VICTOR

IEEE-488 Interface

#31

IEEE-488 Reference Manual

Victor Institute
3240 S. Higuera Suite A
San Luis Obispo, CA 93401

VICTOR Publications
380 El Pueblo Road
Scotts Valley, CA 95066
(408) 438-6680

NOTICE

VICTOR makes no representations or warranties of any kind whatsoever with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. VICTOR shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this publication or its contents.

VICTOR reserves the right to revise this publication from time to time and to make changes in the content hereof without obligation to notify any person of such revision or changes.

First VICTOR printing March, 1983.

ISBN 0-88182-023-7 Printed in U.S.A.

CONTENTS

EXHIBITS

CHAPTERS

IMPORTANT SOFTWARE DISKETTE INFORMATION

For your own protection, do not use this product until you have made a backup copy of your software diskette(s). The backup procedure is described in the user's guide for your computer.

Please read the DISKID file on your new software diskette. DISKID contains important information including:

- The product name and revision number.
- The part number of the product.
- The date of the DISKID file.
- A list of the files on the diskette, with a description and revision number for each one.
- Configuration information (when applicable).
- Release notes giving special instructions for using the product.
- Information not contained in the current manual, including updates, additions, and deletions.

To read the DISKID file onscreen, follow these steps:

1. Load the operating system.
2. Remove your system diskette and insert your new software diskette.
3. Enter —

   TYPE DISKID

   and press Return.
4. The contents of the DISKID file is displayed on the screen. If the file is large (more than 24 lines), the screen display will scroll. Type ALT-S to freeze the screen display; type ALT-S again to continue scrolling.

1

OVERVIEW

The IEEE 488 communications interface is a powerful mechanism for device communication and control. This parallel interface is used to communicate with various devices (such as data acquisition and control products, measurement devices, plotters, and printers) that conform to the ANSI Standard MC1.1. The IEEE 488 communication protocol is also referred to as the General Purpose Interface Bus (GPIB) and the Hewlett-Packard Interface Bus (HP-IB).

The IEEE 488 package consists of high-level functions compatible with those offered by HP BASIC. These functions let application programs custom-write drivers to control IEEE 488 compatible devices. Each function lets control and data messages be exchanged with the desired device.

The first part of this manual discusses communication interfaces in general, and the IEEE 488 interface specifically. Next, the terminology and communication considerations involved in using the IEEE 488 communication protocol are explained. Finally, the manual presents specific information on available high-level device communication and control functions, along with installation information and a communication programming example.

2

BACKGROUND ON THE IEEE 488 INTERFACE

OVERVIEW 2.1

The IEEE 488 interface manages communications between your computer (or another controller) and peripheral devices such as printers, plotters and laboratory instruments. This generalized interface is a way to physically connect these devices; it includes a communication protocol for sending and retrieving data and control information between interconnected devices.

Devices are connected by using digital communication over a parallel bus. Specifications dictate the maximum number of devices on the bus, maximum length of the data bus, and the maximum data rate. The standard specification of this communication technique is IEEE 488 (1978) and ANSI MC1.1.

OVERVIEW OF GENERIC INTERFACES 2.2

Any interface system has two major elements: the message to be transferred and the method for transporting that message. For a generalized device communication interface, the types and contents of messages cannot be limited. Many devices must use the interface, and a variety of application programs must use these devices. The method of transferring information, however, must be standardized—drivers and hardware must be operational with any device that conforms to the standard interface definition.

Interface standards define:

- Physical connectors for the communication line or bus.
- Electrical levels and timing for the line or bus.
- Communication protocol for message passing.

Interface standards allow the development of standard controller hardware that can be connected to any device compatible with a particular interface. Specification of the communication protocol lets software drivers (such as the IEEE 488 package) give application programs a straightforward way to control and communicate with peripherals. The communication driver can consider such application-irrelevant (although critical) considerations as data validity and error recovery, data pacing, and coordination and control of all the devices on the bus.

2.3 RELATED PUBLICATIONS

More information on the IEEE 488 interface standard can be found in the following publications:

Digital Interface for Programmable Instrumentation, IEEE Standards, 345 E. 47th Street, New York, NY 10017.

ANSI Standards, ANSI, 1430 Broadway, New York, NY 10018.

TECHNICAL BACKGROUND

BASIC SPECIFICATIONS AND LIMITATIONS 3.1

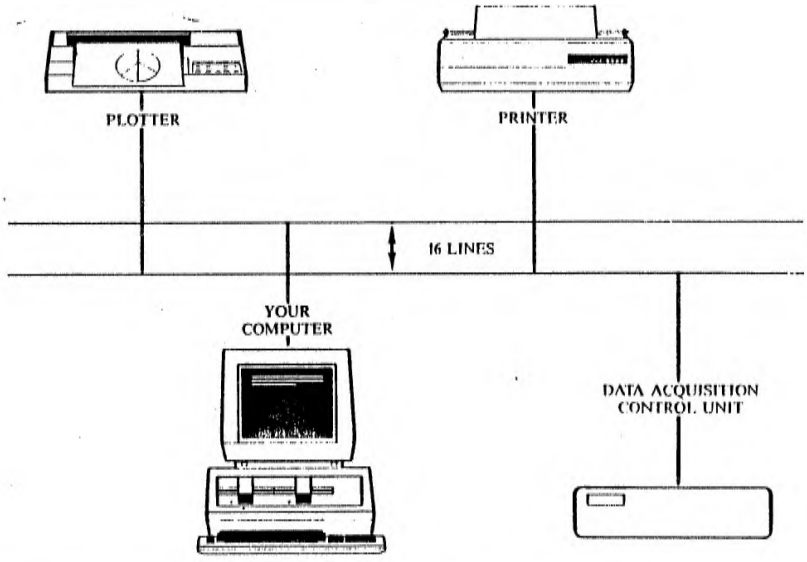
The IEEE 488 interface has the following capabilities and limitations:

- Up to 15 devices can be connected to one contiguous bus.
- Up to 20 meters of total line or bus length (in a star or linear bus) can be used.
- Your computer is capable of a maximum transfer rate of 2000 bytes per second.

FUNCTIONAL EXPLANATION 3.2

You can look at the IEEE 488 bus as a common trunk or communication line that has many devices attached to it (as shown in Exhibit 3a).

Exhibit 3a: Devices on an IEEE 488 Bus



The 16 lines shown in Exhibit 3a consist of eight data lines capable of sending one byte of information at a time; and eight control lines that coordinate the attached devices. Since up to 15 devices can be connected to the bus, control lines are needed to keep all of the devices from trying to communicate at the same time. The control lines also decide when data is available on the data lines and other protocol considerations. Besides the 16 lines used in communication, there are 8 electrical ground lines (not used for communication purposes).

Each device on the bus falls into one or more of the following categories:

- Listener: Receives data from the bus. Examples are printers, tape recorders, and plotters. Since one device must do the talking, there can be up to 14 Listeners on the bus.
- Talker: Transmits data to the other devices on the bus. Examples are tape readers and signal generators. To avoid garbled messages, only one device can talk at a time.
- Controller: An intelligent device that controls the bus by selecting the communicating devices. Only one device—the computer—can be Controller.

3.3 SUPPORTED FUNCTIONS

The IEEE 488 standard lists features or capabilities supported by devices on the bus. Any device can use a set of features from the functions listed in Exhibit 3b.

Exhibit 3b: Functions Supported by IEEE 488

FUNCTION	IEEE MNEMONIC	COMMENTS
Talker, Extended Talker	T TE	These two functions are required for Talkers.
Listener, Extended Listener	L LE	These two functions are required for Listeners.
Source Handshake	SH	Required for transmitting data messages.
Acceptor Handshake	AH	Required for receiving data messages.
Remote/Local	RL	Lets a device choose between two sources of received data. Local data is retrieved from the front panel of the device. Remote data is received from the IEEE bus.
Service Request	SR	Lets a device make an asynchronous request for service from the Controller.
Parallel Poll	PP	Lets a device identify itself when it requires service or when the Controller is asking for a response. PP differs from SR in that PP requires that the Controller regularly conduct a PP. The device cannot spontaneously request the Controller's attention.
Device Clear	DC	Resets a device to its unique idle state.
Device Trigger	DT	When sent by a Talker, DT lets a device begin operation. This allows devices on the bus to be synchronized to each other and/or to an external event.
Controller	C	A Controller sends addresses that determine which devices will be allowed to talk or sends universal commands to any device on the bus. A Controller can also conduct parallel polling to service devices on the bus, and can clear the bus.
Drivers	E	Specifies the electrical interface used by a device.

NOTE: For more information, refer to Chapter 4.

4

IMPLEMENTATION OF IEEE 488 COMMUNICATIONS

DATA LINES 4.1

Data messages to and from devices are transferred by a two-way 8-bit bus that uses 7-bit ASCII code. (Some devices use other data encryption mechanisms for data representation.) Data lines transfer device-specific messages to and from each device, present addresses used in referencing specific devices, and are used for interface commands.

ASCII data is sent down the bus one byte at a time. Each of the eight data lines contains one bit of each byte.

The computer sends all data with space parity (i.e., a parity bit value of zero) and does not check parity on received data. (Received data is set to zero.)

