CHAPTER 6

COMPUTER INTERFACE OPERATION

6.0 GENERAL

This chapter provides operational instructions for the computer interface for the Lake Shore Model 332 Temperature Controller. Either of the two computer interfaces provided with the Model 332 permit remote operation. The first is the IEEE-488 Interface described in Paragraph 6.1. The second is the Serial Interface described in Paragraph 6.2. The two interfaces share a common set of commands detailed in Paragraph 6.3. Only one of the interfaces can be used at a time.

NOTE: The remote interface of the Model 332 can be set to emulate a Lake Shore Model 330 Temperature Controller. Refer to Paragraph 4.20 to select 330 Emulation Mode. Refer to your Model 330 User's Manual for command syntax. The following Model 330 commands are not supported in 330 Emulation Mode: CUID?, CURV, CURV?, ECUR, KCUR, and SCAL.

6.1 IEEE-488 INTERFACE

The IEEE-488 Interface is an instrumentation bus with hardware and programming standards that simplify instrument interfacing. The Model 332 IEEE-488 Interface complies with the IEEE-488.2-1987 standard and incorporates its functional, electrical, and mechanical specifications unless otherwise specified in this manual.

All instruments on the interface bus perform one or more of the interface functions of TALKER, LISTENER, or BUS CONTROLLER. A TALKER transmits data onto the bus to other devices. A LISTENER receives data from other devices through the bus. The BUS CONTROLLER designates to the devices on the bus which function to perform. The Model 332 performs the functions of TALKER and LISTENER but cannot be a BUS CONTROLLER. The BUS CONTROLLER is the digital computer which tells the Model 332 which functions to perform.

Below are Model 332 IEEE-488 interface capabilities:

- **SH1:** Source handshake capability.
- **RL1:** Complete remote/local capability.
- **DC1:** Full device clear capability.
- **DT0:** No device trigger capability.
- **C0:** No system controller capability.
- **T5:** Basic TALKER, serial poll capability, talk only, unaddressed to talk if addressed to listen.
- **L4:** Basic LISTENER, unaddressed to listen if addressed to talk.
- **SR1:** Service request capability.
- **AH1:** Acceptor handshake capability.
- **PP0:** No parallel poll capability.
- **E1:** Open collector electronics.

NOTE: The Model 332 IEEE-488 Interface requires that repeat addressing be enabled on the bus controller.

Instruments are connected to the IEEE-488 bus by a 24-conductor connector cable as specified by the standard. Refer to Paragraph 8.4.2. Cables can be purchased from Lake Shore or other electronic suppliers. A connector extender (Model 4005) is required to use the IEEE-488 Interface and the RELAY and ANALOG OUTPUT Terminal Block at the same time.

Cable lengths are limited to 2 meters for each device and 20 meters for the entire bus. The Model 332 can drive a bus with up to 10 loads. If more instruments or cable length is required, a bus expander must be used.

6.1.1 Changing IEEE-488 Interface Parameters

Two interface parameters, address and terminators, must be set from the front panel before communication with the instrument can be established. Other interface parameters can be set with device specific commands using the interface (Paragraph 6.3).

Press the **Interface** key. The first screen is for selecting the Serial Interface Baud Rate, and can be skipped by pressing the **Enter** key. The Address screen is then displayed as follows.

```
Select—With—°®
IEEE—Address——12
```

Press the s or t keys to increment or decrement the IEEE Address to the desired number. Valid addresses are 1 thru 30. Default is 12. Press **Enter** to accept new number or **Escape** to retain the existing number. Pressing **Enter** displays the Terminators screen.

```
Select—With—°®
IEEE Term——Cr—Lf
```

Press the s or t keys to cycle through the following Terminator choices: CR/LF, LF/CR, LF, and EOI. The default is Cr Lf. To accept changes or the currently displayed setting, push **Enter**. To cancel changes, push **Escape**.

6.1.2 IEEE-488 Command Structure

The Model 332 supports several command types. These commands are divided into three groups.

1. **Bus Control** – Refer to Paragraph 6.1.2.1.
   a. Universal
      (1) Uniline
      (2) Multiline
   b. Addressed Bus Control
2. **Common** – Refer to Paragraph 6.1.2.2.
3. **Device Specific** – Refer to Paragraph 6.1.2.3.
4. **Message Strings** – Refer to Paragraph 6.1.2.4.

6.1.2.1 Bus Control Commands

A Universal Command addresses all devices on the bus. Universal Commands include Uniline and Multiline Commands. A Uniline Command (Message) asserts only a single signal line. The Model 332 recognizes two of these messages from the BUS CONTROLLER: **Remote (REN)** and **Interface Clear (IFC)**. The Model 332 sends one Uniline Command: **Service Request (SRQ)**.

REN (Remote) – Puts the Model 332 into remote mode.

IFC (Interface Clear) – Stops current operation on the bus.

SRQ (Service Request) – Tells the bus controller that the Model 332 needs interface service.

A Multiline Command asserts a group of signal lines. All devices equipped to implement such commands do so simultaneously upon command transmission. These commands transmit with the Attention (ATN) line asserted low. The Model 332 recognizes two Multiline commands:

LLO (Local Lockout) – Prevents the use of instrument front panel controls.

DCL (Device Clear) – Clears Model 332 interface activity and puts it into a bus idle state.

Bus Control Commands (Continued)

Finally, Addressed Bus Control Commands are Multiline commands that must include the Model 332 listen address before the instrument responds. Only the addressed device responds to these commands. The Model 332 recognizes three of the Addressed Bus Control Commands:

SDC (Selective Device Clear) – The SDC command performs essentially the same function as the DCL command except that only the addressed device responds.

GTL (Go To Local) – The GTL command is used to remove instruments from the remote mode. With some instruments, GTL also unlocks front panel controls if they were previously locked out with the LLO command.

SPE (Serial Poll Enable) and **SPD (Serial Poll Disable)** – Serial polling accesses the Service Request Status Byte Register. This status register contains important operational information from the unit requesting service. The SPD command ends the polling sequence.

6.1.2.2 Common Commands

Common Commands are addressed commands which create commonalty between instruments on the bus. All instruments that comply with the IEEE-488 1987 standard share these commands and their format. Common commands all begin with an asterisk. They generally relate to "bus" and "instrument" status and identification. Common query commands end with a question mark (?). Model 332 common commands are detailed in Paragraph 6.3 and summarized in Table 6-8.

6.1.2.3 Device Specific Commands

Device specific commands are addressed commands. The Model 332 supports a variety of device specific commands to program instruments remotely from a digital computer and to transfer measurements to the computer. Most device specific commands perform functions also performed from the front panel. Model 332 device specific commands are detailed in Paragraph 6.3 and summarized in Table 6-8.

6.1.2.4 Message Strings

A message string is a group of characters assembled to perform an interface function. There are three types of message strings commands, queries and responses. The computer issues command and query strings through user programs, the instrument issues responses. Two or more command strings can be chained together in one communication but they must be separated by a semi-colon (;). Only one query is permitted per communication but it can be chained to the end of a command. The total communication string must not exceed 64 characters in length.

A command string is issued by the computer and instructs the instrument to perform a function or change a parameter setting. When a command is issued, the computer is acting as 'talker' and the instrument as 'listener'. The format is:

```
<command mnemonic><space><parameter data><terminators>.
```

Command mnemonics and parameter data necessary for each one is described in Paragraph 6.3. Terminators must be sent with every message string.

A query string is issued by the computer and instructs the instrument which response to send. Queries are issued similar to commands with the computer acting as 'talker' and the instrument as 'listener'. The query format is:

```
<query mnemonic><?><space><parameter data><terminators>.
```

Query mnemonics are often the same as commands with the addition of a question mark. Parameter data is often unnecessary when sending queries. Query mnemonics and parameter data if necessary is described in Paragraph 6.3. Terminators must be sent with every message string. Issuing a query does not initiate a response from the instrument.

A response string is sent by the instrument only when it is addressed as a 'talker' and the computer becomes the 'listener'. The instrument will respond only to the last query it receives. The response can be a reading value, status report or the present value of a parameter. Response data formats are listed along with the associated queries in Paragraph 6.3.

6.1.3 Status Registers

There are two status registers: the Status Byte Register described in Paragraph 6.1.3.1, and the Standard Event Status Register in Paragraph 6.1.3.2.

6.1.3.1 Status Byte Register and Service Request Enable Register

The Status Byte Register contains six bits of information about the operation of the Model 332.

STATUS BYTE REGISTER FORMAT

Bit –	7	6	5	4	3	2	1	0
Weighting –	128	64	32	16	8	4	2	1
Bit Name –	Ramp Done	SRQ	ESB	Error	Alarm	Not Used	Not Used	New A&B

If Service Request is enabled, any of these bits being set will cause the Model 332 to pull the SRQ management line low to signal the BUS CONTROLLER. These bits are reset to zero upon a serial poll of the Status Byte Register. These reports can be inhibited by turning their corresponding bits in the Service Request Enable Register to off.

