable function name. This can happen if the function name is misspelled or if a transmission error occurred that resulted in the function name being corrupted. Check your function name spelling, and check `serial-error` to see if a serial port error has been posted.

Serial Port Error Codes

The following paragraphs describe the serial port errors in detail.

In S mode, when a serial port error occurs as the GPIB-232CT receives a programming message, the GPIB-232CT posts the error and discards the message. If a serial port error occurs in the middle of a data stream following a `cmd` or `wrt` function, the GPIB-232CT discards that data byte and all subsequent data bytes. You can use the `spign` function to tell the GPIB-232CT to ignore all serial port errors.

NSER 0

The GPIB-232CT reports this error when the GPIB-232CT detected no serial port error as a result of the last operation.

EPAR 1

The GPIB-232CT records this error when the parity of the received character is not what was expected. This means that 1 or more bits of the received character were corrupted in such a way as to change the character's parity.

EORN 2

The GPIB-232CT records this error when characters arrive at the serial port faster than the serial port can accept them. When this error occurs, one or more characters sent to the serial port have been lost. If this error occurs, check to see that the GPIB-232CT and your serial device are using the same serial port settings.

EOFL 3

The GPIB-232CT records this error when the GPIB-232CT's internal serial port buffer overflows. This should only occur if XON/XOFF is disabled and there is no hardware handshake in effect.

EFRM 4

The GPIB-232CT records this error when a character is received whose stop bits are not in the expected place. This can happen when the number-of-bits-per-character setting of the GPIB-232CT does not match your serial device. It can also happen if the baud rates of the GPIB-232CT and your serial device do not match, or if one side of the serial link does not use parity and the other side does.

Appendix C
The Serial Connection

The serial port on the GPIB-232CT provides an asynchronous serial communication link between the GPIB-232CT and a serial peripheral device. The connector for the port is located on the rear panel.

This appendix is a review of the RS-232 standard at the physical and electrical levels. This information will be helpful if you decide to build your own cable. If you would like more information on the RS-232 standard, write to:

EIA Engineering Department
Standards Sales Office
2001 Eye Street, N.W.
Washington, D.C. 20006

RS-232C

The RS-232C standard (international standard CCITT V.24) was formulated in 1969 largely from the efforts of the Electronic Industries Association (EIA) and Bell Laboratories. The standard describes the electrical specifications and arrangement of control and data signals on both sides of a serial communications interface. Its original intent was to interface terminals to modems. Since then, many manufacturers of computers and instruments have adopted the standard for their serial communications needs.

The RS-232 serial port on the GPIB-232CT uses a 25-pin, D-subminiature connector with a DTE (Data Terminal Equipment) interface configuration. That is, the GPIB-232CT transmits data on pin 2 and receives data on pin 3. Table C-1 shows the signal lines supported on the GPIB-232CT.

Table C-1. RS-232 Serial Port Pinouts

RS-232 Name	Pin	Signal Name	Function
AA	1	PG (Protective Ground)	This line is connected to the chassis ground of the GPIB-232CT.
BA	2	TxD (Transmit Data)	This line carries serial data from the GPIB-232CT to the serial host.
BB	3	RxD (Receive Data)	This line carries serial data from the serial host to the GPIB-232CT.
CA	4	RTS (Request to Send)	This signal line is asserted by the GPIB-232CT when it is ready to accept serial data over the RS-232. RTS is unasserted when the GPIB-232CT is no longer ready to accept serial data.
CB	5	CTS (Clear to Send)	This signal line is asserted by the serial host and sensed by the GPIB-232CT. It indicates that the serial host is ready to accept data. When CTS is unasserted, data transmission from the GPIB-232CT is disabled.
AB	7	SG (Signal Ground)	This line establishes a reference point for all interface voltages.
CD	20	DTR (Data Terminal Ready	This signal line is asserted by the GPIB-232CT when it is powered on. It can be used to notify the other serial device that the GPIB-232CT has been powered on, or it could be used to *wrap back* to the GPIB-232CT CTS input if no hardware handshake is used.

To interface other products to your GPIB-232CT RS-232 serial port, consult your serial device manual to determine if the device is configured as a DTE or DCE. Also, note how the control lines are used and whether they must be driven for the serial port to operate. The GPIB-232CT's CTS line must be driven either by the GPIB-232CT's DTR line or by a signal line from the serial device used to implement a hardware handshake for data. The following discussions contain some cabling examples from the GPIB-232CT to a serial host device.

Interfacing the GPIB-232CT to a DCE

A correctly configured DTE/DCE interface is wired straight across: GPIB-232CT pin 2 to DCE pin 2, pin 3 to pin 3, and so forth. Wired in this manner, the GPIB-232CT can then interact to function properly (handshake). Figure C-1 shows a properly configured DTE-to-DCE cable.

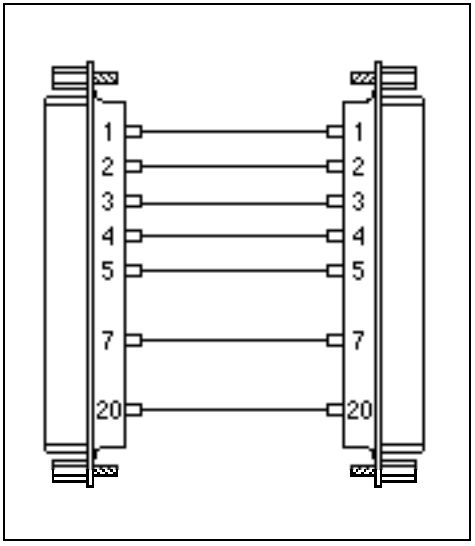


Figure C-1. DTE-to-DCE Cable Configuration

If your serial device does not use the same hardware handshaking protocol as the GPIB-232CT, you can either use a minimum configuration cable and use XON/XOFF handshaking (if necessary), or wire a custom cable that will satisfy the GPIB-232CT hardware handshaking protocol. Figure C-2 shows the connections for a minimum configuration cable. The GPIB-232CT connector is on the left.

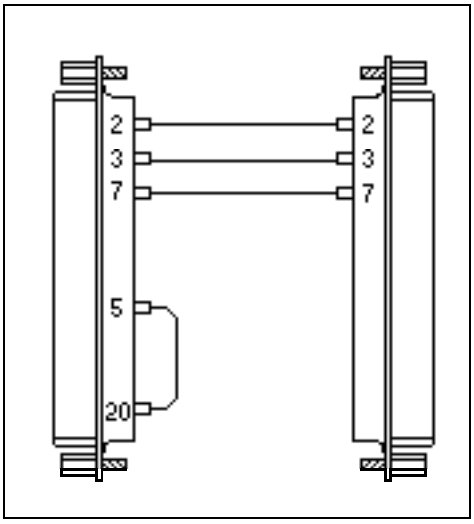


Figure C-2. Minimum DTE-to-DCE Cable Configuration

If your application requires a custom cable, review your serial device's RS-232 characteristics and build the cable to perform the desired functions.

Interfacing the GPIB-232CT to a DTE

For serial devices set up as DTEs, you must wire a DTE-to-DTE interface cable, commonly called a null modem cable. The cable must fool the GPIB-232CT into thinking it is communicating with a DCE. Figure C-3 shows a null modem cable. The GPIB-232CT connector is on the left.