CONTROL LINES 4.2

The eight control lines consist of three handshake lines and five general interface management lines.

The handshake lines coordinate the presentation of data, and give answers to the following questions:

- Is there any data on the bus?
- Is the destination device prepared for more data?
- Was the data sent on the bus received in total by the destination device?

The general interface management lines coordinate the attention of devices on the bus. General interface management lines control the following situations:

- A demand for a device to pay attention.
- A device requesting service of the Controller.
- The Controller resetting the communication session.
- The Controller putting a device in remote programming mode.
- The Controller conducting a parallel poll to check on device status.
- Flagging the end of a message.

4.2.1 HANDSHAKE LINES

The three handshake lines ensure validity of data transmission between sender (Talker) and receiver (Listener). The following items are addressed:

- Rate of Data Transfer: This must be set to the speed of the slowest device in the communication so that Talkers don't transmit faster than Listeners can receive, and so that Listeners accept data only when new values are available.
- Multiple Listeners: Up to 14 Listeners can be involved at once.

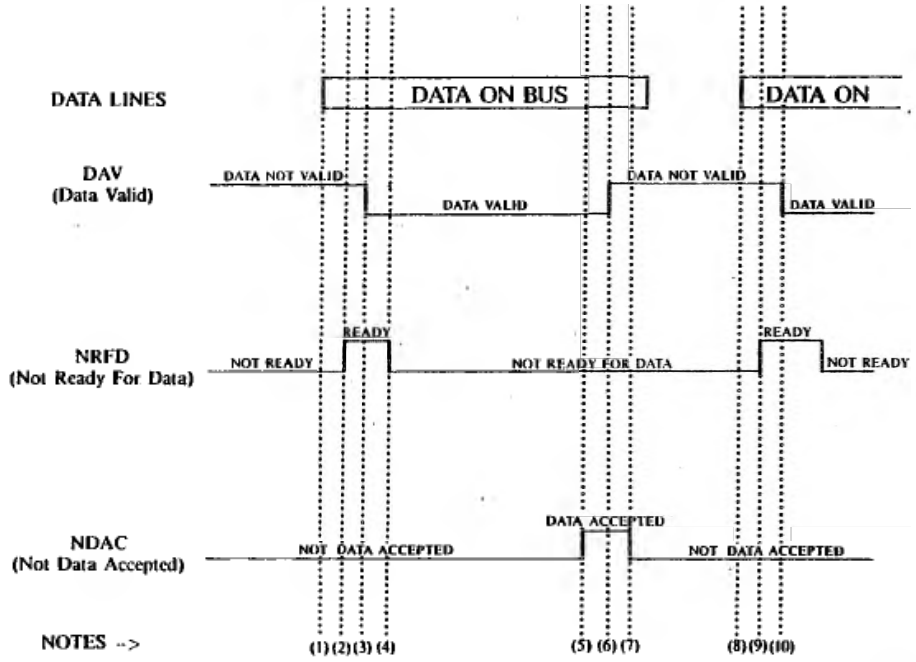
The handshake lines are described in Exhibit 4a.

Exhibit 4a: Handshake Lines

LINE	MNEMONIC	COMMENTS
Data Valid	DAV	Controlled by the Talker or Controller. This line indicates that data is available on the data lines and lets receivers accept data only when new data is available.
Not Ready for Data	NRFD	Controlled by the Listeners. This line tells the data transmitter (Talker) that a device is not ready for the next byte of data. This line helps keep the transmission rate set to the speed of the slowest device receiving the communication.
Not Data Accepted	NDAC	Controlled by the Listeners. This line ensures the valid reception of data. Along with NRFD, it further paces the data transfer rate.

Exhibit 4b illustrates the timing of data transmission and the three handshake signals. Note the reverse logic: high levels represent "false" values. If the Data Valid line is low, the data on the bus is valid.

Exhibit 4b: Data Handshake



When you send data to a device, the line handshaking done by the interface takes place in the following steps:

1. The Talker or Controller puts the first or next byte of data onto the data lines of the IEEE bus.

2. The devices that listen for the data byte become ready for data. (NRFD becomes "ready for data" only when the slowest receiving device is ready.)

3. All appropriate listening devices are ready; Listeners now wait for the Talker. The transmitter of the data asserts DAV to indicate that valid data is on the data lines.

4. Receivers that listen to this data byte read it and then become no longer ready for data. (Since they are processing this data byte, they are not ready for the next byte.)

5. Receivers of data complete internal storing of the data and become ready for the next byte. This is signalled by NDAC going false (the data is accepted). New data is not accepted until the slowest device is ready for the next byte.

6. Since all receivers have acknowledged receipt of the data, the sender can prepare to put the next data byte on the data bus. The Talker responds to this condition by flagging data as no longer valid (DAV high).

7. The receivers see that data is no longer valid (as an acknowledgment to their receipt of the data). They respond by dropping the NDAC line, indicating that data (the next byte) is not yet accepted.

8. When all receivers of data have successfully received the byte, steps 1–7 are repeated for each byte being sent.

GENERAL INTERFACE MANAGEMENT LINES 4.2.2

The general interface management lines further control and coordinate messages on the bus. These lines are used for the following control functions:

- **Addressing and selecting** specific devices to coordinate and choose communicating entities.
- **Broadcasting the mode of the interface** as being in Command mode (where devices must pay attention to demands from the Controller) or in Data mode (where only the receivers of the data pay attention to data).
- **Resetting devices** into a predefined idle state. The idle state is defined by device-specific attributes.

- **Controlling devices** by enabling remote programming, lock-out or enable of front panel switches, or by issuing a synchronizing pulse (enabling multiple devices to perform simultaneous operations).

The general interface management lines are described in Exhibit 4c.

Exhibit 4c: General Interface Management Lines

LINE	MNEMONIC	COMMENTS
Attention	ATN	Used only by the Controller. Tells all devices that the data on the bus is to be interpreted as data (Data mode) or as a command (Command mode).
Interface Clear	IFC	Used by the Controller. Resets and idles the IEEE bus and its communications.
Service Request	SRQ	Used by a device to request service from the Controller.
Remote Enable	REN	Used only by the Controller. Puts a device into Remote Programming mode (i.e., it accepts commands from the communications bus).
End or Identify	EOI	When used by a Talker, EOI flags the last data byte in a message. EOI is also used by the Controller (with ATN) to parallel-poll devices. When a parallel poll is requested, each data line is set by groups of devices to indicate if any of those devices require service.

ATN

All devices constantly check the Attention (ATN) line. When true (low), the interface is in Command mode. In this mode, all devices must accept the data (handshake) presented on the IEEE bus and decode the command being sent. When ATN is false (high), the interface is in Data mode. In this case, the Talker passes data to the selected Listeners. (Listeners are selected by addressing them in Command mode.) In Data mode, device-dependent data (such as programming information for the

device, collected data from the device, or plotter commands) is sent over the bus. The content of this data is defined by the devices being controlled, not specified by IEEE protocol.

Command Mode (ATN true)

There are four basic types of commands in the IEEE protocol:

1. Commands that select the addresses for Talk and Listen devices.

2. Universal commands that request all devices on the bus to perform basic operations (if they are able to perform that operation).

3. Addressed commands that request selected device(s) to perform basic operations.

4. Secondary commands used with one of the above to support multiple units on a single IEEE device.

Each device has an address (or addresses) that identifies it to the Controller, except for devices that only monitor the bus. (A monitoring device is transparent.) These addresses are used by the Controller when determining the Talk and Listen devices. Most devices have an address preset when manufactured; this address can be modified by jumpers, an address selection switch, or a control on the device.

You can modify the five low-order bits of the address to reset a device's address on the bus. This value can be from 0 to 30 decimal (00000 to 11110, binary). Each address is combined with bits 6 and 7 in two ways to provide a unique Talk and Listen address for that device (bit 8 is reserved for parity, and is unused). Bit 7 indicates the Talk address, and bit 6 the Listen address, as listed in Exhibit 4d.

Exhibit 4d: Addresses for Talk and Listen Functions

ADDRESS BITS 5 4 3 2 1	TALK	LISTEN	HEX	DECIMAL
0 0 0 0 0	@	SPACE	00	00
0 0 0 0 1	A	!	01	01
0 0 0 1 0	B	"	02	02
0 0 0 1 1	C	#	03	03
0 0 1 0 0	D	$	04	04
0 0 1 0 1	E	%	05	05
0 0 1 1 0	F	&	06	06
0 0 1 1 1	G	'	07	07
0 1 0 0 0	H	(	08	08
0 1 0 0 1	I	)	09	09
0 1 0 1 0	J	*	0A	10
0 1 0 1 1	K	+	0B	11
0 1 1 0 0	L	,	0C	12
0 1 1 0 1	M	–	0D	13
0 1 1 1 0	N	.	0E	14
0 1 1 1 1	O	/	0F	15
1 0 0 0 0	P	0	10	16
1 0 0 0 1	Q	1	11	17
1 0 0 1 0	R	2	12	18
1 0 0 1 1	S	3	13	19
1 0 1 0 0	T	4	14	20
1 0 1 0 1	U	5	15	21
1 0 1 1 0	V	6	16	22
1 0 1 1 1	W	7	17	23
1 1 0 0 0	X	8	18	24
1 1 0 0 1	Y	9	19	25
1 1 0 1 0	Z	:	1A	26
1 1 0 1 1	[	;	1B	27
1 1 1 0 0	\	<	1C	28
1 1 1 0 1	]	=	1D	29
1 1 1 1 0	^	}	1E	30
1 1 1 1 1	_	?	1F	31

Device addresses can range only from zero to 30. Address 31 is used for two special functions: Untalk (address _) and Unlisten (address ?). Untalk is a shut-up command that tells a device not to talk. Unlisten tells the receiver(s) no longer to receive data.