The Service Request Enable Register allows the user to inhibit or enable any of the status reports in the Status Byte Register. The QSRE command is used to set the bits. If a bit in the Service Request Enable Register is set (1), then that function is enabled. Refer to theQSRE command discussion.

Ramp Done, Bit (7) – This bit is set when the ramp is completed.

Service Request (SRQ) Bit (6) – Determines whether the Model 332 is to report via the SRQ line. If bits 0, 3, 4, 5 and/or 7 are set, then the corresponding bit in the Status Byte Register will be set. The Model 332 will produce a service request only if bit 6 of the Service Request Enable Register is set. If disabled, the Status Byte Register can still be read by the BUS CONTROLLER by means of a serial poll (SPE) to examine the status reports, but the BUS CONTROLLER will not be interrupted by the Service Request. The QSTB common command will read the Status Byte Register but will not clear the bits.

Standard Event Status (ESB), Bit (5) – When bit 5 is set, it indicates if one of the bits from the Standard Event Status Register has been set. (Refer to Paragraph 6.1.3.2.)

Error, Bit (4) – This bit is set when there is an instrument error not related to the bus.

Alarm, Bit (3) – This bit is set when there is an alarm condition.

New A&B, Bit (0) – This bit is set when new data is available from the normal inputs.

6.1.3.2 Standard Event Status Register and Standard Event Status Enable Register

The Standard Event Status Register reports IEEE bus status of the Model 332.

STANDARD EVENT STATUS REGISTER FORMAT

Bit –	7	6	5	4	3	2	1	0
Weighting –	128	64	32	16	8	4	2	1
Bit Name –	PON	Not Used	CME	EXE	DDE	QYE	Not Used	OPC

Bits 2 and 6 are not used. The bus controller will only be interrupted with the reports of this register if the bits have been enabled in the Standard Event Status Enable Register and if bit 5 of the Service Request Enable Register has been set.

The Standard Event Status Enable Register allows the user to enable any of the Standard Event Status Register reports. The Standard Event Status Enable command (QESE) sets the Standard Event Status Enable Register bits. If a bit of this register is set, then that function is enabled. To set a bit, send the command QESE with the bit weighting for each bit you want to be set added together. See the QESE command discussion for further details.

Standard Event Status Register and Standard Event Status Enable Register (Continued)

The Standard Event Status Enable Query, QESE?, reads the Standard Event Status Enable Register. QESR? reads the Standard Event Status Register. Once this register has been read, all of the bits are reset to zero.

Power On (PON), Bit (7) – This bit is set to indicate an instrument off-on transition.

Command Error (CME), Bit (5) – This bit is set if a command error has been detected since the last reading. This means that the instrument could not interpret the command due to a syntax error, an unrecognized header, unrecognized terminators, or an unsupported command.

Execution Error (EXE), Bit (4) – This bit is set if the EXE bit is set, an execution error has been detected. This occurs when the instrument is instructed to do something not within its capabilities.

Device Dependent Error (DDE), Bit (3) – This bit is set if a device dependent error has been detected if the DDE bit is set. The actual device dependent error can be found by executing the various device dependent queries.

Query Error (QYE), Bit (2) – This bit indicates a query error. It occurs rarely and involves loss of data because the output queue is full.

Operation Complete (OPC), Bit (0) – This bit is generated in response to the QOPC common command. It indicates when the Model 332 has completed all selected pending operations. It is not related to the QOPC? command, which is a separate interface feature.

6.1.4 IEEE Interface Example Programs

Two BASIC programs are included to illustrate the IEEE-488 communication functions of the instrument. The first program was written in Visual Basic. Refer to Paragraph 6.1.4.1 for instructions on how to setup the program. The Visual Basic code is provided in Table 6-2. The second program is written in Quick Basic. Refer to Paragraph 6.1.4.3 for instructions on how to setup the program. The Quick Basic code is provided in Table 6-3. Finally, a description of operation common to both programs is provided in Paragraph 6.1.4.5. While the hardware and software required to produce and implement these programs not included with the instrument, the concepts illustrated apply to almost any application where these tools are available.

6.1.4.1 IEEE-488 Interface Board Installation for Visual Basic Program

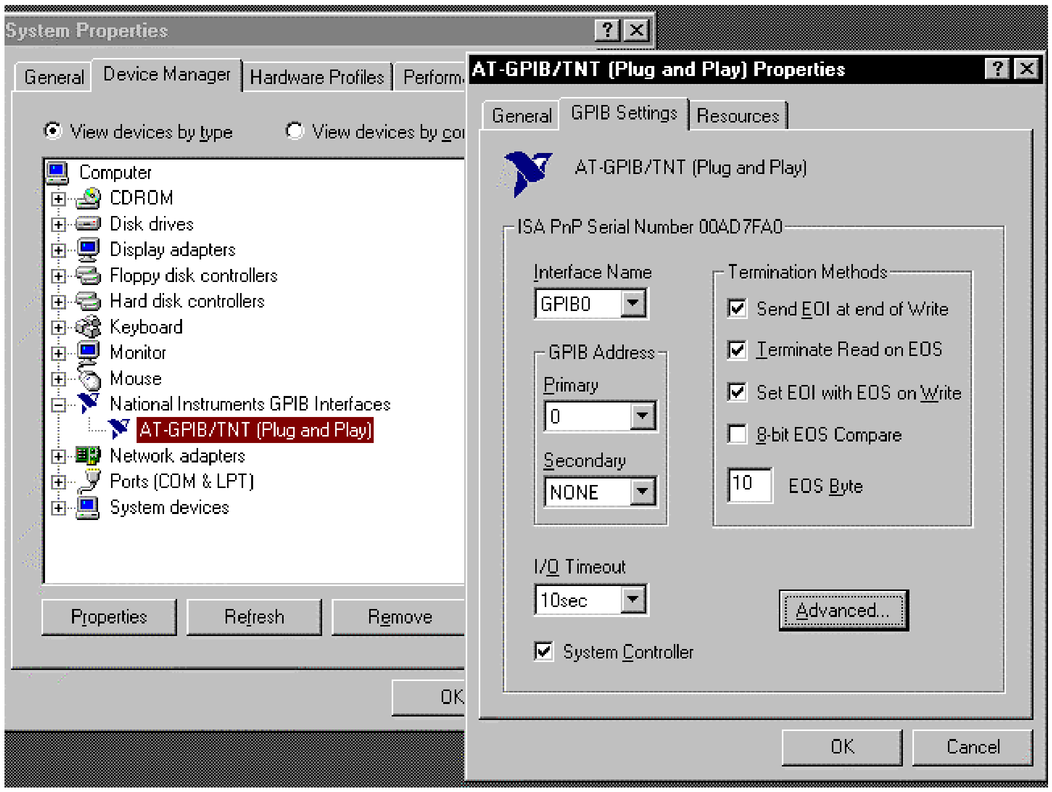
This procedure works for Plug and Play GPIB Hardware and Software for Windows 98/95. This example uses the AT-GPIB/TNT GPIB card.

1. Install the GPIB Plug and Play Software and Hardware using National Instruments instructions.
2. Verify that the following files have been installed to the Windows System folder:
   a. gpib-32.dll
   b. gpib.dll
   c. gpib32ft.dll

   Files b and c will support 16-bit Windows GPIB applications if any are being used.
3. Locate the following files and make note of their location. These files will be used during the development process of a Visual Basic program.
   a. Niglobal.bas
   b. Vbib-32.bas

NOTE: If the files in Steps 2 and 3 are not installed on your computer, they may be copied from your National Instruments setup disks or they may be downloaded from www.natinst.com.

4. Configure the GPIB by selecting the System icon in the Windows 98/95 Control Panel located under Settings on the Start Menu. Configure the GPIB Settings as shown in Figure 6-1. Configure the DEV12 Device Template as shown in Figure 6-2. Be sure to check the Readdress box.