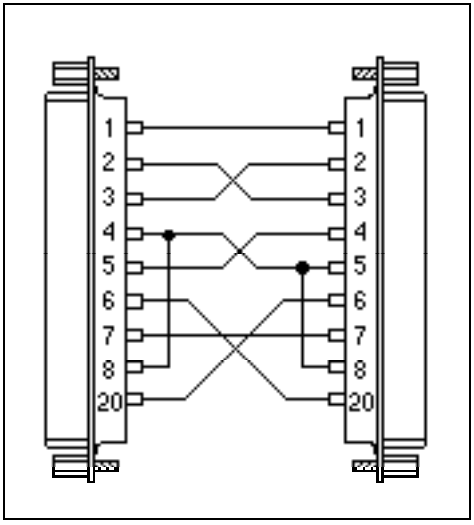


Figure C-3. Null Modem Cable Configuration

If your serial device does not use the same hardware handshaking protocol as the GPIB-232CT, you can use a minimum configuration null modem cable and use XON/XOFF handshaking (if necessary). Or, you can use a custom cable that will satisfy the GPIB-232CT hardware handshaking protocol. Figure C-4 shows the connections for a minimum configuration null modem cable. In Figure C-4, the GPIB-232CT connector is on the left.

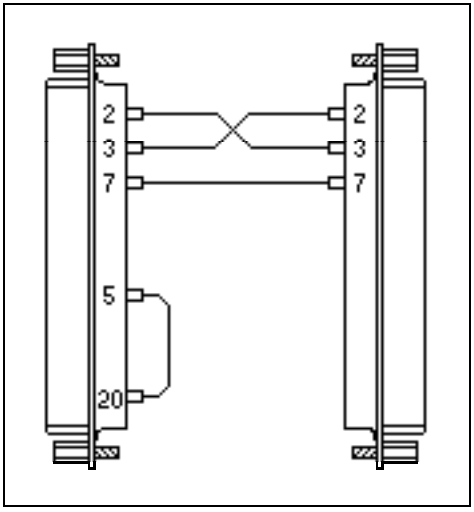


Figure C-4. Minimum Null Modem Cable Configuration

If a custom cable is required for your application, review your serial device's RS-232 characteristics and build the cable to perform the desired functions.

Buffering and Handshaking

The GPIB-232CT is able to accept serial data faster than it is able to process it. Two protection mechanisms are used to ensure that the GPIB-232CT does not lose incoming data: data buffering, and handshaking.

The GPIB-232CT has an internal RAM buffer that stores incoming serial data and serial port error information until it can process the data and, if necessary, output the data to the GPIB port. The size of this RAM buffer, in part, determines how much serial data the GPIB-232CT can accept until

its buffer overflows. The GPIB-232CT comes equipped with either a 64K RAM buffer or a 256K RAM buffer, 32K of which is used for the serial input buffer.

When its RAM buffer is nearly full, the GPIB-232CT is capable of handshaking with the serial host to stop data transmission. When the buffer is almost empty, the GPIB-232CT can again handshake with the serial host to start data transmission. The GPIB-232CT is capable of using both the XON/XOFF and the hardware handshaking protocols. The hardware handshake is always active during RS-232 transfers. You can turn the XON/XOFF handshake on or off by calling `xon`.

Hardware Handshake

The hardware handshake function is always active during RS-232 transfers and uses the Request to Send (RTS) and Clear to Send (CTS) signal lines. When the GPIB-232CT is ready to accept serial data, it asserts the RTS line. This signal remains asserted until the GPIB-232CT's data buffer is almost full. At this point, the GPIB-232CT unasserts the RTS line, signaling to the serial host that the GPIB 232CT is no longer ready to accept data. The serial host should monitor the RTS line and suspend data transmission whenever the RTS line becomes unasserted. The GPIB-232CT asserts RTS when it is again ready to receive serial data.

The GPIB-232CT is also able to suspend transmission when the serial device is no longer ready to accept data. The GPIB-232CT is configured to immediately stop transmission of serial data when CTS becomes unasserted. The GPIB-232CT resumes transmission as soon as CTS is reasserted.

Because most serial devices use the same form of hardware handshaking as the GPIB-232CT, you can achieve bi-directional flow control by using a serial cable that connects the GPIB-232CT's RTS signal to the serial device's CTS signal. In addition, the serial device's RTS signal should be connected to the GPIB-232CT's CTS signal. This setup allows each device to monitor the other device's RTS signal and to suspend transmission when necessary to prevent data loss.

Note: CTS must be asserted for the GPIB-232CT to transmit data. If you do not use hardware handshaking, you can usually make CTS asserted by tying together the CTS and DTR serial signals of the GPIB-232CT.

XON/XOFF

If your RS-232 serial device does not recognize the hardware handshake scheme, you can enable the XON/XOFF handshaking protocol. This handshaking protocol performs the same function as the hardware handshake, but does it by sending special control codes over the data lines instead of by changing logic levels on dedicated control lines.

When you enable the XON/XOFF protocol, the GPIB-232CT sends the XOFF character (decimal 19 or <CTRL>s) when its internal buffer becomes full. Once the GPIB-232CT is able to start receiving characters again, it sends the XON character (decimal 17 or <CTRL>q). When you enable the GPIB-232CT to recognize XON/XOFF, if the GPIB-232CT is transmitting data and receives XOFF, it suspends transmission of any further data until it receives XON.

Appendix D
Operation of the GPIB

The GPIB is a link, bus, or interface system through which interconnected electronic devices communicate. Hewlett-Packard invented the GPIB, which they call the HP-IB, to connect and control programmable instruments manufactured by them. Because of its high system data rate ceilings of from 250 kbytes/sec to 1 Mbytes/sec, the GPIB quickly became popular in other applications such as intercomputer communication and peripheral control. It was later accepted as the industry standard IEEE-488. The versatility of the system prompted the name General Purpose Interface Bus.

Types of Messages

The GPIB carries device-dependent messages and interface messages.

- Device-dependent messages, often called *data* or *data messages*, contain device-specific information such as programming instructions, measurement results, machine status, and data files.

- Interface messages manage the bus itself. They are usually called *commands* or *command messages*. Interface messages perform such tasks as initializing the bus, addressing and unaddressing devices, and setting device modes for remote or local programming.

The term *command* as used here should not be confused with some device instructions that can also be called commands. Such device-specific instructions are actually data messages.

Talkers, Listeners, and Controllers

A Talker sends data messages to one or more Listeners. The Controller manages the flow of information on the GPIB by sending commands to all devices.

Devices can be Listeners, Talkers, and/or Controllers. A digital voltmeter, for example, is a Talker when sending measurements and is a Listener when receiving programming messages.

The GPIB is a bus like an ordinary computer bus, except that the computer has its circuit cards interconnected via a backplane bus, whereas the GPIB has stand-alone devices interconnected via a cable bus.

The role of the GPIB Controller can also be compared to the role of the CPU of a computer, but a better analogy is to the switching center of a city telephone system.

The switching center (Controller) monitors the communications network (GPIB). When the center (Controller) notices that a party (device) wants to make a call (send a data message), it connects the caller (Talker) to the receiver (Listener).

The Controller addresses a Talker and a Listener before the Talker can send its message to the Listener. After the message is transmitted, the Controller can unaddress both devices.

Some bus configurations do not require a Controller. For example, one device can always be a Talker (called a Talk-only device) and there can be one or more Listen-only devices.

A Controller is necessary when the active or addressed Talker or Listener must be changed. The Controller function is usually handled by a computer.