Your computer is initially defined as address 0 (TALK at "@", LISTEN at " "). This setting can be queried or changed by the application program. Any address switches other than the first five bits are typically used to set a device to Talk or Listen only, or for device-specific test features.

Additional addressing capabilities are provided by some IEEE bus devices. Some devices have extended addressing, which allows addressing of sub-units within the device by secondary commands (i.e., address a particular I/O unit on a Controller, or a selected board for special functions). You can also use secondary commands to extend the number of commands available under the IEEE protocol (such as the PPE and PPD commands).

Other devices support multiple addresses as opposed to extended addresses. This multiple feature lets the base device be addressed as two Talk and two Listen addresses. The low-order bit is not switchable, and the device will accept both values. For example, setting switches to 1, 0, 1, 1 would answer to polls for 16 and 17 hex or 22 and 23 decimal.

Universal Commands

Universal commands are broadcast to all devices on the bus. These commands include Untalk and Unlisten (where the address of 31, decimal, selects no device for Talking or Listening). Five other universal commands exist, in which the data on the bus selects the function (Untalk and Unlisten place the address, 31, on the data bus).

UNTALK (UNT)
Decimal 95, ASCII _

Deselects the current Talker. The sender is also deselected when another device is selected to Talk, or when ATN is asserted.

UNLISTEN (UNL)
Decimal 63, ASCII ?

Deselects the current Listener(s). The receiver(s) no longer accept data.

DEVICE CLEAR (DCL)
Decimal 20, ASCII DC4

All devices supporting DCL return to a device-specific idle state. Devices respond to this command whether supported or not. Actual function and status of each device type is defined by device specifications.

LOCAL LOCKOUT (LLO)
Decimal 17, ASCII DC1

Disables front- or rear-panel device control on devices that recognize the command. Devices respond to this whether it is supported or not. To re-enable the reset/local selection at the device, the REN line must be set false.

SERIAL POLL ENABLE (SPE)
Decimal 24, ASCII CAN

Sets a serial poll mode for all supporting Talker devices. When these devices are addressed to talk, they return a status byte (rather than data).

SERIAL POLL DISABLE (SPD)
Decimal 25, ASCII EM

Ends the serial poll mode begun by SPE. Devices will output data (rather than the status byte) when they become Talkers.

PARALLEL POLL UNCONFIGURE COMMAND (PPU)
Decimal 21, ASCII NAK

Undoes the parallel poll configure command. All devices are returned to their idle, unconfigured state.

The following four universal commands use a single control line. No data is required on the data bus.

INTERFACE CLEAR (IFC)

Used by the system Controller to halt all communications on the bus. Talkers and Listeners are deselected, and Serial Poll is disabled. All devices must constantly monitor the IFC control line.

REMOTE ENABLE (REN)

Used by the system Controller to re-enable devices that were put into local operation. All devices must monitor the bus for remote programming.

ATTENTION (ATN)

Used by the system Controller to set all devices in Command mode (when all monitor the bus for commands) or Data mode (when a Talker and Listener(s) exchange data).

END OR IDENTIFY (EOI)

When ATN is true (Command mode), EOI executes a parallel poll conducted by the system Controller. To respond to a parallel poll, each device puts a value on a pre-defined data line. Each one-bit value indicates whether that device requires service. When ATN is false, EOI is used by the sender (Talker) to flag the end of the data message.

Addressed Commands

These general interface management commands present control functions to specific device(s). All devices selected as Listeners will perform the commands.

GROUP EXECUTE TRIGGER (GET)
Decimal 8, ASCII BS

If a device supports this feature, GET provides a trigger to synchronize its operation with other devices or with external events. When a device receives the GET command, it performs its set function.

SELECTED DEVICE CLEAR (SDC)
Decimal 4, ASCII EOT

Much as device clear (DCL) resets all devices to a device-dependent idle state, SDC sets the device(s) selected as Listener to that idle state.

GO TO LOCAL (GTL)
Decimal 1, ASCII SOH

Listener(s) leave the remote programming state and get information from the front control panel. They return to Remote Programming mode when further information is sent.

PARALLEL POLL CONFIGURE (PPC)
Decimal 5, ASCII ENQ

Tells a device that a configuration for a parallel poll is about to occur. PPE is used for the actual configuration.

Secondary Commands

PARALLEL POLL ENABLE (PPE)
Decimal 96-111, ASCII ‘ and a-o

Assigns a DIO (data) line to a device for transmission of its response to a parallel poll. Most devices use jumpers or local switches to select their parallel poll data line assignment. For devices without this feature, the PPC is required for parallel polling. The Parallel Poll Unconfigure command (PPU) resets the configuration provided by the PPC command.

PARALLEL POLL DISABLE (PPD)
Decimal 112, ASCII p

Keeps devices addressed by the PPC command from answering the parallel poll.

TAKE CONTROL (TCT)
Decimal 9, ASCII HT

Allows a selected Listener to become the new bus Controller. Since your computer is always the bus (system) Controller of the IEEE 488 interface, this command is not used.

NOTIFICATION OF SERVICE REQUEST 4.3

Sometimes a device needs to tell the Controller that it needs some attention. To do this, the device can set a line indicating that it needs service or it can wait until the Controller asks if service is required.

SERVICE REQUEST (SRQ) 4.3.1

The service request (SRQ) line is set by a device on the bus to tell the Controller that it needs service. A device might set SRQ because it is ready to transmit data or because an error condition has occurred.

The SRQ doesn't tell the Controller which device needs attention. If there is more than one device on the bus, the Controller must poll the devices to determine which one asked for service.

POLLING OF DEVICES 4.3.2

A serial poll lets the Controller ask a device if it requires service or retrieve status information from that device. A talking device returns its status byte when serially polled. From this, the Controller can tell if the device needs service (or any other status information about that device). Typically, the Controller serially polls each device on the bus to update all device-state information.

The Controller issues a parallel poll as simultaneous true ATN and EOI lines; it retrieves status information from all devices on the bus at once. In response to a parallel poll, each device returns a single bit of status (telling the Controller whether the device requires service) on the data (DIO) lines. One or more devices can be assigned to each DIO line. This lets the Controller use a parallel poll to isolate a group of devices in which one or more may require service.

4.4 SPECIFICATIONS OF DEVICE CAPABILITIES

The encoding scheme presented in this section defines the IEEE capability set supported by a communicating device. Since the IEEE bus allows a wide variety of devices, you should know both the communication limitations and functional limitations of each device. There are eleven functional groups for IEEE communication; each group specifies ranges of capabilities.

Two new abbreviations appear in this section—MLA (My Listen Address) and MTA (My Talk Address). These refer to the Talk and Listen addresses under which your computer operates.

SOURCE HANDSHAKE (SH)

SH0: No capability
SH1: Full capability

ACCEPTOR HANDSHAKE (AH)

AH0: No capability
AH1: Full capability

TALKER (T) and EXTENDED TALKER (TE)

	BASIC TALKER	SERIAL POLL	TALK ONLY MODE	UNADDRESSED IF MLA
T0	NO	NO	NO	NO
T1	YES	YES	YES	NO
T2	YES	YES	NO	NO
T3	YES	NO	YES	NO
T4	YES	NO	NO	NO
T5	YES	YES	YES	YES
T6	YES	YES	NO	YES
T7	YES	NO	YES	YES
T8	YES	YES	NO	YES

LISTENER (L) and EXTENDED LISTENER (LE)

	BASIC LISTENER	LISTEN ONLY MODE	UNADDRESSED IF MTA
L0	NO	NO	NO
L1	YES	YES	NO
L2	YES	NO	NO
L3	YES	YES	YES
L4	YES	NO	YES

PARALLEL POLL (PP)

PP0: No capability
PP1: Remote configuration
PP2: Local configuration

SERVICE REQUEST (SR)

SR0: No capability
SR1: Full capability

REMOTE LOCAL (RL)

RL0: No capability
RL1: Full capability
RL2: No local lockout

DEVICE CLEAR

DC0: No capability
DC1: Full capability
DC2: Omit selective device CLEAR

DEVICE TRIGGER (DT)

DT0: No capability
DT1: Full capability

DRIVER ELECTRONICS (E)

E1: Open collector
E2: Tri-state

CONTROLLER (C)

A Controller is specified by one or more of C1 to C4, and one of the range C5 to C28.