AT-GPIB/TNT (Plug and Play) Properties

General | GPIB Settings | Resources

AT-GPIB/TNT (Plug and Play)

ISA PnP Serial Number 00AD7FA0

Interface Name: GPIB0

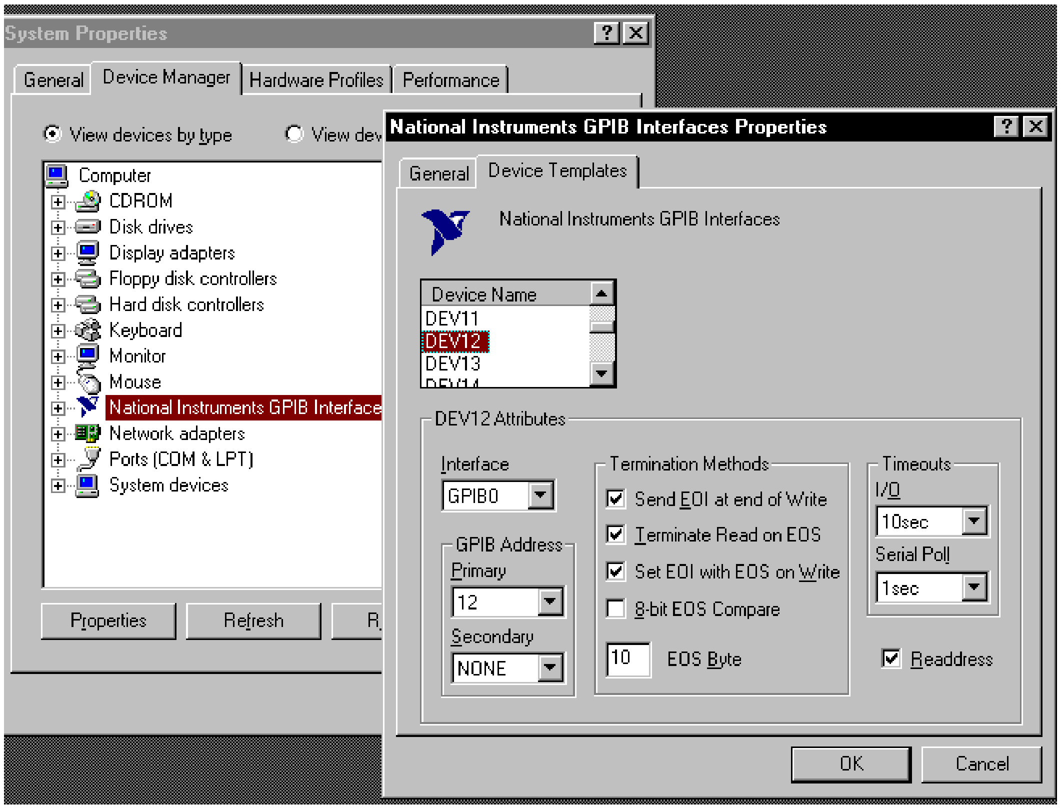
GPIB Address
Primary: 0
Secondary: NONE

Termination Methods
[✓] Send EOI at end of Write
[✓] Terminate Read on EOS
[✓] Set EOI with EOS on Write
[ ] 8-bit EOS Compare
10 EOS Byte

I/O Timeout: 10sec

[✓] System Controller

Figure 6-1. GPIB Setting Configuration



National Instruments GPIB Interfaces Properties

General | Device Templates

National Instruments GPIB Interfaces

Device Name
DEV11
DEV12
DEV13
DEV14

DEV12 Attributes

Interface: GPIB0

GPIB Address
Primary: 12
Secondary: NONE

Termination Methods
[✓] Send EOI at end of Write
[✓] Terminate Read on EOS
[✓] Set EOI with EOS on Write
[ ] 8-bit EOS Compare
10 EOS Byte

Timeouts
I/O: 10sec
Serial Poll: 1sec

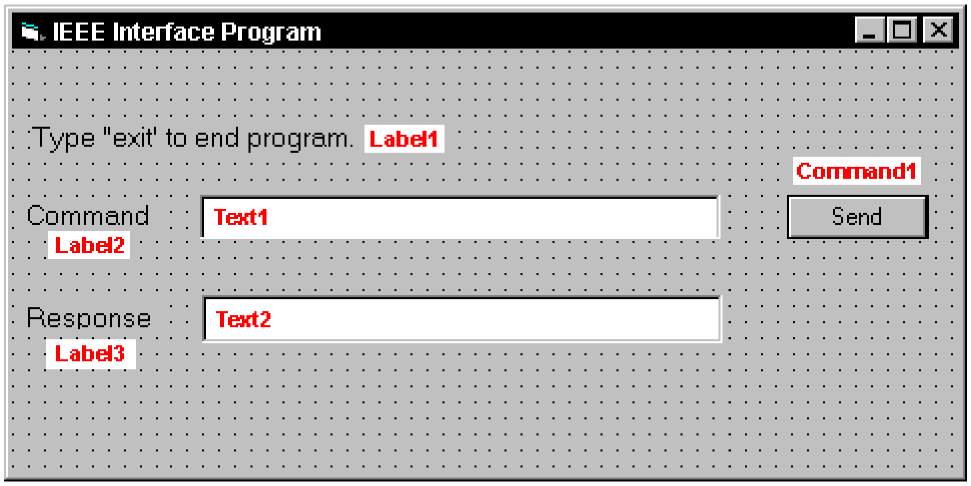
[✓] Readdress

Figure 6-2. DEV 12 Device Template Configuration

6.1.4.2 Visual Basic IEEE-488 Interface Program Setup

This IEEE-488 interface program works with Visual Basic 6.0 (VB6) on an IBM PC (or compatible) with a Pentium-class processor. A Pentium 90 or higher is recommended, running Windows 95 or better. It assumes your IEEE-488 (GPIB) card is installed and operating correctly (refer to Paragraph 6.1.4.1). Use the following procedure to develop the IEEE-488 Interface Program in Visual Basic.

1. Start VB6.
2. Choose Standard EXE and select Open.
3. Resize form window to desired size.
4. On the Project Menu, select Add Module, select the Existing tab, then navigate to the location on your computer to add the following files: Niglobal.bas and Vbib-32.bas.
5. Add controls to form:
   a. Add three Label controls to the form.
   b. Add two TextBox controls to the form.
   c. Add one CommandButton control to the form.
6. On the View Menu, select Properties Window.
7. In the Properties window, use the dropdown list to select between the different controls of the current project.

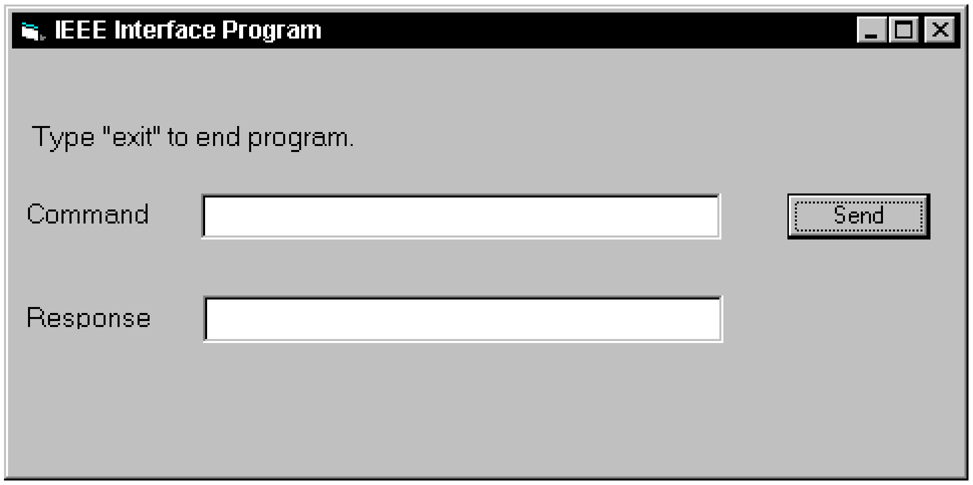


10. Set the properties of the controls as defined in Table 6-1.
11. Save the program.

Table 6-1. IEEE-488 Interface Program Control Properties

Current Name	Property	New Value
Label1	Name Caption	lblExitProgram Type "exit" to end program.
Label2	Name Caption	lblCommand Command
Label3	Name Caption	lblResponse Response
Text1	Name Text	txtCommand <blank>
Text2	Name Text	txtResponse <blank>
Command1	Name Caption Default	cmdSend Send True
Form1	Name Caption	frmIEEE IEEE Interface Program

12. Add code (provided in Table 6-2).
    a. In the Code Editor window, under the Object dropdown list, select (General). Add the statement: Public gSend as Boolean
    b. Double Click on cmdSend. Add code segment under Private Sub cmdSend_Click( ) as shown in Table 6-2.
    c. In the Code Editor window, under the Object dropdown list, select Form. Make sure the Procedure dropdown list is set at Load. The Code window should have written the segment of code: Private Sub Form_Load( ). Add the code to this subroutine as shown in Table 6-2.
13. Save the program.
14. Run the program. The program should resemble the following.



15. Type in a command or query in the Command box as described in Paragraph 6.1.4.5.
16. Press Enter or select the Send button with the mouse to send command.
17. Type Exit and press Enter to quit.