In S mode, the GPIB-232CT peforms the following three roles:

- Controller - to manage the GPIB
- Talker - to send data to an attached GPIB device
- Listener - to receive data from an attached GPIB device

In G mode, the GPIB-232CT is a GPIB device. It performs only the following two roles:

- Talker - to send data to the GPIB host
- Listener - to receive data from the GPIB host

The Controller-In-Charge and System Controller

Although there can be multiple Controllers on the GPIB, only one Controller at a time is active or Controller-In-Charge (CIC). Active control can be passed from the current CIC to an idle Controller. Only one device on the bus, the System Controller, can make itself the CIC. The GPIB interface board is usually the System Controller in S mode and is never the System Controller in G mode.

GPIB Signals and Lines

The interface system consists of 16 signal lines and 8 ground return or shield drain lines.

The 16 signal lines are divided into the following three groups.

- Eight data lines
- Three handshake lines
- Five interface management lines

Figure D-1 shows the arrangement of these signals on the GPIB cable connector (a * suffix indicates that the signal is active low).

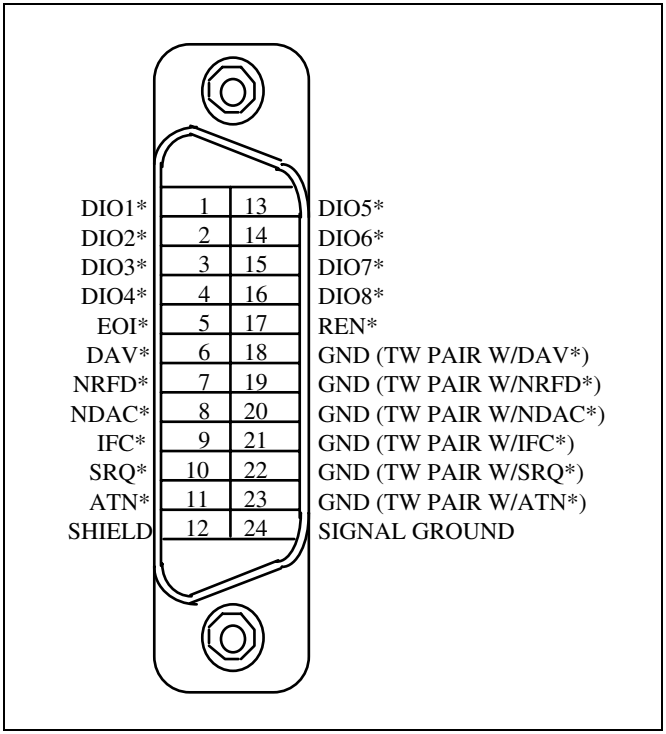


Figure D-1. GPIB Cable Connector

Data Lines

The eight data lines, DI01* through DI08*, carry both data and command messages. All commands and most data use the 7-bit ASCII or ISO code set, in which case the eighth bit, DI08*, is unused or used for parity.

Appendix A lists the GPIB command messages.

Handshake Lines

Three lines asynchronously control the transfer of message bytes among devices. The process is called a three-wire interlocked handshake, and it guarantees that message bytes on the data lines are sent and received without transmission error.

NRFD* (not ready for data)

NRFD* indicates when a device is ready or not ready to receive a message byte. The line is driven by all devices when receiving commands and by Listeners when receiving data messages.

NDAC* (not data accepted)

NDAC* indicates when a device has or has not accepted a message byte. The line is driven by all devices when receiving commands and by Listeners when receiving data messages.

DAV* (data valid)

DAV* tells when the signals on the data lines are stable (valid) and can be accepted safely by devices. The Controller drives DAV* when sending commands and the Talker drives it when sending data messages.

The way in which NRFD* and NDAC* are used by the receiving device is called the Acceptor Handshake. Likewise, the sending device uses DAV in the Source Handshake.

Interface Management Lines

Five lines are used to manage the flow of information across the interface.

ATN* (attention)

The Controller drives ATN* true when it uses the data lines to send commands and false when it allows a Talker to send data messages.

IFC* (interface clear)

The System Controller drives the IFC* line to initialize the bus and become Controller-In-Charge.

REN* (remote enable)

The System Controller drives the REN* line, which is used to place devices in remote or local program mode.

SRQ* (service request)

Any device can drive the SRQ* line to asynchronously request service from the Active Controller with the SRQ* line.

EOI* (end or identify)

The EOI* line has two purposes. The Talker uses the EOI* line to mark the end of a message string. The Controller uses the EOI* line to tell devices to identify their response in a parallel poll.

Physical and Electrical Characteristics

Devices are usually connected with a cable assembly consisting of a shielded 24 conductor cable with both a plug and receptacle connector at each end. This design allows devices to be linked in either a linear or a star configuration, or a combination of the two. See Figures D-2 and D-3.

The standard connector is the Amphenol or Cinch Series 57 *Microribbon* or *Amp Champ* type. An adapter cable using a non-standard cable and/or connector is used for special interconnection applications.

The GPIB uses negative logic with standard TTL logic levels. When DAV* is true, for example, it is a TTL low level (• 0.8V), and when DAV* is false, it is a TTL high level (• 2.0V).