	System Controller	Send IFC and Take Charge	Send REN	Respond to SRQ	Send Interface Messages	Receive Control	Pass Control	Pass Control to Self	Parallel Poll	Take Control Synchronously
C0	N	N	N	N	N	N	N	N	N	N
C1	Y	-	-	-	-	-	-	-	-	-
C2	-	Y	-	-	-	-	-	-	-	-
C3	-	-	Y	-	-	-	-	-	-	-
C4	-	-	-	Y	-	-	-	-	-	-
C5	-	-	-	-	Y	Y	Y	Y	Y	Y
C6	-	-	-	-	Y	Y	Y	Y	Y	N
C7	-	-	-	-	Y	Y	Y	Y	N	Y
C8	-	-	-	-	Y	Y	Y	Y	N	N
C9	-	-	-	-	Y	Y	Y	N	Y	Y
C10	-	-	-	-	Y	Y	Y	N	Y	N
C11	-	-	-	-	Y	Y	Y	N	N	Y
C12	-	-	-	-	Y	Y	Y	N	N	N
C13	-	-	-	-	Y	Y	N	N	Y	Y
C14	-	-	-	-	Y	Y	N	N	Y	N
C15	-	-	-	-	Y	Y	N	N	N	Y
C16	-	-	-	-	Y	Y	N	N	N	N
C17	-	-	-	-	Y	N	Y	Y	Y	Y
C18	-	-	-	-	Y	N	Y	Y	Y	N
C19	-	-	-	-	Y	N	Y	Y	N	Y
C20	-	-	-	-	Y	N	Y	Y	N	N
C21	-	-	-	-	Y	N	Y	N	Y	Y
C22	-	-	-	-	Y	N	Y	N	Y	N
C23	-	-	-	-	Y	N	Y	N	N	Y
C24	-	-	-	-	Y	N	Y	N	N	N
C25	-	-	-	-	Y	N	N	N	Y	Y
C26	-	-	-	-	Y	N	N	N	Y	N
C27	-	-	-	-	Y	N	N	N	N	Y
C28	-	-	-	-	Y	N	N	N	N	N

#31

Your computer is the bus Controller. To ensure the integrity of the IEEE bus, the computer never passes control (i.e., issues a TCT) to another device. As Controller, the computer has none of the non-Controller capabilities (SR0, RL0, PP0, DC0, DT0). Your computer does have the following capabilities:

- Complete Source and Acceptor Handshake (SH1, AH1).
- Basic Talker and Listener (T8, L4).
- Active System Controller Ability with service request response (C1, C2, C3, C4, C25).

Your computer is initially set up as device address 0 (TALK at "@", LISTEN at " "). This address can be queried or changed by the application program.

HIGH-LEVEL IEEE 488 FUNCTIONS

This chapter explains the routines used to communicate with devices on the IEEE 488 bus interface. It also gives instructions on how to set up an IEEE 488 bus, and provides programming examples.

OVERVIEW OF FUNCTIONS 5.1

In this section, the functions marked with an asterisk are the ones you'll use most often (unmarked functions are used in unusual applications).

GETTING CONTROL OVER A DEVICE 5.1.1

*REMOTE: Tells a device to accept remote programming from the IEEE bus. This is accomplished through the remote enable (REN) command, if the remote programming is to be a universal command; or by directing specific devices to Listen for non-universal applications.

*LOCAL LOCKOUT: Disables a device's control switch to keep it from interfering with the control signals. A typical use of this feature is in setting a device in Remote and Local Lockout—the Local Lockout ensures that remote programming is not interfered with. This command uses the LLO bus command.

*LOCAL: For a single device, a Go to Local enables that device to get input from its panel switches. With a universal command, the REN line is made false to disable remote programming for all devices.

5.1.2 FUNDAMENTAL DEVICE COMMUNICATIONS

*OUTPUT: Sends data bytes or programming information to a device(s) using the Data mode to transfer the message. Examples include plotter commands or printer output. The contents and values of the data are strictly device-dependent. Parity bit values of zero (space) are sent on all data bytes.

*ENTER: Retrieves data bytes from a device using the Data mode to transfer the message. Examples include data from a signal generator, or acquisition devices. The contents and values of the data are strictly device-dependent. Data parity is not checked, and is returned as 0 to the application.

*TRIGGER: Causes the Controller to send a GET command. This group-execute trigger starts operations in one or more devices and synchronizes them to each other and/or an external event. When finished, ATN remains true (see RESUME).

5.1.3 DEVICE POLLING AND RESETTING

*SPOLL: Lets the application perform a serial poll (SP) of a specific device. The status returned by the device is strictly device-dependent.

*PPOLL: Does a parallel poll (ATN and EOI lines both true) and returns the response byte to the requesting program.

*CLEAR: Returns selected device(s) to the idle state. Uses the Device Clear (DCL) for universal commands or Selected Device Clear (SDC) to clear a single device.

ABORTIO: Stops all communications. Untalks any active Talkers. Uses the IFC command.

RESET: Provides a hardware reset of the entire IEEE 488 bus.

*TIMEOUT: Lets the program set timeouts to prevent the computer from waiting for non-responsive or absent devices. The application can set the time limit in which a device must respond (before the device is considered malfunctioning and reported to the application). This function also lets the application query the present timeout value.

ADVANCED IEEE BUS FUNCTIONS 5.2

*STATUS: Retrieves the present status of the IEEE interface. Status values are listed and explained in the routine's calling sequence.

SEND: These functions allow application programmers to create their own commands. (Explanations of available commands are given in previous sections.) Calling sequences for the SEND functions are explained in Section 5.3, "Calling Sequences for IEEE 488 High-Level Routines."

DIRECT BUS I/O: For advanced users only. This lets the program write and read bus data and control lines without considering the present state or protocol. A direct write or read of bus lines is performed. A possible use of this function is a diagnostic program manipulating the bus directly. CAUTION: If used carelessly, this function can cause loss of data or (under certain conditions) hardware damage.

END OF LINE: Lets the knowledgeable programmer select an end of line sequence that is automatically inserted at the end of each transmitted data sequence by the IEEE drivers. A second end of line sequence is used to recognize the end of input data. You can ask for the currently active end of line sequences.

CONTROLLER ADDRESS: The computer is initially configured to be at device address 0—TALK "@" and LISTEN " ". This function lets the application reset the computer's address to any other valid device address (0 to 30), and ask for the value of the computer's address.

RESUME: Sets ATN false and puts the bus into Data mode. This is useful after functions such as Clear, which leaves the bus in Control

mode (ATN true). Resume is required only in special cases, since normal data transfer statements (Output, Enter, and Transfer) automatically set ATN false.

5.3 CALLING SEQUENCES FOR HIGH-LEVEL ROUTINES

This section defines the calling sequences for the routines available to the application programmer. The parameters and calls are described in a language-independent style. Mechanisms exist for interfacing these routines to any computer-supported language (as explained in Section 5.4, "High-Level Support for Programming Languages").

5.3.1 PARAMETERS USED

DEVICE SELECTOR

Value: Integer

Chooses the device(s) affected by the function. The integer must be in the range 00 to 30. If the integer is 31, DEVICE SELECTOR is a "Universal Address" (the command affects all devices on the bus).

DEVICE COUNT

Value: Integer

Used with a multiple device selector or multiple device secondary address. The integer is the number of devices to address, or the number of device secondary addresses to send. The integer must be in the range 1 to 31.

MULTIPLE DEVICE SELECTOR

Value: Array of 31 (maximum) Integers

Chooses multiple specific devices (as opposed to universal) affected by this command. The array contains 1 to 31 integer values (as specified by the first parameter) in contiguous array positions 1 to n. These are device selectors. Device selectors of value 31 (universal) are illegal.

DEVICE SECONDARY ADDRESS

Value: Integer

Chooses the secondary address used with the selected device. The secondary address is specific to the device and represents particular functions performed by that device. The integer must be in the range 0 to 31.

MULTIPLE DEVICE SECONDARY ADDRESS

Value: Array of 31 (maximum) Integers

Chooses a secondary address for the multiple specific devices affected by this command. The array contains 1 to 31 integer values in contiguous array positions 1 to n. (These are device secondary addresses.)

OUTPUT COUNT

Value: Integer

Used with an output buffer. The integer value is the number of bytes contained in a message to send.

OUTPUT BUFFER

Value: Pointer (Segment and Offset)

This is the memory location of a buffer that contains data to send on the interface. Data length is determined by the OUTPUT COUNT parameter. Parity bits on the data are stripped (providing space parity) before transmission.

INFORMATION TYPE

Value: Integer

This determines the state of the ATN line during output. If the value is 1, the output is a command and the ATN bit on the control lines is set high. Otherwise, the output information is data, and the ATN bit is set low.

INPUT COUNT

Value: Integer

Used with an input buffer. INPUT COUNT is the maximum number of data bytes that can be received. When a data transfer is finished, INPUT COUNT contains the actual data length retrieved.

INPUT BUFFER

Value: Pointer (Segment and Offset)

This is the memory location of a buffer that contains the data to be sent from the desired device. Parity bits are masked off (set to zero) by the drivers (and not checked) before being returned to the application.

RETURN STATUS

Value: Integer

Returns the success or error status of the requested function.

5.3.2 ROUTINES AVAILABLE

Routines of general interest are marked with an asterisk. Unmarked routines are available for extraordinary or advanced application requirements.