Table 6-2. Visual Basic IEEE-488 Interface Program

```
Public gSend As Boolean                                     'Global used for Send button state
Private Sub cmdSend_Click()                                 'Routine to handle Send button press
    gSend = True                                            'Set Flag to True
End Sub
Private Sub Form_Load()                                     'Main code section
    Dim strReturn As String                                 'Used to return response
    Dim term As String                                      'Terminators
    Dim strCommand As String                                'Data string sent to instrument
    Dim intDevice As Integer                                'Device number used with IEEE

    frmIEEE.Show                                            'Show main window
    term = Chr(13) & Chr(10)                                'Terminators are <CR><LF>
    strReturn = ""                                          'Clear return string

    Call ibdev(0, 12, 0, T10s, 1, &H140A, intDevice)        'Initialize the IEEE device
    Call ibconfig(intDevice, ibcREADDR,1)                   'Setup Repeat Addressing
    Do
        Do                                                  'Wait loop
        DoEvents                                            'Give up processor to other events
        Loop Until gSend = True                             'Loop until Send button pressed
        gSend = False                                       'Set Flag as False

        strCommand = frmIEEE.txtCommand.Text                'Get Command
        strReturn = ""                                      'Clear response display

        strCommand = UCase(strCommand)                      'Set all characters to upper case
        If strCommand = "EXIT" Then                         'Get out on EXIT
            End
        End If

        Call ibwrt(intDevice, strCommand & term)            'Send command to instrument
        If (ibsta And EERR) Then                            'Check for IEEE errors
            'do error handling if needed                    'Handle errors here
        End If

        If InStr(strCommand, "?") <> 0 Then                 'Check to see if query
            strReturn = Space(100)                          'Build empty return buffer
            Call ibrd(intDevice, strReturn)                 'Read back response
            If (ibsta And EERR) Then                        'Check for IEEE errors
                'do error handling if needed                'Handle errors here
            End If

            If strReturn <> "" Then                         'Check if empty string
                strReturn = RTrim(strReturn)                'Remove extra spaces and Terminators
                Do While Right(strReturn, 1) = Chr(10) Or Right(strReturn, 1) = Chr(13)
                    strReturn = Left(strReturn, Len(strReturn) - 1)
                Loop
            Else
                strReturn = "No Response"                   'Send No Response
            End If

            frmIEEE.txtResponse.Text = strReturn            'Put response in text on main form
        End If
    Loop
End Sub
```

6.1.4.3 IEEE-488 Interface Board Installation for Quick Basic Program

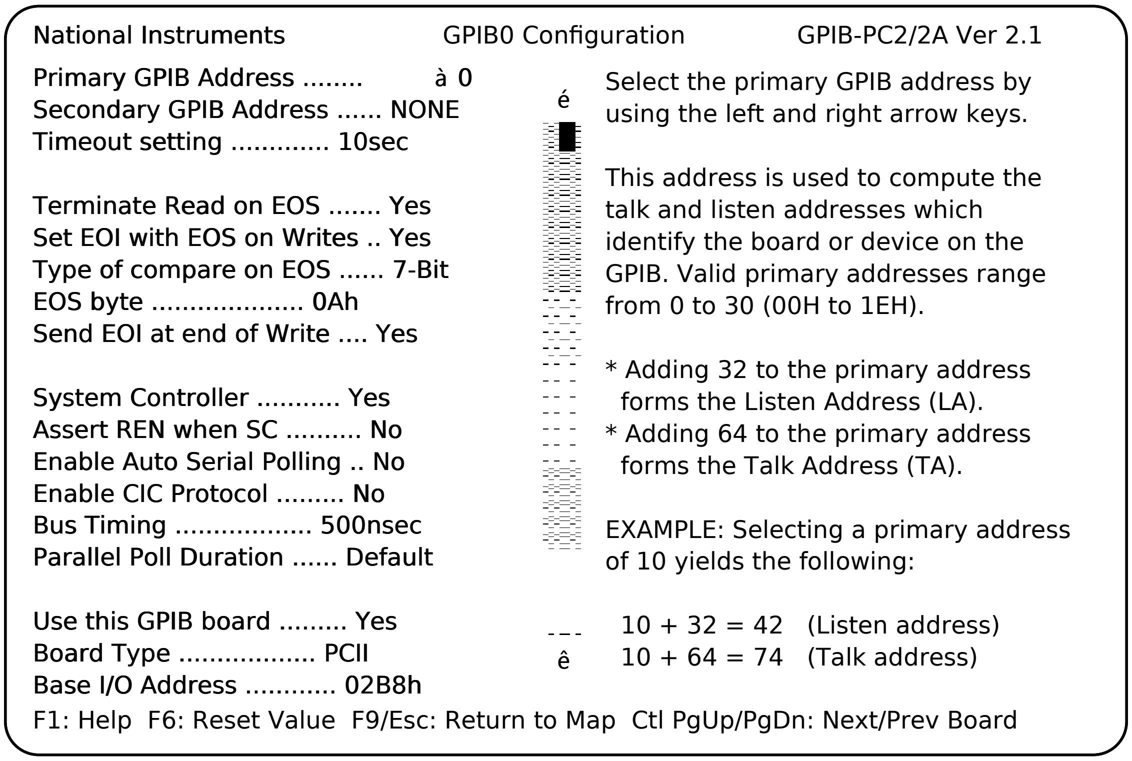
This procedure works on an IBM PC (or compatible) running DOS or in a DOS window. This example uses the National Instruments GPIB-PCII/IIA card.

1. Install GPIB-PCII/IIA card using National Instruments instructions.
2. Install NI-488.2 software (for DOS). Version 2.1.1 was used for the example.
3. Verify that config.sys contains the command: device = \gpib-pc\gpib.com
4. Reboot the computer.
5. Run IBTEST to test software configuration. Do not install the instrument before running IBTEST.
6. Run IBCONF to configure the GPIB – PCII/IIA board and dev 12. Set the EOS byte to 0AH and Enable Repeat Addressing to Yes. See Figure 6-3. IBCONF modifies gpib.com.
7. Connect the instrument to the interface board and power up the instrument. Verify the address is 12 and terminators are CR LF.

6.1.4.4 Quick Basic Program

The IEEE-488 interface program in Table 6-3 works with QuickBasic 4.0/4.5 or Qbasic on an IBM PC (or compatible) running DOS or in a DOS window. It assumes your IEEE-488 (GPIB) card is installed and operating correctly (refer to Paragraph 6.1.4.3). Use the following procedure to develop the Serial Interface Program in Quick Basic.

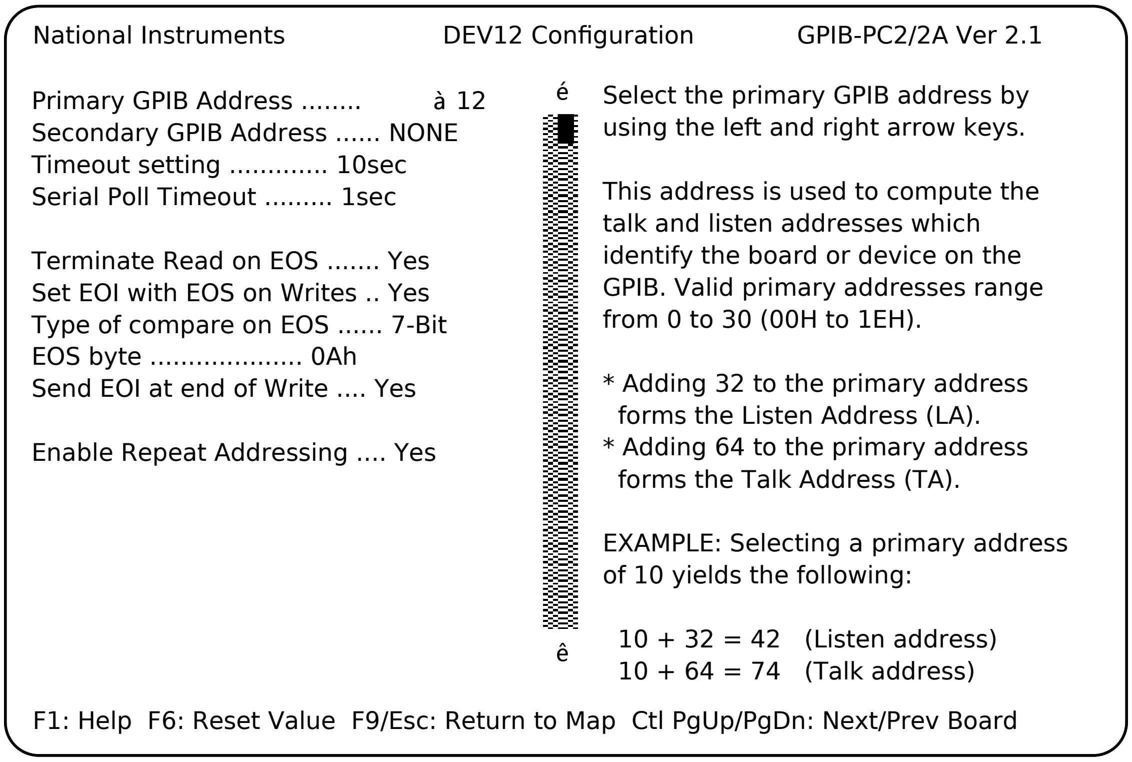
1. Copy c:\gpib-pc\Qbasic\qbib.obj to the QuickBasic directory (QB4).
2. Change to the QuickBasic directory and type: link /q qbib.obj,,,bqlb4x.lib; where x = 0 for QB4.0 and 5 for QB4.5 This one-time only command produces the library file qbib.qlb. The procedure is found in the National Instruments QuickBasic readme file Readme.qb.
3. Start QuickBasic. Type: qb /l qbib.qlb. Start QuickBasic in this way each time the IEEE interface is used to link in the library file.
4. Create the IEEE example interface program in QuickBasic. Enter the program exactly as presented in Table 6-3. Name the file "ieeeexam.bas" and save.
5. Run the program.
6. Type a command query as described in Paragraph 6.1.4.5.
7. Type "EXIT" to quit the program.



```
National Instruments            GPIB0 Configuration          GPIB-PC2/2A Ver 2.1

Primary GPIB Address ........      à 0      é   Select the primary GPIB address by
Secondary GPIB Address ...... NONE              using the left and right arrow keys.
Timeout setting ............. 10sec
                                                This address is used to compute the
Terminate Read on EOS ....... Yes               talk and listen addresses which
Set EOI with EOS on Writes .. Yes               identify the board or device on the
Type of compare on EOS ...... 7-Bit             GPIB. Valid primary addresses range
EOS byte .................... 0Ah               from 0 to 30 (00H to 1EH).
Send EOI at end of Write .... Yes
                                                * Adding 32 to the primary address
System Controller ........... Yes                 forms the Listen Address (LA).
Assert REN when SC .......... No                * Adding 64 to the primary address
Enable Auto Serial Polling .. No                  forms the Talk Address (TA).
Enable CIC Protocol ......... No
Bus Timing .................. 500nsec          EXAMPLE: Selecting a primary address
Parallel Poll Duration ...... Default           of 10 yields the following:

Use this GPIB board ......... Yes                 10 + 32 = 42   (Listen address)
Board Type .................. PCII          ê     10 + 64 = 74   (Talk address)
Base I/O Address ............ 02B8h
F1: Help  F6: Reset Value  F9/Esc: Return to Map  Ctl PgUp/PgDn: Next/Prev Board
```



```
National Instruments            DEV12 Configuration          GPIB-PC2/2A Ver 2.1

                                            é   Select the primary GPIB address by
Primary GPIB Address ........      à 12         using the left and right arrow keys.
Secondary GPIB Address ...... NONE
Timeout setting ............. 10sec             This address is used to compute the
Serial Poll Timeout ......... 1sec              talk and listen addresses which
                                                identify the board or device on the
Terminate Read on EOS ....... Yes               GPIB. Valid primary addresses range
Set EOI with EOS on Writes .. Yes               from 0 to 30 (00H to 1EH).
Type of compare on EOS ...... 7-Bit
EOS byte .................... 0Ah               * Adding 32 to the primary address
Send EOI at end of Write .... Yes                 forms the Listen Address (LA).
                                                * Adding 64 to the primary address
Enable Repeat Addressing .... Yes                 forms the Talk Address (TA).

                                                EXAMPLE: Selecting a primary address
                                                of 10 yields the following:

                                                  10 + 32 = 42   (Listen address)
                                            ê     10 + 64 = 74   (Talk address)

F1: Help  F6: Reset Value  F9/Esc: Return to Map  Ctl PgUp/PgDn: Next/Prev Board
```