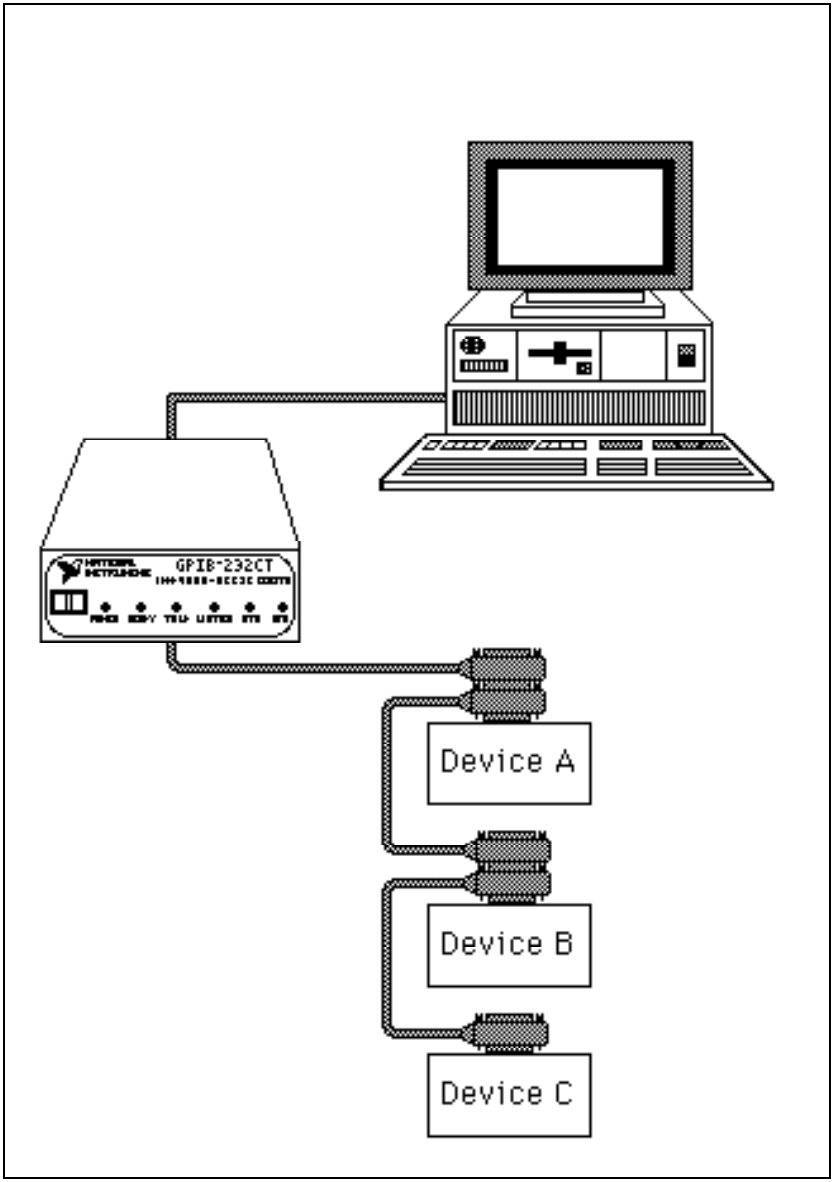


Figure D-2. Linear Configuration of the GPIB Devices

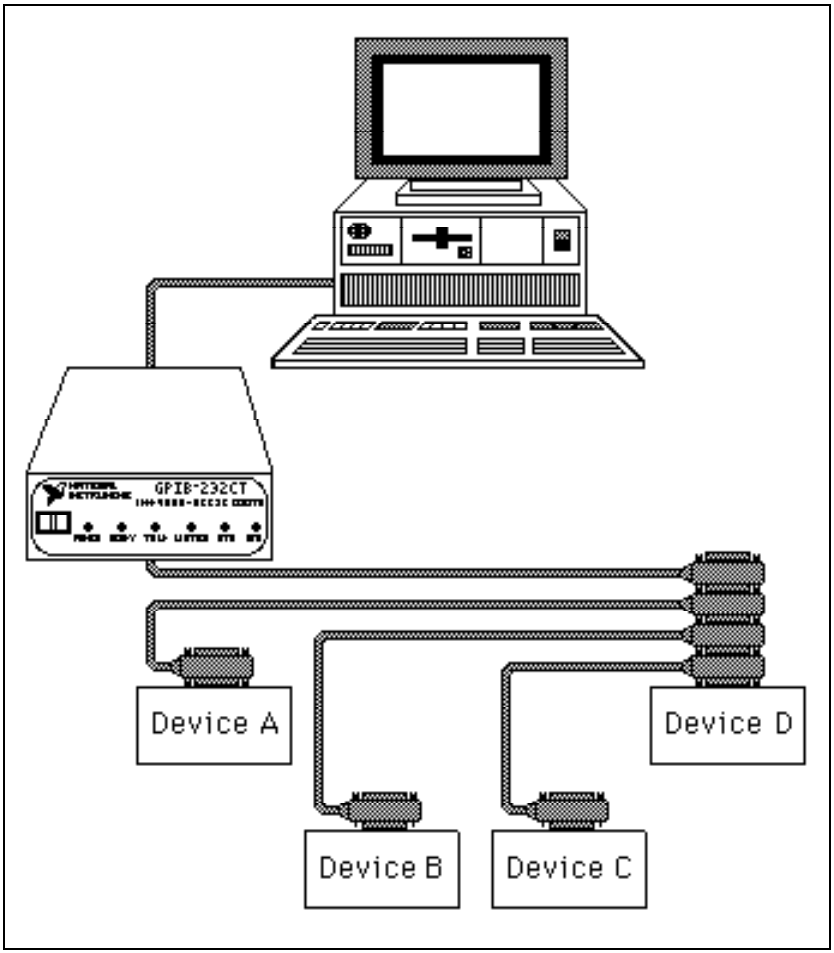


Figure D-3. Star Configuration of GPIB Devices

Configuration Requirements

To achieve the high data transfer rate that the GPIB was designed for, the physical distance between devices and the number of devices on the bus are limited.

The following restrictions are typical.

- A maximum separation of 4 m between any two devices and an average separation of 2 m over the entire bus.
- A maximum total cable length of 20 m.
- No more than 15 devices connected to each bus, with at least two-thirds powered on.

Bus extenders are available from National Instruments and other manufacturers for use when these limits must be exceeded.

Related Document

For more information on topics covered in this appendix, consult *IEEE Standard Digital Interface for Programmable Instrumentation*, ANSI/IEEE Std. 488-1987.

Appendix E
Common Questions

This appendix consists of two sections. The first section answers questions about S mode. The second section answers questions about G mode.

S Mode

Question

Why does the manual suggest that I use `INPUT$` sometimes, and `LINE INPUT#` at other times? Microsoft suggests using `INPUT$` to read from the serial port.

Answer

Use `LINE INPUT#` to read status information from the GPIB-232CT. GPIB-232CT software formats its status information so that your BASIC program can easily read and interpret each of its pieces. Each logical piece of status information is followed by a carriage return (<CR>) and linefeed (<LF>). `LINE INPUT#` allows you to easily read each piece of status information and assign it to a variable.

Use `INPUT$` to read a data string from your GPIB device. `INPUT$` requires that you know the exact number of characters you wish to read from the serial port. When reading status information from the GPIB-232CT, this is not always possible since the responses can vary in length from one call to the next. However, when reading a data string from your GPIB device, you requested a certain number of bytes and you should use `INPUT$` to read the number of bytes you requested in your rd function. The GPIB-232CT appends to the end of the data string a string containing the number of bytes that were actually read from the GPIB. When you have read in the data bytes using `INPUT$`, use `LINE INPUT#` to read the string containing the byte count. Refer to the example following the `rd` function description in Chapter 5.

Question

When I use `LINE INPUT#`, my strings are usually preceded by a linefeed (<LF>)and followed by a carriage return (<CR>). Why don't my strings contain both a carriage return and linefeed at the end?

Answer

`LINE INPUT#` stops reading when a carriage return is seen and does not skip over the linefeed in the sequence. The linefeed is not read until the following `LINE INPUT#`. In most cases, you will be using the `val` function to convert the string to a value and a leading linefeed is ignored.

Question

I sent the programming message `"rsp 10"` to the GPIB-232CT to serial poll device 10. Then, I used `LINE INPUT#` to read the response byte and got nothing but a carriage return and linefeed (<CR><LF>) as a response. Am I doing something wrong?

Answer

No. To conduct a serial poll, the GPIB-232CT must be Controller-In-Charge or it must be able to become Controller-In-Charge. If the GPIB-232CT cannot become Controller-In-Charge, no serial poll is conducted, and therefore you will not get a response string. To see if this is the problem, ask for status (see `stat`) and check to see if the ECIC error occurred. If it did, you have passed control or System Controller authority to some other GPIB device, and will not be able to perform a serial poll until the GPIB-232CT gets Controller authority back.

G Mode

Question

After I write the programming message `"stat cs"` to the GPIB-232CT, status is not returned. All I get is a carriage return followed by a linefeed (<CR><LF>). Why don't I get status information?

Answer

Every programming message must be followed by a carriage return and/or linefeed. In BASIC, you can build your string as follows:

```
"stat c s"+chr$(13)
```

Question

After I write the programming message `"stat c s"+chr$(13)` to the GPIB-232CT in G mode, my system times out when I try to read status. Why?

Answer

Have you addressed the GPIB-232CT to talk? To read responses to programming messages you send to the GPIB-232CT, you must address the GPIB-232CT to talk, using the address you selected on the configuration switch and adding decimal 64 to it.