REMOTE *

Parameters: DEVICE SELECTOR, RETURN STATUS

If DEVICE SELECTOR is a universal address (31 decimal), the REN line is set to true, enabling all devices to accept programming from the communications bus; devices do not go into Remote state until they are addressed to listen. If DEVICE SELECTOR is a device address, that device is instructed to the IEEE bus by using the following sequence: (1) REN is set true; (2) an Unlisten command is sent, disabling all other Listeners; and (3) a Listen is sent to the specified device. ATN is left true. (Use Resume if Data mode is desired.)

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE REMOTE *

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, RETURN STATUS

Similar to Remote, except that this routine lets you address multiple devices to Listen. A universal device address is not allowed. Devices are instructed to listen by using the following sequence: (1) REN is set true; (2) an Unlisten command is sent to disable all other Listeners; and (3) Listens are sent to the specified devices. ATN is left true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

LOCAL LOCKOUT *

Parameters: RETURN STATUS

Sends a Local Lockout (LLO) to prevent all previously addressed devices on the bus from using their reset/return to local mode capabilities. This ensures that the interface has exclusive control over the device. ATN is left true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

LOCAL *

Parameters: DEVICE SELECTOR, RETURN STATUS

If DEVICE SELECTOR is a universal address (31 decimal), remote enable (REN) is set false. All devices allow you to control them through their local switches. If DEVICE SELECTOR is a single device address, all devices except that device are unlistened. Then the specified device is selected to listen, and a Go to Local (GTL) command is sent to that device. In this case, only that device is set to expect local operator commands. ATN is left true.

NOTE: A GTL is necessary before you can establish local control of a device set to Remote and Local Lockout.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE LOCAL *

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, RETURN STATUS

Similar to Local, except that this routine lets you place multiple single devices in Local mode. A universal device address is not allowed. A UNL command is sent to unlisten all devices, the specified devices are addressed to listen, and then a Go to Local (GTL) command sets them to expect local operator commands. ATN is left true.

NOTE: A GTL is needed before establishing local panel control of a device that has been set to Remote and Local Lockout.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

OUTPUT *

Parameters: DEVICE SELECTOR, OUTPUT COUNT, OUTPUT BUFFER, RETURN STATUS

If DEVICE SELECTOR is a universal address, OUTPUT goes to all devices addressed to Listen. Otherwise, the following sequence sends the data to the particular device selected:

1. Send the Controller's Talk address.
2. Send a UNL command.
3. Address the specified device to Listen.
4. Output the data.

(If you select a universal device, steps 2 and 3 are unnecessary.)

Data bytes are retrieved from the output buffer for a length specified by OUTPUT COUNT. An end of line sequence is sent after the last byte transferred.

If you are using a device that requires a secondary address, you must use SEND SCG followed by OUTPUT with a universal device selector. You can also use SEND SCG followed by SEND CMD or SEND DATA, depending on the type of information you want to output. If any other sequence is used, the secondary address is not sent to the device.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE OUTPUT *

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, OUTPUT COUNT, OUTPUT BUFFER, INFORMATION TYPE, RETURN STATUS

Same as OUTPUT, except that DEVICE SELECTOR cannot be a universal address or a device requiring a secondary address. The output data is sent to all devices specified by using the same sequence as OUTPUT.

If you want to output to several devices, some of which need a secondary address, use SEND SCG, MULTIPLE SEND SCG to set the devices to Listen and send the secondary address. You can use SEND LISTEN or MULTIPLE SEND LISTEN for devices that do not need a secondary address. Follow these with calls to OUTPUT, SEND CMD, or SEND DATA as specified in OUTPUT.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

ENTER *

Parameters: DEVICE SELECTOR, INPUT COUNT, INPUT BUFFER, RETURN STATUS

If DEVICE SELECTOR is a universal address, data is retrieved from any device set to Talk. Otherwise, the device is selected to Talk, the Controller is selected to Listen, ATN is set false to resume Data mode, and data is retrieved from the specified device and placed in the input buffer. A maximum data length of INPUT COUNT is placed into the buffer. When finished, INPUT COUNT contains the actual data count retrieved from the device.

When a return status of 3 occurs (too much data to fit in the user's buffer), you can retrieve the next group of bytes by doing another Enter with a universal address specified. If more data is not wanted, an ABORTIO is recommended.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error
3 means too much data was sent from the device for the size of the input buffer

INPUT COUNT equal to the number of bytes (characters) of data received.

INPUT BUFFER equal to the data bytes received from the device selected.

TRIGGER *

Parameters: DEVICE SELECTOR, RETURN STATUS

Sends a GET command. If the device selected is universal, devices previously addressed to Listen are sent the command. Otherwise, all other devices are unlistened and only the device selected operates on the GET. Trigger leaves ATN true, and a Resume can be used to placed the bus in Data mode.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE TRIGGER *

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, RETURN STATUS

Same as Trigger, except that multiple devices can be specified as the receiver of the GET. A universal address cannot be used.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SPOLL *

Parameters: DEVICE SELECTOR, POLL BYTE, RETURN STATUS

A serial poll of the specified device is conducted by unlistening all other devices, sending a Serial Poll Enable (SPE) resuming Data mode, retrieving the poll byte, and sending a Serial Poll Disable (SPD). The resulting status is returned as an integer value in POLL BYTE. A universal address cannot be used.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

POLL BYTE is the status returned by the polled device (its value is device-dependent).

PPOLL *

Parameters: POLL BYTE, RETURN STATUS

Conducts a parallel poll of all devices on the bus. The resulting status byte (from all devices) is returned in the integer, POLL BYTE.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

POLL BYTE is the resulting value from the parallel poll. It contains the 8-bit (byte) value from the data lines.

CLEAR *

Parameters: DEVICE SELECTOR, RETURN STATUS

If the device selector is a universal address, the Device Clear (DCL) command is issued, resetting every device on the bus. If DEVICE SELECTOR is a single device address, all other devices are unlistened and a Selected Device Clear (SDC) is issued to the specified device. Leaves ATN true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE CLEAR *

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, RETURN STATUS

Same as Clear, except that the Selected Device Clear (SDC) is sent to each chosen device. A universal address cannot be used.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

ABORTIO

Parameters: None

Stops all communications activity and Untalks any active Talkers. Issues an IFC to clear all communications on the bus.

RESET

Parameters: None

Resets the IEEE interface to its initial power-on state. All IEEE driver parameters are reset to their initial (default) values. IFC and then REN are sent.

The IEEE default values follow:

Controller = 0.
Timeout = 0 (wait forever).
End of input = 0 specifying <cr><lf> (end of line in).
End of output = <cr><lf> (end of line out).
Length of end of output = 2.
End of output condition = 0 (No EOI at the end of the data).

TIMEOUT *

Parameters: FUNCTION, MILLISECONDS, RETURN STATUS

Lets the application manipulate the timeout value. The timeout value is the number of milliseconds (specified as an integer from 0 to 32,767 in the MILLISECONDS parameter) that the IEEE driver waits before a device is considered malfunctioning.

If FUNCTION is 0, then the TIMEOUT call asks for the set timeout value (returned in MILLISECONDS). If FUNCTION is 1, then the timeout value is set to MILLISECONDS. A MILLISECONDS value of 0 means that the driver should wait forever.

When a timeout does occur, a communication error is returned to the active caller in the RETURN STATUS parameter. The initial timeout value is 0 (wait forever).

Returns: RETURN STATUS of 0 means success
1 means bad parameter

MILLISECONDS (value depends on FUNCTION).

STATUS*

Parameters: SERVICE REQUEST

Retrieves interface status information into Service Request.

SERVICE REQUEST remains at the status of the SRQ line, indicating whether a device on the bus requires the Controller's attention.

Returns: SERVICE REQUEST 0 = no current SRQ
1 = SRQ line is true

SEND CMD

Parameters: OUTPUT COUNT, OUTPUT BUFFER, RETURN STATUS

The data specified in OUTPUT BUFFER is sent as a control message, allowing the user to create custom command sequences as required. ATN is left true. You should already have done any needed device addressing.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND DATA

Parameters: OUTPUT COUNT, OUTPUT BUFFER, RETURN STATUS

The data specified in OUTPUT BUFFER is sent as a data message, letting you create custom data sequences without any control from the IEEE driver. If no end of line sequence is desired, you must use END OF LINE OUT to set the parameters correctly. If this is not done, an end of line sequence is sent automatically. The computer is addressed to Talk; any other device addressing should have been done.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND TALK

Parameters: DEVICE SELECTOR, RETURN STATUS

Sends a Talk command to the selected device. ATN is left true. DEVICE SELECTOR cannot be a universal address.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND LISTEN

Parameters: DEVICE SELECTOR, RETURN STATUS

A Listen command is sent to the selected device. ATN is left true. DEVICE SELECTOR cannot be a universal address.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE SEND LISTEN

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, RETURN STATUS

Listen commands are sent to the selected devices. ATN is left true. DEVICE SELECTOR cannot be a universal command.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND SCG

Parameters: DEVICE SELECTOR, DEVICE SECONDARY ADDRESS, RETURN STATUS

Send Secondary Command Group. The device specified by DEVICE SELECTOR is addressed to Listen and sent the DEVICE SECONDARY ADDRESS as a secondary command address. ATN is left true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

MULTIPLE SEND SCG

Parameters: DEVICE COUNT, MULTIPLE DEVICE SELECTOR, MULTIPLE DEVICE SECONDARY ADDRESS, RETURN STATUS

Multiple Send Secondary Command Group. Each device specified in MULTIPLE DEVICE SELECTOR is addressed to Listen and is then sent to the corresponding DEVICE SECONDARY ADDRESS in the MULTIPLE DEVICE SECONDARY ADDRESS array as a secondary command address. A device is not unlistened by addressing subsequent devices in the array. ATN is left true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND UNL

Parameters: RETURN STATUS

Sends an Unlisten command. ATN is left true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND UNT

Parameters: RETURN STATUS

Sends an Untalk command. ATN is left true.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND MLA

Parameters: RETURN STATUS

Sends My Listen Address (the address of your computer). ATN is left true. Use RESUME to enter Data mode.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

SEND MTA

Parameters: RETURN STATUS

Sends My Talk Address (the address of your computer). ATN is left true. Use RESUME to enter Data mode.