C-331-6-3.eps

Figure 6-3. Typical National Instruments GPIB Configuration from IBCONF.EXE

Table 6-3. Quick Basic IEEE-488 Interface Program

```
'       IEEEEXAM.BAS    EXAMPLE PROGRAM FOR IEEE-488 INTERFACE
'
'       This program works with QuickBasic 4.0/4.5 on an IBM PC or compatible.
'
'       The example requires a properly configured National Instruments GPIB-PC2 card. The REM
'       $INCLUDE statement is necessary along with a correct path to the file QBDECL.BAS.
'       CONFIG.SYS must call GPIB.COM created by IBCONF.EXE prior to running Basic. There must
'       be QBIB.QBL library in the QuickBasic Directory and QuickBasic must start with a link
'       to it. All instrument settings are assumed to be defaults: Address 12, Terminators
'       <CR> <LF> and EOI active.
'
'       To use, type an instrument command or query at the prompt. The computer transmits to
'       the instrument and displays any response. If no query is sent, the instrument responds
'       to the last query received. Type "EXIT" to exit the program.
'
        REM $INCLUDE: 'c:\gpib-pc\qbasic\qbdecl.bas'        'Link to IEEE calls
        CLS                                                 'Clear screen
        PRINT "IEEE-488 COMMUNICATION PROGRAM"
        PRINT

        CALL IBFIND("dev12", DEV12%)                        'Open communication at address 12
        TERM$ = CHR$(13) + CHR$(10)                         'Terminators are <CR><LF>

LOOP2:  IN$ = SPACE$(2000)                                  'Clear for return string

        LINE INPUT "ENTER COMMAND (or EXIT):"; CMD$         'Get command from keyboard
        CMD$ = UCASE$(CMD$)                                 'Change input to upper case
           IF CMD$ = "EXIT" THEN END                        'Get out on Exit
        CMD$ = CMD$ + TERM$

        CALL IBWRT(DEV12%, CMD$)                            'Send command to instrument

        CALL IBRD(DEV12%, IN$)                              'Get data back each time

        ENDTEST = INSTR(IN$, CHR$(13))                      'Test for returned string
           IF ENDTEST > 0 THEN                              'String is present if <CR> is seen
              IN$ = MID$(IN$, 1, ENDTEST - 1)               'Strip off terminators
              PRINT "RESPONSE:", IN$                        'Print return string
           ELSE
              PRINT "NO RESPONSE"                           'No string present if timeout
           END IF
        GOTO LOOP2                                          'Get next command
```

6.1.4.5 Program Operation

Once either example program is running, try the following commands and observe the response of the instrument. Input from the user is shown in **bold** and terminators are added by the program. The word [term] indicates the required terminators included with the response.

ENTER COMMAND? ***IDN?**	Identification query. Instrument will return a string identifying itself.
RESPONSE: LSCI,MODEL332,123456,020301[term]	
ENTER COMMAND? **KRDG?**	Temperature reading in kelvin query. Instrument will return a string with the present temperature reading.
RESPONSE: +273.15[term]	
ENTER COMMAND? **RANGE 0**	Heater range command. Instrument will turn off the heater. No response will be sent.
ENTER COMMAND? **RANGE?**	Heater range query. Instrument will return a string with the present heater range setting.
RESPONSE: 0[term]	
ENTER COMMAND? **RANGE 1;RANGE?**	Heater range command followed by a query. Instrument will change to heater Low setting then return a string with the present setting.
RESPONSE: 1[term]	

The following are additional notes on using either IEEE-488 Interface program.

- If you enter a correctly spelled query without a "**?**," nothing will be returned. Incorrectly spelled commands and queries are ignored. Commands and queries and should have a space separating the command and associated parameters.
- Leading zeros and zeros following a decimal point are not needed in a command string, but are sent in response to a query. A leading "+" is not required but a leading "–"*is* required.

6.1.5 Troubleshooting

New Installation

1. Check instrument address.
2. Always send terminators.
3. Send entire message string at one time including terminators.
4. Send only one simple command at a time until communication is established.
5. Be sure to spell commands correctly and use proper syntax.
6. Attempt both 'Talk' and 'Listen' functions. If one works but not the other, the hardware connection is working, so look at syntax, terminators, and command format.
7. If only one message is received after resetting the interface, check the "repeat addressing" setting. It should be enabled.

Old Installation No Longer Working

1. Power instrument off then on again to see if it is a soft failure.
2. Power computer off then on again to see if the IEEE card is locked up.
3. Verify that the address has not been changed on the instrument during a memory reset.
4. Check all cable connections.

Intermittent Lockups

1. Check cable connections and length.
2. Increase delay between all commands to 50 ms to make sure instrument is not being over loaded.

6.2 SERIAL INTERFACE OVERVIEW

The serial interface used in the Model 332 is commonly referred to as an RS-232C interface. RS-232C is a standard of the Electronics Industries Association (EIA) that describes one of the most common interfaces between computers and electronic equipment. The RS-232C standard is quite flexible and allows many different configurations. However, any two devices claiming RS-232C compatibility cannot necessarily be plugged together without interface setup. The remainder of this paragraph briefly describes the key features of a serial interface that are supported by the instrument. A customer supplied computer with similarly configured interface port is required to enable communication.

6.2.1 Physical Connection

The Model 332 has a 9 pin D-Subminiature plug on the rear panel for serial communication. The original RS-232C standard specifies 25 pins but both 9- and 25-pin connectors are commonly used in the computer industry. Many third party cables exist for connecting the instrument to computers with either 9- or 25-pin connectors. Paragraph 8.4.1 gives the most common pin assignments for 9- and 25-pin connectors. Please note that not all pins or functions are supported by the Model 332.

The instrument serial connector is the plug half of a mating pair and must be matched with a socket on the cable. If a cable has the correct wiring configuration but also has a plug end, a "gender changer" can be used to mate two plug ends together.