Question

Whenever I send a programming message to the GPIB-232CT, it seems as if the GPIB-232CT never receives it. The GPIB-232CT is powered on and its **READY** LED is on.

Answer

On U22, switch 1 must be on for G mode and switches 2 and 3 must be off. If you must change these switch settings, be sure to power the GPIB-232CT off and then power it on again. Check all of your cable connections.

Appendix F
Parallel Polling

A GPIB Controller can use parallel polling to obtain information from several devices on the GPIB in one operation. The Controller polls configured devices and reads back a single response byte that contains one bit of information from each device. From this information, the Controller can determine which devices need servicing.

Operation

A device is configured by sending it its listen address and a parallel poll enable (PPE) message. There are 16 possible PPE messages: hex 60 through hex 6F. The bits in the PPE message have the following meaning:

			U	S	DIO lines 1-8		
0	1	1	X	X	X	X	X

U	When 0 (hex 6X), parallel poll is enabled. When 1 (hex 7X), parallel poll is disabled.
S	When the device's IST (individual status) bit matches the S bit, the device will set the appropriate data line. Hex 60 through hex 67, set S to 0; hex 68 through hex 6F, set S to 1.
DIO	The value `n` in bits 0-2 corresponds to one of the DIO lines 1 through 8, where `n` corresponds to DIO line `n+1`. Thus, a value of 2 (binary 010) corresponds to DIO line 3.

The circumstances under which a device sets its IST bit are specific to that device. For example, a device might always set the IST bit to 1 when it is

busy and 0 when it is free, or vice versa. Consult your device documentation for this information. With this information, the Controller can configure devices according to the information desired.

Only the Active Controller can perform a parallel poll.

There are two steps to conducting a parallel poll: the following configuration step, and the actual polling. The following paragraphs describe these two steps.

Configuration

In the S Mode, the `ppc` function configures devices for parallel polls. For example, if you want to configure a device at address 5 to respond on DIO line 3 when the IST bit is 1, the programming message would be `ppc 5,3,1`. The GPIB-232CT takes the arguments `3,1` and constructs the following parallel poll enable byte:

			U	S	DIO lines 1-8		
0	1	1	0	1	0	1	0

The value of this byte is hex 6A where:

U = 0	Enable
S = 1	Thus, when IST = 1 the device will assert DIO line 3 (which corresponds to 010 in bits 0-2).

The `ppc` function sends the device's listen address, Parallel Poll Configure, hex 6A, then unlisten. The Active Controller can configure itself to respond to a parallel poll using `ppc`, also. This might be used in the case where the GPIB-232CT is not the System Controller and the System Controller does not have the capability to do the configuration. Since the GPIB-232CT cannot be the Controller in the G Mode, the GPIB Controller in your system must configure the GPIB-232CT in order to parallel poll it.

The Parallel Poll

In the S Mode, after configuring the device, the GPIB-232CT now conducts a parallel poll by calling `rpp`. In the previous example, where the device was sent a configuration byte of hex 6A, if the device's IST bit matches the S bit of hex 6A, `rpp` will return the value 04. Here, the third least significant bit is set, corresponding to DIO line 3. (If any other devices responded positively on other lines, those corresponding bits would be set as well.)

Note: The Controller can configure more than one device to respond on the same data line, in which case the bits in the response byte are set by the ORing of all the responses on that line.

In the G Mode, the GPIB-232CT sets its IST bit whenever it asserts SRQ, and clears it whenever it unasserts SRQ. Refer to Chapter 7 for the conditions under which the GPIB-232CT asserts SRQ. If the Active Controller has sent it the Parallel Poll Configure byte hex 6D (binary 0110 1101) and parallel polls it while its IST bit is set, it responds by asserting data line 6. If the Active Controller sent it the parallel poll configure byte hex 65 (binary 0110 0101) and parallel polls it while its IST bit is set, it responds by not asserting data line 6.

Disabling Parallel Poll Response

The Active Controller can disable a specific device from responding to a parallel poll by calling `ppu` with the device address as a parameter. `ppu` sends the device the parallel poll unconfigure byte hex 70 (binary 0111 0000), which sets U to 1 to disable the device from responding to a parallel poll.

To unconfigure all devices, the Controller can call `ppu` with no arguments, which sends PPU (parallel poll unconfigure, hex 15).

S Mode Example

A system has three line printers, two tape drives, one card reader, and one PC on a system. The PC uses a GPIB-232CT to communicate on the GPIB. All other devices are GPIB devices. The PC is designated to be Active Controller, and all other devices recognize this. Furthermore, all devices will set their IST bit to 1 when they are busy and 0 when they are free.

The Active Controller configures the card reader (at address 6) to respond positively on DIO line 4 (which sets bit 3 of the response byte) when free by sending the configuration byte 01100011. (The S bit is set to 0, the value of bits 0 through 2 is 3.)

```
PPC 6,4,0
```

When a parallel poll is conducted, one of two things will happen. If the card reader is free, bit 3 of the response byte will be 1; if it is busy, bit 3 will be 0. When the device is free, its IST bit is 0 and because this equals the value of the S bit, the device asserts DIO line 4.

The Active Controller configures all of the line printers to respond positively on DIO line 1 when busy. In this case, the `ppr,s` argument for each of them is `0,1`. Thus, the configuration byte for each of them is 01101000 (hex 68). When a parallel poll is conducted, the Controller can immediately find out if all line printers are free, because the response in this situation will be 0. If any line printer is busy, bit 0 of the parallel poll response will be 1, corresponding to DIO line 1 being asserted. However, what if the Active Controller wants to know if one line printer is free? If the Controller reconfigures the line printers to respond positively when free (dio, s = 0, 0; configuration byte = 01100000), then if any device is free, it will drive the DIO line to 1. Thus, the Controller can use S-bit/IST bit correspondence for different types of information.

Appendix G
Setting Switches

This appendix explains how to set the configuration switch of the GPIB-232CT when operating in S mode.

As shown in Chapter 3, switch 1 is placed down (or off) to operate in S mode. Set the remaining switches to match the characteristics of the terminal or computer you will attach to the other end of the serial cable.

Often, you can change the serial port characteristics of the terminal or computer by setting switches, running a utility program, or from within BASIC. Determine the default characteristics of your computer or terminal's serial port. If you want to change the configuration on that side, do so before attempting to communicate with the GPIB-232CT. Then, set the configuration switch on the GPIB-232CT to match your serial port characteristics.

For example, for Microsoft BASIC on a Macintosh, the default characteristics are as follows:

- baud rate: 300
- parity: even
- data bits: 7
- stop bits: 1

If these defaults meet your needs, set the switches on the GPIB-232CT as shown in Figure G-1.

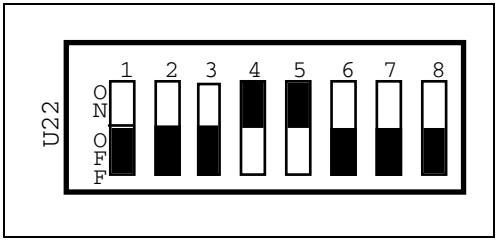


Figure G-1. S mode, 7 data bits, 1 stop bit, even parity, 300 baud

In many cases, you will want to change the default characteristics of the serial port on the Macintosh. For instance, you may want to run at a higher baud rate and you may want to send 8-bit data bytes for binary data that will be sent to the GPIB device. To change the default characteristics to 19.2K baud and 8-bit data from within Microsoft BASIC, place the following BASIC statement at the beginning of your program:

```
OPEN "COM1:19200,,8," AS #1
```

then set the switches on the GPIB-232CT as shown in Figure G-2.