Returns: RETURN STATUS of 0 means success
1 means bad parameter
2 means communication error

DIRECT BUS IO

Parameters: FUNCTION, WHICH LINE, POLL BYTE, RETURN STATUS

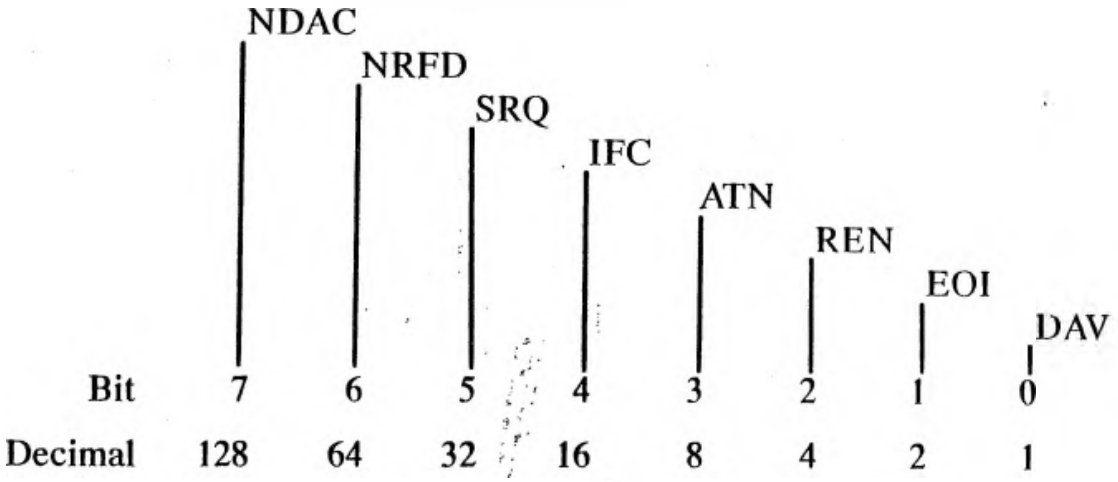
FUNCTION indicates whether this is a read or write of the control or data lines.

FUNCTION of 0 is a read: The value of POLL BYTE (a single byte) is retrieved from the control or data lines as indicated by the integer parameter, WHICH LINE (0=control, 1=data lines). ATN is left false when data lines are read; ATN is unchanged for control lines.

FUNCTION of 1 is a write: The value of POLL BYTE (always a single byte) is given to the control or data lines as indicated by WHICH LINE (0=control, 1=data lines). ATN is left false when data lines are written.

CAUTION: Only experienced programmers should use this function. Careless use can garble your communication session or damage hardware.

The format of POLL BYTE for control lines is:



To set SRQ, you can set bit 5 on or write a decimal 32 to the lines.

Returns: RETURN STATUS of 0 means success
1 means bad parameter

POLL BYTE, depending on FUNCTION.

END OF LINE OUT

Parameters: FUNCTION, EOI, EOL LENGTH, EOL DATA, RETURN STATUS

This function lets you manipulate the end of line sequence which the IEEE driver inserts at the end of each data line. If FUNCTION is 0, the call asks for the values of the end of line parameters. If FUNCTION is 1, the call sets the end of line sequence as specified.

The parameters are:

EOI — Integer. If 0, no EOI line setting at end of data. If 1, EOI line is set true at the end of data. This value is initially set at 0 (no EOI).

EOL LENGTH — Integer. Length of the end of line sequence. Initially 2 for a carriage return/line feed sequence.

EOL DATA — Integer array having a length of 7. The end of line sequence is inserted by the driver at end of data. This parameter is initially set to <cr><lf> (hexadecimal 0D, 0A) and five nulls.

NOTE: This integer array specifies the end of line sequence up to the length specified in EOL LENGTH.

Returns: RETURN STATUS of 0 means success
1 means bad parameter

EOI, EOL LENGTH, EOL DATA (depending on call).

END OF LINE IN

Parameters: FUNCTION, EOL, RETURN STATUS

This routine lets the application program specify the sequence used as an input data terminator. This end of line sequence is used by the IEEE driver to determine when an Enter is complete. If 0, FUNCTION asks for the current END OF LINE IN sequence. If 1, FUNCTION sets the END OF LINE IN sequence as indicated.

EOL is the selection for the input end of line sequence:

EOL = 0 Carriage Return/Line Feed
1 EOI line flags end of data
2 Either CR/LF or EOI is required

Initially, EOL is set to 0 (default value is the carriage return/line feed sequence).

Returns: RETURN STATUS of 0 means success
1 means invalid parameter

EOL, depending on FUNCTION

CONTROLLER ADDRESS

Parameters: FUNCTION, DEVICE SELECTOR, RETURN STATUS

This function lets you manipulate the computer's device address. If FUNCTION is 0, the call asks for the value of the Controller's device address. If FUNCTION is 1, the call sets the device address. DEVICE SELECTOR specifies the device address; it cannot be a universal address.

The initial value of the computer's address is 0.

Returns: RETURN STATUS of 0 means success
1 means bad parameter

DEVICE SELECTOR, depending on FUNCTION.

RESUME

Parameters: None

Sets ATN false, putting the bus into Data mode.

HIGH-LEVEL SUPPORT FROM PROGRAMMING LANGUAGES 5.4

The IEEE 488 High-Level drivers are accessible from all supported languages except COBOL and FORTRAN. Sets of High-Level modules are provided to overcome incompatibility in the procedure call/return mechanisms in these languages. A particular selection of High-Level modules must match the application language being used.

You must use the operating system included in this package when using the IEEE High-Level package. This operating system is the only one that supports the IEEE Low-Level drivers in the BIOS. Following application development, one of the following modules must be linked into your program (except in the case of interpretive BASIC and GW BASIC).

5.4.1 ASSEMBLY LANGUAGE AND PL/M

Module: IEEEASM.LIB

Using assembly language, push the segment and offset of each parameter on the stack and perform a CALL of the desired high-level routine. Each segment and offset pushed on the stack points to the variable's data structure (16-bit integer, character string, or array of 31 consecutive 16-bit integers). For return variables, the variable itself is modified by the routines.

After you have compiled your routine, you must link it. Enter the IEEEASM.LIB module when an additional library is requested by the linker. This library contains the IEEE high-level routines plus the module that these routines use to access the BIOS low-level routines.

5.4.2 INTERPRETIVE BASIC AND GW BASIC

Module: IEEEBAS.COM

Before the IEEE 488 interface can be accessed from interpretive BASIC or GW BASIC, you must load the IEEE/BASIC interface module. To do this, type:

```
IEEEBAS
```

before running BASIC. This loads the module into memory and displays the message:

```
IEEE 488/BASIC interface ready - version x.x
```

The interface must be loaded before you try to access the IEEE interface from a BASIC program. However, you only need to load it once during an operating session.

After the IEEE/BASIC interface is loaded, programs written in MS-BASIC or GW BASIC can access devices on the bus by using the CALL statement. Before you can use CALL, however, you must define the segment and offset of the IEEE 488 interface entry point with the DEF SEG statement. The entry address of the IEEE/BASIC interface is found in interrupt vector table entry 221 (the four bytes of memory starting at address 0:884). By PEEKing these four bytes, you can construct the entry address of the IEEE routines. The following sample code illustrates this procedure.

```
5  'GET IEEE 488 ENTRY POINT FROM INTERRUPT VECTOR
   ENTRY 221
10 DEF SEG = 0
20 LOWOFF = PEEK(884)      'Low byte of offset address
30 HIOFF = PEEK(885)       'High byte of offset address
40 LOWSEG = PEEK(886)      'Low byte segment address
50 HISEG = PEEK(887)       'High byte segment address
60 SEG = (256*HISEG) + LOWSEG      'Full segment address
70 IEEE = (256*HIOFF) + LOWOFF     'Full offset address
80 DEF SEG = SEG     'Define segment for CALL
```

After you execute these statements, call the IEEE interface from BASIC with a statement in this form:

CALL IEEE(PARAMETER1,...,PARAMETERn,FUNCTION)

where:

FUNCTION is the number of the desired IEEE interface function.