The letters DTE near the interface connector stand for Data Terminal Equipment and indicate the pin connection of the directional pins such as transmit data (TD) and receive data (RD). Equipment with Data Communications Equipment (DCE) wiring can be connected to the instrument with a straight through cable. As an example, Pin 3 of the DTE connector holds the transmit line and Pin 3 of the DCE connector holds the receive line so the functions complement.

It is likely both pieces of equipment are wired in the DTE configuration. In this case Pin 3 on one DTE connector (used for transmit) must be wired to Pin 2 on the other (used for receive). Cables that swap the complementing lines are called null modem cables and must be used between two DTE wired devices. Null modem adapters are also available for use with straight through cables. Paragraph 8.4.1 illustrates suggested cables that can be used between the instrument and common computers.

The instrument uses drivers to generate the transmission voltage levels required by the RS-232C standard. These voltages are considered safe under normal operating conditions because of their relatively low voltage and current limits. The drivers are designed to work with cables up to 50 feet in length.

6.2.2 Hardware Support

The Model 332 interface hardware supports the following features. Asynchronous timing is used for the individual bit data within a character. This timing requires start and stop bits as part of each character so the transmitter and receiver can resynchronized between each character. Half duplex transmission allows the instrument to be either a transmitter or a receiver of data but not at the same time. Communication speeds of 300, 1200 or 9600 Baud are supported. The Baud rate is the only interface parameter that can be changed by the user.

Hardware handshaking is not supported by the instrument. Handshaking is often used to guarantee that data message strings do not collide and that no data is transmitted before the receiver is ready. In this instrument appropriate software timing substitutes for hardware handshaking. User programs must take full responsibility for flow control and timing as described in Paragraph 6.2.5.

6.2.3 Character Format

A character is the smallest piece of information that can be transmitted by the interface. Each character is 10 bits long and contains data bits, bits for character timing and an error detection bit. The instrument uses 7 bits for data in the ASCII format. One start bit and one stop bit are necessary to synchronize consecutive characters. Parity is a method of error detection. One parity bit configured for odd parity is included in each character.

ASCII letter and number characters are used most often as character data. Punctuation characters are used as delimiters to separate different commands or pieces of data. Two special ASCII characters, carriage return (CR 0DH) and line feed (LF 0AH), are used to indicate the end of a message string.

Table 6-4. Serial Interface Specifications

Connector Type:	9-pin D-style connector plug
Connector Wiring:	DTE
Voltage Levels:	EIA RS-232C Specified
Transmission Distance:	50 feet maximum
Timing Format:	Asynchronous
Transmission Mode:	Half Duplex
Baud Rate:	300, 1200, 9600
Handshake:	Software timing
Character Bits:	1 Start, 7 Data, 1 Parity, 1 Stop
Parity:	Odd
Terminators:	CR(0DH) LF(0AH)
Command Rate:	20 commands per second maximum

6.2.4 Message Strings

A message string is a group of characters assembled to perform an interface function. There are three types of message strings commands, queries and responses. The computer issues command and query strings through user programs, the instrument issues responses. Two or more command strings can be chained together in one communication but they must be separated by a semi-colon (;). Only one query is permitted per communication but it can be chained to the end of a command. The total communication string must not exceed 64 characters in length.

A command string is issued by the computer and instructs the instrument to perform a function or change a parameter setting. The format is:

```
<command mnemonic><space><parameter data><terminators>.
```

Command mnemonics and parameter data necessary for each one is described in Paragraph 6.3. Terminators must be sent with every message string.

A query string is issued by the computer and instructs the instrument to send a response. The query format is:

```
<query mnemonic><?><space><parameter data><terminators>.
```

Query mnemonics are often the same as commands with the addition of a question mark. Parameter data is often unnecessary when sending queries. Query mnemonics and parameter data if necessary is described in Paragraph 6.3. Terminators must be sent with every message string. The computer should expect a response very soon after a query is sent.

A response string is the instruments response or answer to a query string. The instrument will respond only to the last query it receives. The response can be a reading value, status report or the present value of a parameter. Response data formats are listed along with the associated queries in Paragraph 6.3. The response is sent as soon as possible after the instrument receives the query. Typically it takes 10 ms for the instrument to begin the response. Some responses take longer.

6.2.5 Message Flow Control

It is important to remember that the user program is in charge of the serial communication at all times. The instrument can not initiate communication, determine which device should be transmitting at a given time or guarantee timing between messages. All of this is the responsibility of the user program.

When issuing commands only the user program should:

- Properly format and transmit the command including terminators as one string.
- Guarantee that no other communication is started for 50 ms after the last character is transmitted.
- Not initiate communication more than 20 times per second.

When issuing queries or queries and commands together the user program should:

- Properly format and transmit the query including terminators as one string.
- Prepare to receive a response immediately.
- Receive the entire response from the instrument including the terminators.
- Guarantee that no other communication is started during the response or for 50 ms after it completes.
- Not initiate communication more than 20 times per second.

Failure to follow these simple rules will result in inability to establish communication with the instrument or intermittent failures in communication.

6.2.6 Changing Baud Rate

To use the Serial Interface, you must first set the Baud rate. Press **Interface** key to display the following screen.

```
Select—With—°®
Baud——9600
```

Press the s or t key to cycle through the choices of 300, 1200, or 9600 Baud. Press the **Enter** key to accept the new number.

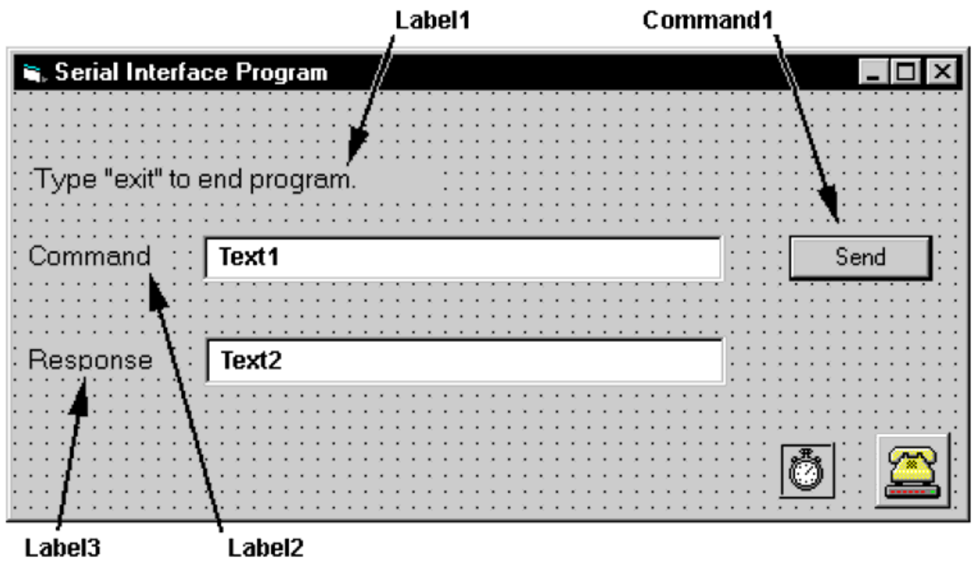
6.2.7 Serial Interface Example Programs

Two BASIC programs are included to illustrate the serial communication functions of the instrument. The first program was written in Visual Basic. Refer to Paragraph 6.2.7.1 for instructions on how to setup the program. The Visual Basic code is provided in Table 6-6. The second program was written in Quick Basic. Refer to Paragraph 6.2.7.2 for instructions on how to setup the program. The Quick Basic code is provided in Table 6-7. Finally, a description of operation common to both programs is provided in Paragraph 6.2.7.3. While the hardware and software required to produce and implement these programs not included with the instrument, the concepts illustrated apply to almost any application where these tools are available.

6.2.7.1 Visual Basic Serial Interface Program Setup

The serial interface program works with Visual Basic 6.0 (VB6) on an IBM PC (or compatible) with a Pentium-class processor. A Pentium 90 or higher is recommended, running Windows 95 or better, with a serial interface. It uses the COM1 communications port at 9600 Baud. Use the following procedure to develop the Serial Interface Program in Visual Basic.

1. Start VB6.
2. Choose Standard EXE and select Open.
3. Resize form window to desired size.
4. On the Project Menu, click Components to bring up a list of additional controls available in VB6.
5. Scroll through the controls and select Microsoft Comm Control 6.0. Select OK. In the toolbar at the left of the screen, the Comm Control will have appeared as a telephone icon.
6. Select the Comm control and add it to the form.
7. Add controls to form:
   a. Add three Label controls to the form.
   b. Add two TextBox controls to the form.
   c. Add one CommandButton control to the form.
   d. Add one Timer control to the form.
8. On the View Menu, select Properties Window.
9. In the Properties window, use the dropdown list to select between the different controls of the current project.