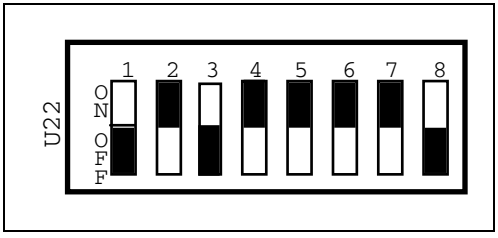


Figure G-2. S mode, 8 data bits, 1 stop bit, even parity, 19200 baud

As another example, if your computer is an Apple IIc, the default characteristics are as follows:

- baud rate: 9600
- parity: none
- data bits: 8
- stop bits: 1

If these default characteristics meet your needs, set the switches on the GPIB-232CT as shown in Figure G-3.

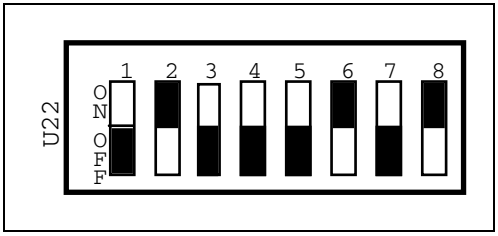


Figure G-3. S mode, 8 data bits, 1 stop bit, no parity, 9600 baud

These default characteristics might be adequate for most applications. However, if you want to change any of the Apple IIc's serial port characteristics, use the utility program described in Chapter 4 of the Apple IIc Owner's Manual, *Configure the Serial Port*. After changing the Apple IIc's serial port characteristics, set the switches on the GPIB-232CT to match precisely. See Chapter 3 of this manual for details on setting each switch.

As one final example, if your computer is an IBM PC or compatible, the serial port default characteristics are as follows:

- baud rate: 300
- parity: even
- data bits: 7
- stop bits: 1

If these defaults meet the needs of your application, set the GPIB-232CT switches as shown in Figure G-4.

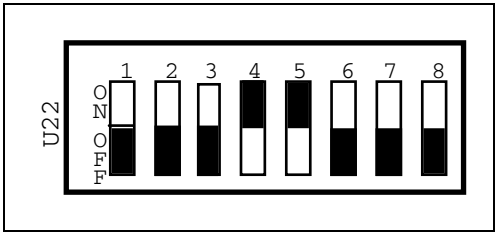


Figure G-4. S mode, 7 data bits, 1 stop bit, even parity, 300 baud

In many cases, you will want to change the default characteristics of the serial port on the IBM PC. You may want to run at a higher baud rate (up to 19200 baud) and you may want to send 8-bit data bytes for binary data that will be sent to the GPIB device. To change the IBM PC's serial port default characteristics to 9600 baud and 8 data bits from within BASICA, place the following BASIC statement at the beginning of your application program:

```
OPEN "COM1:9600,,8," AS #1
```

then set the switches on the GPIB-232CT as shown in Figure G-5.

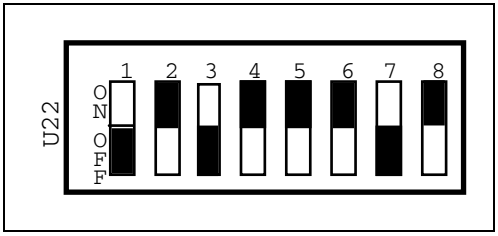


Figure G-5. S mode, 8 data bits, 1 stop bit, even parity, 9600 baud

If your computer (or terminal) is other than those described above, read your user's manual to learn the default settings of the serial port.

On a terminal, there will be switches or a setup program. If you are using Applesoft BASIC or Integer BASIC for your application, you should configure for your serial port before you enter BASIC. If you are using a version of Microsoft BASIC, you can change the serial port characteristics from within BASIC. Refer to the OPEN command in your BASIC manual.

Remember, whatever serial port characteristics you decide to use, you must set up both your serial device and your GPIB-232CT to identical characteristics.

Appendix H
Sample Programs

This appendix contains program examples you can use as guidelines as you start writing programs for the GPIB-232CT. The first part of the appendix applies to S mode. The remainder of the appendix applies to G mode.

S Mode Sample Programs

The following steps are general programming steps. The following pages contain detailed explanations of these steps and show some sample programs.

1. Send the `stat` function to have status information returned to you after your programming message.

2. Send serial port initialization functions if you need to change default serial port settings.

3. Send GPIB initialization functions if you need to change default GPIB settings.

4. Communicate with the device using the `rd` and `wrt` functions, and check status if you requested it.

After you initialize the GPIB-232CT, the `rd` and `wrt` functions may be the only functions you will need.

Using an HP 7475A Plotter with a Terminal

This program is a sequence of programming messages and data strings entered from a terminal to draw a circle using an HP 7475A Color Plotter.

Getting Ready to Program

Before you start programming, determine the serial port settings you need by looking at the settings of your terminal; then, set the configuration switches on the GPIB-232CT so that they match the terminal characteristics. Figure H-1 shows how to configure the GPIB-232CT if your terminal has the following characteristics:

- baud rate: 19200
- parity: none
- data bits: 8
- stop bits: 1

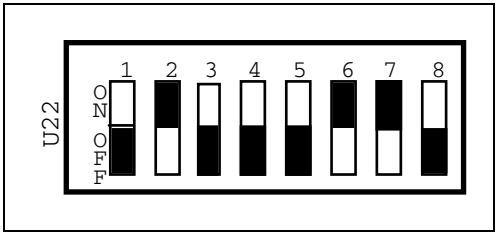


Figure H-1. S mode, 8 data bits, 1 stop bit, no parity, 19200 baud

Next, connect the serial cable to the serial port of the terminal and to the GPIB-232CT. Connect the GPIB cable to the GPIB port on your device and to the GPIB-232CT. Power-on the GPIB-232CT.

A program written from a terminal does not exactly follow the steps used for most programs. To be able to see on the screen the information you are entering, the echo function must be enabled. In this case, a serial port initialization message is sent before `stat`. To enable echo, type the following:

```
echo 1 <CR>
```

As you type, the characters will not be echoed yet. You are enabling echo for future programming messages. Remember to type a carriage return.

Programming Steps

Step 1. `stat` Function

When using the `stat` function, use `stat c s` so the GPIB-232CT will report status after each programming message. In `stat c s`, the `c` stands for continuous, and the `s` stands for symbolic. Because you are using a terminal, it is easier to interpret the status report if it is returned in symbolic form. Type the following:

```
stat c s <CR>
```

The status displays on the terminal's screen immediately. The status should look something like this:

```
CMPL
NGER
NSER
0
```

This status report shows the status after the previous programming message. It also shows that `echo 1` was a previous programming message because characters you enter are now being echoed. This status reports `CMPL` and no errors.

Step 2. Serial Port Functions

Determine what serial port functions are needed to change default settings. `echo` was enabled as a preliminary step. No other changes are necessary for this GPIB plotter.

Step 3. GPIB Initialization Functions

Determine what GPIB initialization functions are needed to change default settings. If you are not sure what GPIB settings you might want to change, try using the defaults. No changes were necessary for this GPIB plotter.