PARAMETER1,...,PARAMETERn are the parameters required by this function.

To initialize a Hewlett-Packard 7470A plotter, for example, you must send the sequence 'IN;' as data to the plotter on the bus. The following BASIC program sends this sequence (assuming that the plotter is at bus address 5).

```
100 INFO% = 0
110 OUTPUT% = 5      'Output command
120 DEVICE% = 5      '7470A plotter bus address
130 MSG$ = "IN;"
140 CNT% = LEN(MSG$)
150 RETSTAT% = 0
160 CALL IEEE(DEVICE%, CNT%, MSG$, INFO%, RETSTAT%,
    OUTPUT%)
```

This example also shows two other important points. First, all the numeric variables must be defined as integers, either by appending a percent sign to their names, or by using the DEFINT statement. Second, parameters must be passed as variables for the interface to work correctly.

Exhibit 5a lists function values for interpretive BASIC.

Exhibit 5a: Interpretive BASIC Function Values

FUNCTION NUMBER	HIGH-LEVEL FUNCTION
0	REMOTE
1	MULTIPLE REMOTE
2	LOCAL LOCKOUT
3	LOCAL
4	MULTIPLE LOCAL
5	OUTPUT
6	MULTIPLE OUTPUT
7	ENTER
8	TRIGGER
9	MULTIPLE TRIGGER
10	SPOLL
11	PPOLL
12	CLEAR
13	MULTIPLE CLEAR
14	ABORTIO
15	RESET
16	TIMEOUT
17	STATUS
18	SEND CMD
19	SEND DATA
20	SEND TALK
21	SEND LISTEN
22	MULTIPLE SEND LISTEN
23	SEND SCG
24	MULTIPLE SEND SCG
25	SEND UNL
26	SEND UNT
27	SEND MLA
28	SEND MTA
29	DIRECT BUS IO
30	END OF LINE OUT
31	END OF LINE IN
32	CONTROLLER ADDRESS
33	RESUME

5.4.3 COMPILED BASIC

Module: IEEECBAS.LIB

This library contains the following:

- IEEE interface module that allows access to the high-level routines from compiled BASIC.
- IEEE high-level routines.
- An interface from the high-level routines to the low-level BIOS routines.

Use the following format in a CALLS statement when you want to use IEEE 488 with compiled BASIC:

5

CALLS IEEE (parameter1,...parameterN, function)

where:

IEEE is the routine in the IEEECBAS library that interfaces to IEEE high-level routines.

parameter1,...parameterN are the parameters specific to the high-level routine being accessed.

function is the number assigned to the high-level routine. See Exhibit 5a for a list of these numbers.

After writing your program, use the following procedure to link your routine and the IEEE routines (filename is the name of your routine).

LINK <filename>
Run File [A:filename]:(user discretion)
List File [nul.MAP]: (user discretion)
Libraries [.LIB]: IEEECBAS.LIB

Module: IEEEPAS.LIB

Any IEEE high-level routines that you use must be declared as external procedures in your program. Following is a list of the external declarations for each of the high-level routines. In these declarations, the following are assumed to be user-declared types:

```
Device_array : Array [0..31] of Integer;
Eol_array : Array [0..6] of Integer;
Procedure IEEE_ABORTIO; External;

Procedure IEEE_CLEAR (VARS device_selector,
                      return_status : Integer); External;
Procedure IEEE_MULTIPLE_CLEAR (VARS device_count ; Integer;
                               VARS multiple_device_selector :
                                    device_array;
                               VARS return_status : Integer);
                                    External;
Procedure IEEE_CONTROLLER_ADDRESS (VARS function, device_selector,
                                        return_status : Integer);
                                        External;
Procedure IEEE_DIRECT_BUS_IO (VARS function, poll_byte, return_status :
                                   Integer); External;
Procedure IEEE_END_OF_LINE_IN (VARS function, input_terminator,
                                    return_status : Integer);
                                    External;
Procedure IEEE_END_OF_LINE_OUT (VARS function, output_condition :
                                     Integer;
                                VARS output_terminator : eol array;
                                VARS return_status : Integer);
                                     External;
Procedure IEEE_LOCAL (VARS device_selector, return status : Integer);
                      External;
Procedure IEEE_MUTIPLE_LOCAL (VARS device_count ; Integer;
                              VARS multiple_device_selector :
                                   Device_array;
                              VARS return_status : Integer);
                                   External;
Procedure IEEE_LOCAL_LOCKOUT (VARS return_status : Integer);
                                   External;
```

```
Procedure IEEE_OUTPUT (VARS device_selector, output_count : Integer;
                       VARS output_buffer : String;
                       VARS return_status : Integer); External;
Procedure IEEE_MULTIPLE_OUTPUT (VARS device_count : Integer;
                                VARS multiple_device_selector :
                                     Device array
                                VARS count : Integer;
                                VARS output_buffer : String;
                                VARS info-type : Integer;
                                VARS return_status : Integer);
                                     External;
Procedure IEEE_PPOLL (VARS poll_byte,
                      return_status : Integer); External;
Procedure IEEE_REMOTE (VARS device_selector,
                       return_status : Integer); External;
Procedure IEEE_MULTIPLE_REMOTE (VARS device_count : Integer;
                                VARS multiple_device_selector :
                                device array;
                                VARS return_status : Integer);
                                     External;
Procedure IEEE_RESET; External;
Procedure IEEE_RESUME; External;
Procedure IEEE_SEND_CMD (VARS output_count : Integer;
                         VARS output_buffer : String;
                         VARS return_status : Integer); External;
Procedure IEEE_SEND_DATA (VARS output_count : Integer;
                          VARS output_buffer : String;
                          VARS return_status : Integer); External;
Procedure IEEE_SEND_LISTEN (VARS device_selector,
                            return_status : Integer); External;
Procedure IEEE_MULTIPLE_SEND_LISTEN (VARS device_count : Integer;
                                     VARS multiple_device_selector :
                                          device array;
                                     VARS return_status : Integer);
                                          External;
Procedure IEEE_SEND_MLA (VARS return_status : Integer); External;
Procedure IEEE_SEND_MTA (VARS return_status : Integer); External;
Procedure IEEE_SEND_SCG (VARS device_selector, secondary address,
                         return_status : Integer); External;
Procedure IEEE_MULTIPLE_SEND_SCG (VARS device_count : Integer;
                                  VARS multiple_device_selector,
                                       multiple_secondary_address :
                                       device array
                                  VARS return_status : Integer);
                                       External;
Procedure IEEE_SEND_TALK (VARS device_selector,
                          return_status, : Integer); External;
Procedure IEEE_SEND_UNL (VARS return_status : Integer); External;
Procedure IEEE_SEND_UNT (VARS return_status : Integer); External;
```

```
Procedure IEEE_SPOLL (VARS device_selector, poll_byte,
                                  return_status : Integer); External;
Procedure IEEE_STATUS (VARS service_request : Integer); External;
Procedure IEEE_TIMEOUT (VARS function, milliseconds,
                                  return_status : Integer); External;
Procedure IEEE_TRIGGER (VARS device_selector,
                                  return_status : Integer); External;
Procedure IEEE_MULTIPLE_TRIGGER (VARS device_count : Integer;
                                        VARS multiple_device_selector :
                                              device_array;
                                        VARS return_status : Integer);
                                              External;
```

If you are using several IEEE routines, you can include EXTERNAL.PAS in your program by using the $INCLUDE metacommand instead of typing the declarations listed. (The EXTERNAL.PAS file is on your IEEE 488 diskette.) If you do this, calls to the IEEE interface can be made by typing the name of the procedure you want to access, followed by the parameters:

```
IEEE_procedure_name(parameter_1,...Parameter_N);
```

Build your program and compile it. When you link your program, proceed normally until the linker asks if additional libraries will be used. Respond by entering IEEEPAS. The IEEEPAS library contains the high-level routines and an interface from these routines to the BIOS routines.

SET-UP AND CHECK-OUT OF THE IEEE 488 INTERFACE 5.5

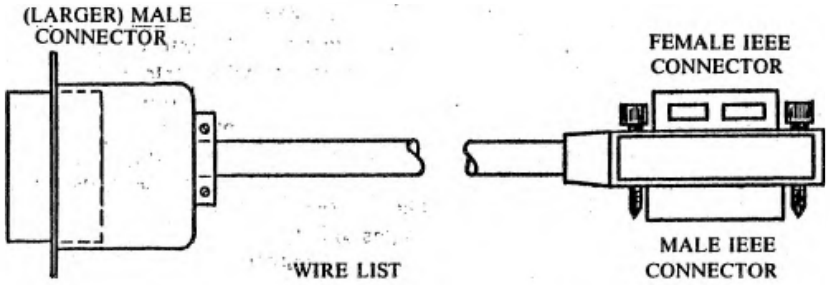
First, gather the devices you want to put on the IEEE 488 communications bus. (Follow manufacturer instructions on the set-up and installation of the devices.) Specify a bus address for each device, ensuring that each address is unique. If any device must be at address zero (the one to which your computer defaults), be sure your application uses the CONTROLLER ADDRESS function to give the computer a different (and unique) device address. (You must do this each time the software is loaded, or after a RESET function is done.) Determine which devices are to use which data lines during a parallel poll and set those values (either on the device itself, or by using software commands).