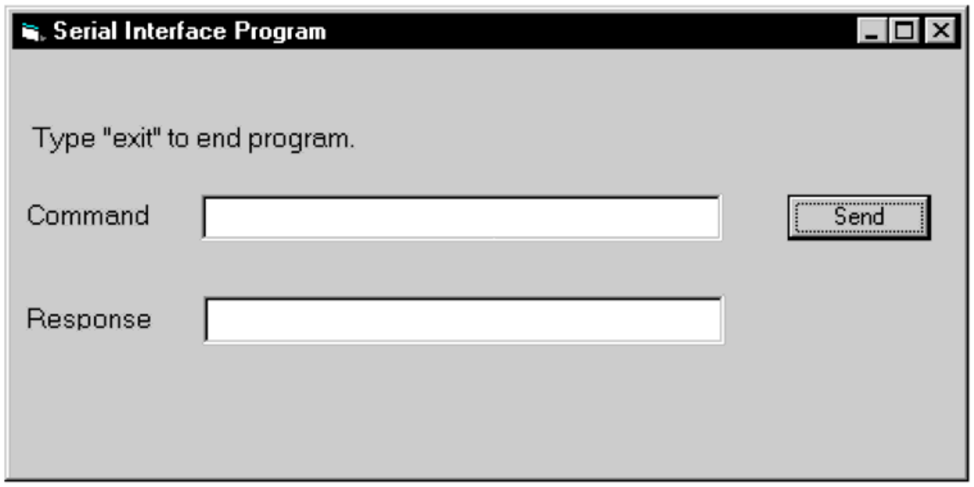


10. Set the properties of the controls as defined in Table 6-5.
11. Save the program.

Table 6-5. Serial Interface Program Control Properties

Current Name	Property	New Value
Label1	Name Caption	lblExitProgram Type "exit" to end program.
Label2	Name Caption	lblCommand Command
Label3	Name Caption	lblResponse Response
Text1	Name Text	txtCommand <blank>
Text2	Name Text	txtResponse <blank>
Command1	Name Caption Default	cmdSend Send True
Form1	Name Caption	frmSerial Serial Interface Program
Timer1	Enabled Interval	False 10

12. Add code (provided in Table 6-6).
    a. In the Code Editor window, under the Object dropdown list, select (General). Add the statement: Public gSend as Boolean
    b. Double Click on cmdSend. Add code segment under Private Sub cmdSend_Click( ) as shown in Table 6-6.
    c. In the Code Editor window, under the Object dropdown list, select Form. Make sure the Procedure dropdown list is set at Load. The Code window should have written the segment of code: Private Sub Form_Load( ). Add the code to this subroutine as shown in Table 6-6.
    d. Double Click on the Timer control. Add code segment under Private Sub Timer1_Timer() as shown in Table 6-6.
    e. Make adjustments to code if different Com port settings are being used.
13. Save the program.
14. Run the program. The program should resemble the following.



15. Type in a command or query in the Command box as described in Paragraph 6.2.7.3.
16. Press Enter or select the Send button with the mouse to send command.
17. Type Exit and press Enter to quit.

Table 6-6. Visual Basic Serial Interface Program

```
Public gSend As Boolean                                 'Global used for Send button state
Private Sub cmdSend_Click()                             'Routine to handle Send button press
    gSend = True                                        'Set Flag to True
End Sub
Private Sub Form_Load()                                 'Main code section
    Dim strReturn As String                             'Used to return response
    Dim strHold As String                               'Temporary character space
    Dim Term As String                                  'Terminators
    Dim ZeroCount As Integer                            'Counter used for Timing out
    Dim strCommand As String                            'Data string sent to instrument

    frmSerial.Show                                      'Show main window
    Term = Chr(13) & Chr(10)                            'Terminators are <CR><LF>
    ZeroCount = 0                                       'Initialize counter
    strReturn = ""                                      'Clear return string
    strHold = ""                                        'Clear holding string
    If frmSerial.MSComm1.PortOpen = True Then           'Close serial port to change settings
        frmSerial.MSComm1.PortOpen = False
    End If
    frmSerial.MSComm1.CommPort = 1                      'Example of Comm 1
    frmSerial.MSComm1.Settings = "9600,o,7,1"           'Example of 9600 Baud,Parity,Data,Stop
    frmSerial.MSComm1.InputLen = 1                      'Read one character at a time
    frmSerial.MSComm1.PortOpen = True                   'Open port

  Do
    Do                                                  'Wait loop
    DoEvents                                            'Give up processor to other events
    Loop Until gSend = True                             'Loop until Send button pressed
    gSend = False                                       'Set Flag as false

    strCommand = frmSerial.txtCommand.Text              'Get Command
    strReturn = ""                                      'Clear response display

    strCommand = UCase(strCommand)                      'Set all characters to upper case
    If strCommand = "EXIT" Then                         'Get out on EXIT
        End
    End If

    frmSerial.MSComm1.Output = strCommand & Term        'Send command to instrument
    If InStr(strCommand, "?") <> 0 Then                 'Check to see if query
        While (ZeroCount < 20) And (strHold <> Chr$(10))  'Wait for response
            If frmSerial.MSComm1.InBufferCount = 0 Then   'Add 1 to timeout if no character
                frmSerial.Timer1.Enabled = True
                Do
                DoEvents                                'Wait for 10 millisecond timer
                Loop Until frmSerial.Timer1.Enabled = False
                ZeroCount = ZeroCount + 1               'Timeout at 2 seconds
            Else
                ZeroCount = 0                           'Reset timeout for each character
                strHold = frmSerial.MSComm1.Input       'Read in one character
                strReturn = strReturn + strHold         'Add next character to string
            End If
        Wend                                            'Get characters until terminators

        If strReturn <> "" Then                         'Check if string empty
            strReturn = Mid(strReturn, 1, InStr(strReturn, Term) - 1)  'Strip terminators
        Else
            strReturn = "No Response"                   'Send No Response
        End If
        frmSerial.txtResponse.Text = strReturn          'Put response in textbox on main form
        strHold = ""                                    'Reset holding string
        ZeroCount = 0                                   'Reset timeout counter
    End If
  Loop
End Sub
Private Sub Timer1_Timer()                              'Routine to handle Timer interrupt
    frmSerial.Timer1.Enabled = False                    'Turn off timer
End Sub
```

6.2.7.2 Quick Basic Serial Interface Program Setup

The serial interface program listed in Table 6-7 works with QuickBasic 4.0/4.5 or Qbasic on an IBM PC (or compatible) running DOS or in a DOS window with a serial interface. It uses the COM1 communication port at 9600 Baud. Use the following procedure to develop the Serial Interface Program in Quick Basic.

1. Start the Basic program.
2. Enter the program exactly as presented in Table 6-7.
3. Adjust the Com port and Baud rate in the program as necessary.
4. Lengthen the "TIMEOUT" count if necessary.
5. Save the program.
6. Run the program.
7. Type a command query as described in Paragraph 6.2.7.3.
8. Type "EXIT" to quit the program.

Table 6-7. Quick Basic Serial Interface Program

```
       CLS                                                'Clear screen
       PRINT " SERIAL COMMUNICATION PROGRAM"
       PRINT
       TIMEOUT = 2000                                     'Read timeout (may need more)
       BAUD$ = "9600"
       TERM$ = CHR$(13) + CHR$(10)                        'Terminators are <CR><LF>
       OPEN "COM1:" + BAUD$ + ",O,7,1,RS" FOR RANDOM AS #1 LEN = 256

LOOP1: LINE INPUT "ENTER COMMAND (or EXIT):"; CMD$        'Get command from keyboard
       CMD$ = UCASE$(CMD$)                                'Change input to upper case
          IF CMD$ = "EXIT" THEN CLOSE #1: END             'Get out on Exit
       CMD$ = CMD$ + TERM$
       PRINT #1, CMD$;                                    'Send command to instrument

       IF INSTR(CMD$, "?") <> 0 THEN                      'Test for query
          RS$ = ""                                        'If query, read response
          N = 0                                           'Clr return string and count

          WHILE (N < TIMEOUT) AND (INSTR(RS$, TERM$) = 0)    'Wait for response
             IN$ = INPUT$(LOC(1), #1)                     'Get one character at a time
             IF IN$ = "" THEN N = N + 1 ELSE N = 0        'Add 1 to timeout if no chr
             RS$ = RS$ + IN$                              'Add next chr to string
          WEND                                            'Get chrs until terminators

          IF RS$ <> "" THEN                               'See if return string is empty
             RS$ = MID$(RS$, 1, (INSTR(RS$, TERM$) – 1))  'Strip off terminators
             PRINT "RESPONSE:"; RS$                       'Print response to query
          ELSE
             PRINT "NO RESPONSE"                          'No response to query
          END IF
       END IF                                             'Get next command
       GOTO LOOP1
```

6.2.7.3 Program Operation

Once either example program is running, try the following commands and observe the response of the instrument. Input from the user is shown in **bold** and terminators are added by the program. The word [term] indicates the required terminators included with the response.