Step 4. Communicate with `rd` and `wrt` Functions

Use `rd` and `wrt` to communicate with the device. In this example we want to send plotter commands to the GPIB plotter, so we only need the `wrt` programming message. The plotter's address is 5. In the following example, the programming messages and data strings that you enter are shown in regular type. Responses sent to you by the GPIB-232CT are shown in **boldface** type.

By looking at the status information returned after each programming message, you can see if any GPIB or serial port errors occurred, and you can see the number of bytes actually transferred out on the GPIB.

The command (`tmo 30`) on the second line of the following example lengthens the timeout from 10 seconds (default) to 30 seconds to allow you more time to type the data string for the `wrt` commands.

```
echo 1<CR>
tmo 30<CR>
stat c s<CR>
CMPL
NGER
NSER
0
wrt 5<CR>
IN;SP1;IP2650,1325,7650,6325;<CR>
CMPL,CIC,TACS
NGER
NSER
29
wrt<CR>
SC-100,100,-100,100;PA0,0;CI40;<CR>
CMPL,CIC,TACS
NGER
NSER
31
```

Using an HP 7475A Plotter with an IBM PC

This example shows how to write a program on an IBM PC using Microsoft BASIC to draw a circle using an HP 7475A Plotter.

Getting Ready to Program

Before you start programming, determine the serial port settings you will use. Figure H-2 shows how to configure the GPIB-232CT for this example, using the following settings:

- baud rate: 9600
- parity: none
- data bits: 7
- stop bits: 1

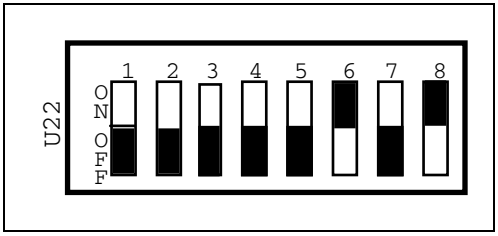


Figure H-2. S mode, 7 data bits, 1 stop bit, no parity, 9600 baud

Next, connect the serial cable to the serial port of your computer and to the GPIB-232CT. Connect the GPIB cable to your device and the GPIB-232CT. Power-on the GPIB-232CT.

Programming Steps

Step 1. `stat` Function

In BASIC, before reading or writing to the serial port, a device must be *opened*. Place the following BASIC statement at the beginning of your program to open and configure the serial port (COM1) and name it device `#1`.

```
OPEN "COM1:9600,n,7,1" as #1
```

You will now use `PRINT #1` to redirect strings to the serial port.

Now, send the `stat` function if you want status information returned after every programming message. To do this, include the following code in your program:

```
PRINT #1, "stat c n"
```

After you send this programming message, you can expect four lines of data at the serial port (each line is terminated by <CR><LF>). For now, call a subroutine to check the status. You can write this later. Add the following line to call the subroutine:

```
GOSUB status
```

Step 2. Serial Port Functions

Send serial port initialization programming messages, if necessary. For this example, this step is not necessary.

Step 3. GPIB Initialization Functions

Send GPIB initialization programming messages, if necessary. For this example, this step is not necessary.

Step 4. Communicate with `rd` and `wrt` Functions

Communicate with the device using `wrt` programming messages, and reading back status after each. Here is the heart of the program. After each `wrt` string, call the subroutine `status`, which will check for errors. The plotter's GPIB address is 5.

```
OPEN "com1:9600,n,7,1" AS #1
PRINT #1,"stat c n"
     GOSUB status
PRINT #1,"wrt 5"
PRINT #1,"in;sp1,pa1000,3000;ci500;"
     GOSUB status
END

status:
STAT%=VAL(LINE INPUT #1, status$)
LINE INPUT #1,gpiberr$
LINE INPUT #1,sperr$
LINE INPUT #1,cnt$
PRINT status$ gpiberr$ sperr$ cnt$
if stat% < 0 GOSUB error

error:
REM Place your code to handle errors here.
STOP
```

Programming a Tektronix 2445 Oscilloscope from an Apple IIc

This example shows how to use an Apple IIc to control a Tektronix 2445 oscilloscope. The Apple IIc's serial port is configured with a utility program. In Applesoft BASIC, there is no need to use the OPEN statement to open the serial port for communications. You direct output to the serial port with the statement `PR#1`, and direct input to the serial port with the statement `IN#1`. A <CTRL>D (`CHR$(4)`) must be sent before each of these commands. Refer to your Applesoft manual for more information.

Getting Ready to Program

Before you start programming, determine what serial port settings you require. The default settings for the GPIB-232CT as given below are fine for this application.

- baud rate: 9600
- parity: none
- data bits: 8
- stop bits: 1

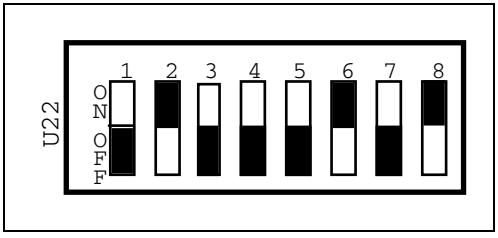


Figure H-3. S mode, 8 data bits, 1 stop bit, no parity, 9600 baud

Connect the serial cable to the serial port of your computer and to the GPIB-232CT. Power-on the GPIB-232CT.

Programming Steps

Step 1. `stat` Function

In Applesoft BASIC, you must redirect input and output to the serial port before attempting to communicate with the GPIB-232CT. Place the following statements at the beginning of your program:

```
D$ = CHR$(4)
PRINT D$,"PR#1"
PRINT D$,"IN#1"
```

Step 2. Serial Port Functions

Send serial port initialization programming messages, if necessary. For this example, this step is not necessary.

Step 3. GPIB Initialization Functions

Send GPIB initialization programming messages, if necessary. For this example, this step is not necessary.

Step 4. Communicate with `rd` and `wrt` Functions

Communicate with the device using `rd` and `wrt` programming messages. Here is the heart of the program:

```
 10  D$ = CHR$ (4)
 20  PRINT D$; "PR#2": REM INPUT FROM SLOT #2 (MODEM)
 30  PRINT D$; "IN#2": REM OUTPUT TO SLOT #2 (MODEM)
 40  PRINT
 60  PRINT "WRT 6":    REM WRITE STRING TO DEVICE 6
 70  PRINT "CTT?;"
 90  PRINT "RD 20 6":  REM READ UP TO 20 BYTES FROM
 95                    REM DEVICE 6
100  INPUT "";RESP$:   REM INPUT FREQUENCY AND BYTE
105                    REM COUNT
110  PRINT D$;"PR#0":  PRINT RESP$:  PRINT D$;"PR#2"
140  PRINT D$;"PR#0"
```

```
150  PRINT D$;"IN#0"
500  END
```

G Mode Sample Programs

The following steps are general programming steps. The following pages contain detailed explanations of these steps and show some sample programs.

1. Send the `stat` function to have status information returned to you after your programming message.

2. Send GPIB initialization functions if you need to change default GPIB settings.

3. Send serial port initialization functions if you need to change default serial port settings.

4. Communicate with the serial device and obtain status from the GPIB-232CT, if desired.

After you initialize the GPIB-232CT, you may only need to perform reads and writes from the serial device.

To send data to the device you must address the device. To send programming messages to the GPIB-232CT, you must address the GPIB-232CT. The GPIB software you use on the Controller side can provide high-level calls that perform this addressing automatically.