Next, attach the devices using the IEEE cables (see Exhibit 5b). Attach the large male connector to the parallel port in the rear of the computer. The smaller male connector is to be attached to a device on the bus.

Additional devices can be attached by connecting Standard IEEE Cables (included with IEEE-compatible devices or available from the vendor). Many devices can be connected to the bus (up to a maximum total cable length of 20 meters and a maximum of 15 devices), in any configuration (daisy chain, linear, or star). The final result, though, is always a continuous bus.

Because the principal cable is included with this product, one Standard IEEE Cable remains after all devices are interconnected.

Exhibit 5b: IEEE Bus Cables



WIRE LIST

CENTRONIX		IEEE	
1	DAV	6	TWISTED PAIR
19	GND	18	
2	DI01	1	
3	DI02	2	
4	DI03	3	
5	DI04	4	
6	DI05	13	
7	DI06	14	
8	DI07	15	
9	DI08	16	
10	NRFD	7	TWISTED PAIR
28	GND	19	
11	SRQ	10	TWISTED PAIR
29	GND	22	
13	NDAC	8	TWISTED PAIR
33	GND	20	
15	EOI	5	
17	SHIELD	12	
34	REN	17	
35	ATN	11	TWISTED PAIR
16	GND	23	
36	IFC	9	TWISTED PAIR
27	GND	21	
20	GND	24	

CHECK-OUT EXERCISES 5.6

Now your IEEE bus and devices are ready to be tested. Here are two trial application programs to help you get familiar with the interface.

The first example shows how to receive information from the plotter. To begin, set the variables needed by the IEEE 488 interface.

```
10 'This example shows how to receive information from the plotter.
20 'First we will do a serial poll on the plotter to determine
30 'its status and then we will ask it to identify itself.
40 '
50 '...IEEE necessary variables
60 '
70    PLOTTER% = 5                    'Address of the 7470a plotter
80    OUTPUT% = 5                     'Number associated with output routine
90    SERIAL.POLL% = 10               'Number associated with serial poll
100   ENTER% = 7                      'Number associated with enter
110   INFO% = 0                       'Tells IEEE 488 to output data
115   PLOTTER.STATUS% = 0             'Status returned from a serial poll
120   RETSTAT% = 0                    'Status returned from IEEE 488 routines
130   CNT% = 0                        'Will be assigned to length of
140                                   'messages to output
150   INPUT.COUNT% = 7                'Length of plotter ID
160 DIM IDENTIFICATION% (7)           'Storage for the ID
170 '
180 '...Plotter messages
190 '
200   INITIALIZE.PLOTTER$ = "IN;"          'Puts plotter in power on state
210   OUTPUT.ID$ = "OI;"                   'Puts plotter in power on ID
220 '
230 '...Main program
240 '
250   DEF SEG = 0                          'Set up to get high level address
260   LOWOFF = PEEK(884)                   'Low byte of offset address
270   HIOFF = PEEK(885)                    'High byte of offset address
280   LOWSEG = PEEK(886)                   'Low byte of segment address
290   HISEG = PEEK(887)                    'High byte of segment address
300   SEG = (256*HISEG) + LOWSEG           'Full segment address
310   IEEE = (256*HIOFF) + LOWOFF          'Full offset address
320   DEF SEG = SEG                        'Define segment for CALL
330 '
340 '...Enter data
350 '
360                                        'Initialize plotter
370   CNT% = LEN(INITIALIZE.PLOTTER$)
380   CALL IEEE(PLOTTER%, CNT%, INITIALIZE.PLOTTER$, INFO%, RETSTAT%, OUTPUT%)
390                                        'Retrieve the s poll integer
400   RETSTAT% = 0
410   CALL IEEE(PLOTTER%, PLOTTER.STATUS%, RETSTAT%, SERIAL.POLL%)
```

```
420                                             'Display the plotter status
430   PRINT 'The current plotter status is ";PLOTTER.STATUS%
440                                             'Command the plotter to send ID
450   RETSTAT% = 0
460   CNT% = LEN(OUTPUT.ID$)
470   CALL IEEE(PLOTTER%, CNT%, OUTPUT.ID$, INFO%, RETSTAT%, OUTPUT%)
480                                             'Retrieve the plotter ID
490   RETSTAT% = 0
500   CALL IEEE(PLOTTER%, INPUT.COUNT%, IDENTIFICATION%, RETSTAT%, ENTER%)
510                                             'Convert to ASCII
520   ID$ = ""
530   FOR I = 0 TO INPUT.COUNT% - 1
540         ID$ = ID$ + CHR$(IDENTIFICATION%(I))
550   NEXT I
560                                             'Display on screen
570   PRINT "You are talking to a "; ID$; "plotter."
580   END


10 'This example draws two triangles back to back using
20 'the 7470A plotter
30 '
40 '....IEEE necessary variables
50 '
60    PLOTTER% = 5                              'Address of the 7470a plotter
70    OUTPUT = 5                                'Number associated with output routine
80    INFO% = 0                                 'Tells IEEE 488 to output data
90    RETSTAT% = 0                              'Status returned from IEEE routines
100   CNT% = 0                                  'Will be assigned to length of
110                                             'messages to output
120 '....Plotter messages
130 '
140   SELECT.PEN.1$ = "SP1;"                    'Select pen from left stall
150   INITIALIZE.PLOTTER$ = "IN;"               'Puts plotter in power on state
160   POSITION.PEN$ = "PA2000,1500;"            'Plots Absolute at given x,y
170                                             'position
180   PEN.DOWN$ = "PD;"                         'Lowers the pen
190                                             'Plots Absolute to given x,y
200   DRAW.TRIANGLE.1$ = "PA0,1500,2000,3500,2000,1500;"
210   PEN.UP$ = "PU;"                           'Lifts the pen
220   POSITION.PEN.2$ = "PA2500,1500;"          'Moves to beginning of triangle 2
230                                             'Draws the 2nd triangle at given locs
240   DRAW.TRIANGLE.2$ = "PA4500,1500,2500,3500,2500,1500;"
250   STORE.PEN$ = "SP;"                        'SP without parameters will
260                                             'store the pen
270 '
280 '....Main program
290 '
300   DEF SEG = 0                               'Set up to get high level address
310   LOWOFF = PEEK(884)                        'Low byte offset address
320   HIOFF = PEEK(885)                         'High byte offset address
330   LOWSEG = PEEK(886)                        'Low byte segment address
340   HISEG = PEEK(887)                         'High byte segment address
350   SEG = (256*HISEG) + LOWSEG                'Full segment address
360   IEEE = (256*HIOFF) + LOWOFF               'Full offset address
370   DEF SEG = SEG
380 '
390 '....Perform plots
400 '
410                                             'Initialize plotter
420   CNT% = LEN(INITIALIZE.PLOTTER$);
```

```
430  CALL IEEE(PLOTTER%, CNT%, INITIALIZE.PLOTTER$, INFO%, RETSTAT%, OUTPUT%)
440                                          'Select a new pen to use
450  RETSTAT% = 0                            'Re-initialize return status
460  CNT% = LEN(SELECT.PEN.1$)
470  CALL IEEE(PLOTTER%, CNT%, SELECT.PEN.1$, INFO%, RETSTAT%, OUTPUT%)
480                                          'Put pen at lower left corner
490  RETSTAT% = 0
500  CNT% = LEN(POSITION.PEN$)
510  CALL IEEE(PLOTTER%, CNT%, POSITION.PEN$, INFO%, RETSTAT%, OUTPUT%)
520                                          'Put the pen down
530  RETSTAT% = 0
540  CNT% = LEN(PEN.DOWN$)
550  CALL IEEE(PLOTTER%, CNT%, PEN.DOWN$, INFO%, RETSTAT%, OUTPUT%)
560                                          'Draw the first triangle
570  RETSTAT% = 0
580  CNT% = LEN(DRAW.TRIANGLE.1$)
590  CALL IEEE(PLOTTER%, CNT%, DRAW.TRIANGLE.1$, INFO%, RETSTAT%, OUTPUT%)
600                                          'Raise the pen to move to new triangle
610  RETSTAT% = 0
620  CNT% = LEN(PEN.UP$)
630  CALL IEEE(PLOTTER%, CNT%, PEN.UP$, INFO%, RETSTAT%, OUTPUT%)
640                                          'Move to new triangle lower left corner
650  RETSTAT% = 0.
660  CNT% = LEN(POSITION.PEN.2$)
670  CALL IEEE(PLOTTER%, CNT%, POSITION.PEN.2$, INFO%, RETSTAT%, OUTPUT%)
680                                          'Draw the 2nd triangle
690  RETSTAT% = 0
700  CNT% = LEN(DRAW.TRIANGLE.2$)
710  CALL IEEE(PLOTTER%, CNT%, DRAW.TRIANGLE.2$, INFO%, RETSTAT%, OUTPUT%)
720                                          'Store the pen
730  RETSTAT% = 0
740  CNT% = LEN(STORE.PEN$)
750  CALL IEEE(PLOTTER%, CNT%, STORE.PEN$, INFO%,RETSTAT%, OUTPUT%)
760  END
```