Using a Serial HP 7475A Plotter with a GPIB Controller

This is a program to draw a circle using an HP 7475A Color Plotter with a serial interface. The Controller is an IBM PC with the National Instruments GPIB-PC card and software.

Getting Ready to Program

First, set up the GPIB-232CT configuration switches with a primary address of 2 as shown in Figure H-4.

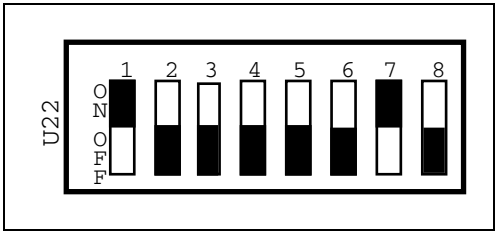


Figure H-4. G mode, primary GPIB address 2

Next, connect the GPIB cable to the GPIB port on your computer and to the GPIB-232CT. Connect the serial cable to the serial port on your device and to the GPIB-232CT. Power-on the GPIB-232CT.

The serial port characteristics of the GPIB-232CT and the serial device must be exactly the same. In this case, the plotter's serial port settings are as follows:

- baud rate: 9600
- parity: none
- data bits: 7
- stop bits: 1

The GPIB-232CT's default serial port characteristics, however, are as follows:

- baud rate: 9600
- parity: none
- data bits: 8
- stop bits: 1

The number of data bits do not match, so the default setting of the GPIB-232CT must be changed before you read or write from the plotter.

To communicate with the GPIB-232CT, send it data strings using the `IBWRT` function. The following steps explain each of the strings sent to the GPIB-232CT.

Programming Steps

Step 1. `stat` Function

Send `stat c n` if you want status information returned after each programming message. Next, use the `ibrd` function to read back status after this string is sent.

Step 2. GPIB Initialization Functions

If there are GPIB functions on the GPIB-232CT you wish to change from their default settings, send strings to the GPIB-232CT at this time. No changes are necessary to communicate with this plotter.

Step 3. Serial Port Initialization Functions

Change any serial port initialization functions, if necessary. This plotter sends and receives 7-bit data. Since the default of the GPIB-232CT is 8-bit data, send the string `spset 7` to change to 7-bit data transfers. Read status after the string has been sent.

Step 4. Communicate with Plotter

Before communicating with the plotter, send the initialization strings to the GPIB-232CT. Next, send strings to the plotter (the GPIB-232CT does not interpret these strings, but sends them straight to the plotter).

Remember to read status after you send a programming message to the GPIB-232CT.

```
 10  G232CT$="GPIB232CT"
 20  CALL IBFIND(G232CT$,GPIB232CT%)
 25  'Open GPIB-232CT
 30  SDNAME$="PLOTTER"
 40  CALL IBFIND(SDNAME$,PLOTTER%)
 45  'Open PLOTTER
 50  WRT$="stat c n" + CHR$(13)
 60  CALL IBWRT(GPIB232CT%,WRT$)
 65  'Enable continuous status reporting
 70  RD$=SPACE$(25)
 80  CALL IBRD(GPIB232CT%,RD$)    'Read up to
 90  '25 bytes of status information
100  IF ASC(RD$) = CHR$(45) THEN GOTO 400
110  'If first character in RD$ is a minus sign
120  'then go to error ftn.
130  WRT$="spset 7"
135  'Change the serial port configuration
140  CALL IBWRT(GPIB232CT%,WRT$)  'to 7 data
145  'bits (plotter configuration)
150  CALL IBRD(GPIB232CT%,RD$) 'Read status info.
160  IF ASC(RD$) = CHR$(45) THEN GOTO 400
170  'If first character in RD$
180  'is a minus sign then go to error function.
190  WRT$="IN;SP1;IP2650,1325,7650,6325;"
200  'Initialize plotter, select pin 1, set
210  'scaling points
220  WRT$=WRT$ + "SC-100,100,-100,100;"
230  'Scale the plotting area
240  WRT$=WRT$ + "PA50,-40;CI30,30;"
250  'Plot absolute, circle=radius
260  '30 degrees, chordangle 30 degrees
270  CALL IBWRT(PLOTTER%,WRT$)
330  STOP
400  PRINT "An error occurred:"
410  PRINT "status, GPIB-error, serial-error,
```

```
415  count:";RD$
420  END
```

IBM PC (with GPIB-PC) to Serial Printer

This example demonstrates how to control a serial printer on the GPIB. The printer is the Apple Imagewriter. The following program example prints the message "Hello World."

Getting Ready to Program

First, set the GPIB-232CT configuration switch with a primary address of 18, as shown in Figure H-5.

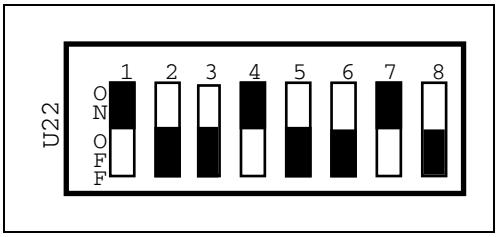


Figure H-5. G mode, primary GPIB address 18

Next, check the DIP switch settings on the Imagewriter. Switches 2-1 and 2-2 should be *closed* for 9600 baud rate.

The serial port characteristics of the GPIB-232CT must match those of the serial device. The baud rate of the printer is 9600. Since the configurations match, there is no need to initialize the software of the serial port.

Programming Steps

Step 1. `stat` Function

Send `stat c n` if you want status information returned after each programming message. Next, use the `ibrd` function to read back status after this string is sent, and after every other string sent to the GPIB-232CT until continuous status is disabled.

Step 2. GPIB Initialization Functions

If there are GPIB functions on the GPIB-232CT you wish to change from their default settings, send strings to the GPIB-232CT at this time. No changes are necessary to communicate with this plotter.

Step 3. Serial Port Initialization Functions

Next, change any serial port characteristics, if necessary. If you have set up the printer as previously described, no software initialization is necessary.

Step 4. Communicate with the Printer

Before communicating with the printer, send the initialization strings to the GPIB-232CT. Next, send strings to the printer (the GPIB-232CT does not interpret these strings, but sends them straight to the printer).

```
 10  G232CT$="GPIB232CT"
 20  CALL IBFIND(G232CT$,GPIB232CT%)
 25  'Open GPIB-232CT
 30  SDNAME$="PRINTER"
 40  CALL IBFIND(SDNAME$,PRINTER%)  'Open PRINTER
 50  WRT$="stat c n"+CHR$(13)
 60  CALL IBWRT(GPIB232CT%,WRT$)
 65  'Enable continuous status reporting
 70  RD$=SPACE$(25)
 80  CALL IBRD(GPIB232CT%,RD$)
 90  'Read up to 25 bytes of status information.
100  IF ASC(RD$) <> CHR$(45) THEN GOTO 310
110  'If first characterin RD$ is a minus sign,
```

```
120  'then go to error function.
210  WRT$="Hello, world"+CHR$(13)
220  CALL IBWRT(PRINTER%,WRT$)      'Send string to
230                                 'printer.
300  STOP
310  PRINT "An error occurred:"
320  PRINT "status, GPIB-error, serial-error,
     count:";RD%
330  END
```

User Comment Form

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

Title: **GPIB-232CT User Manual**

Edition Date: **July 1992**

Part Number: **320114-01**

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

(continues)

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

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039