`ENTER COMMAND?` **`*IDN?`**	Identification query. Instrument will return a string identifying itself.
`RESPONSE: LSCI,MODEL332,123456,020301[term]`	
`ENTER COMMAND?` **`KRDG?`**	Temperature reading in kelvin query. Instrument will return a string with the present temperature reading.
`RESPONSE: +273.15[term]`	
`ENTER COMMAND?` **`RANGE 0`**	Heater range command. Instrument will turn off the heater. No response will be sent.
`ENTER COMMAND?` **`RANGE?`**	Heater range query. Instrument will return a string with the present heater range setting.
`RESPONSE: 0[term]`	
`ENTER COMMAND?` **`RANGE 1;RANGE?`**	Heater range command followed by a query. Instrument will change to heater Low setting then return a string with the present setting.
`RESPONSE: 1[term]`	

The following are additional notes on using either Serial Interface program.

- If you enter a correctly spelled query without a "**?**," nothing will be returned. Incorrectly spelled commands and queries are ignored. Commands and queries and should have a space separating the command and associated parameters.
- Leading zeros and zeros following a decimal point are not needed in a command string, but they will be sent in response to a query. A leading "**+**" is not required but a leading "**–**" *is* required.

6.2.8 Troubleshooting

New Installation

1. Check instrument Baud rate.
2. Make sure transmit (TD) signal line from the instrument is routed to receive (RD) on the computer and vice versa. (Use a null modem adapter if not).
3. Always send terminators.
4. Send entire message string at one time including terminators. (Many terminal emulation programs do not.)
5. Send only one simple command at a time until communication is established.
6. Be sure to spell commands correctly and use proper syntax.

Old Installation No Longer Working

1. Power instrument off then on again to see if it is a soft failure.
2. Power computer off then on again to see if communication port is locked up.
3. Verify that Baud rate has not been changed on the instrument during a memory reset.
4. Check all cable connections.

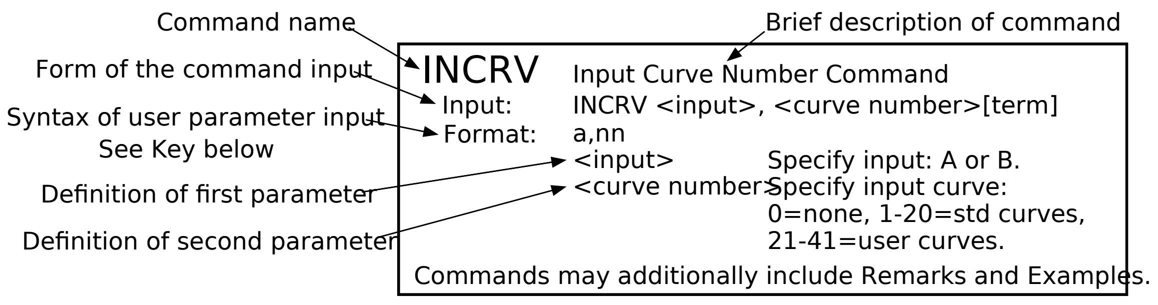
Intermittent Lockups

1. Check cable connections and length.
2. Increase delay between all commands to 100 ms to ensure instrument is not being over loaded.

6.3 COMMAND SUMMARY

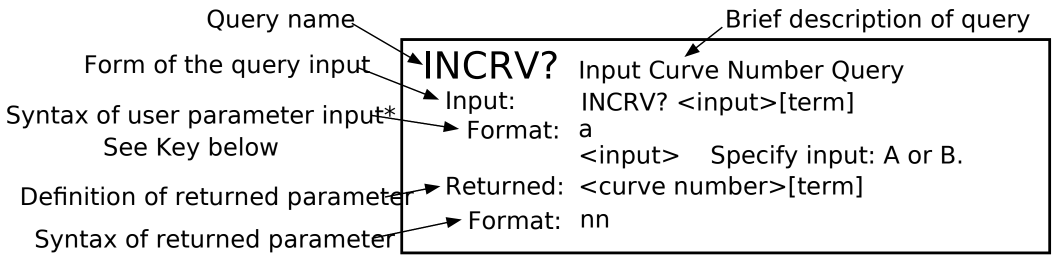
This paragraph provides a listing of the IEEE-488 and Serial Interface Commands. A summary of all the commands is provided in Table 6-8. All the commands are detailed in Paragraph 6.3.1, which is presented in alphabetical order.

Sample Command Format


Command name
Brief description of command
Form of the command input
Syntax of user parameter input
See Key below
Definition of first parameter
Definition of second parameter

```
INCRV     Input Curve Number Command
 Input:   INCRV <input>, <curve number>[term]
 Format:  a,nn
          <input>         Specify input: A or B.
          <curve number>  Specify input curve:
                          0=none, 1-20=std curves,
                          21-41=user curves.
Commands may additionally include Remarks and Examples.
```

Sample Query Format


Query name
Brief description of query
Form of the query input
Syntax of user parameter input*
See Key below
Definition of returned parameter
Syntax of returned parameter

```
INCRV?      Input Curve Number Query
  Input:    INCRV? <input>[term]
   Format:  a
            <input>    Specify input: A or B.
  Returned: <curve number>[term]
   Format:  nn
```

The initial Format definition is omitted for queries that do not require parameter input.

Key

Q	Begins common interface command.
?	Required to identify queries.
aa…	String of alpha numeric characters.
nn…	String of number characters that may include a decimal point.
[term]	Terminator characters.
<…>	Indicated a parameter field, many are command specific.
<state>	Parameter field with only On/Off or Enable/Disable states.
<value>	Floating point values have varying resolution depending on the type of command or query issued.

Table 6-8. Command Summary

Command	***Function***	***Page***	***Command***	***Function***	***Page***
QCLS	Clear Interface Cmd	24	HTR?	Heater Output Query	32
QESE	Event Status Enable Cmd	24	HTRST?	Heater Status Query	32
QESE?	Event Status Enable Query	24	IEEE	IEEE Interface Parameter Cmd	32
QESR?	Event Status Register Query	24	IEEE?	IEEE Interface Parameter Query	32
QIDN?	Identification Query	24	INCRV	Input Curve Number Cmd	32
QOPC	Operation Complete Cmd	25	INCRV?	Input Curve Number Query	32
QOPC?	Operation Complete Query	25	INTYPE	Input Type Parameter Cmd	33
QRST	Reset Instrument Cmd	25	INTYPE?	Input Type Parameter Query	33
QSRE	Service Request Enable Cmd	25	KEYST?	Keypad Status Query	33
QSRE?	Service Request Enable Query	25	KRDG?	Kelvin Reading Query	33
QSTB?	Status Byte Query	25	LDAT?	Linear Equation Data Query	34
QTST?	Self-Test Query	26	LINEAR	Input Linear Equation Cmd	34
QWAI	Wait-To-Continue Cmd	26	LINEAR?	Input Linear Equation Query	34
ALARM	Input Alarm Parameter Cmd	26	LOCK	Front Panel Keyboard Lock Cmd	34
ALARM?	Input Alarm Parameter Query	26	LOCK?	Front Panel Keyboard Lock Query	35
ALARMST?	Input Alarm Status Query	26	MDAT?	Min/Max Data Query	35
ALMRST	Reset Alarm Status Cmd	27	MNMX	Min/Max Input Function Cmd	35
ANALOG	Analog Output Parameter Cmd	27	MNMX?	Min/Max Input Function Query	35
ANALOG?	Analog Outputs Parameter Query	27	MNMXRST	Min/Max Function Reset Cmd	35
AOUT?	Analog Output Data Query	27	MODE	Set Local/Remote Mode	35
BAUD	RS-232 Baud Rate Cmd	27	MODE?	Query Local/Remote Mode	36
BAUD?	RS-232 Baud Rate Query	27	MOUT	Control Loop MHP Output Cmd	36
BEEP	System Beeper Cmd	28	MOUT?	Control Loop MHP Output Query	36
BEEP?	System Beeper Query	28	PID	Control Loop PID Values Cmd	36
BRIGT	Display Brightness Command	28	PID?	Control Loop PID Values Query	36
BRIGT?	Display Brightness Query	28	RAMP	Control Loop Ramp Cmd	37
CMODE	Control Loop Mode Cmd	28	RAMP?	Control Loop Ramp Query	37
CMODE?	Control Loop Mode Query	28	RAMPST?	Control Loop Ramp Status Query	37
CRDG?	Celsius Reading Query	28	RANGE	Heater Range Cmd	37
CRVDEL	Delete User Curve Cmd	29	RANGE?	Heater Range Query	37
CRVHDR	Curve Header Cmd	29	RDGST?	Input Status Query	38
CRVHDR?	Curve Header Query	29	RELAY	Relay Control Parameter Cmd	38
CRVPT	Curve Data Point Cmd	29	RELAY?	Relay Control Parameter Query	38
CRVPT?	Curve Data Point Query	30	RELAYST?	Relay Status Query	38
CSET	Control Loop Parameter Cmd	30	REV?	Input Firmware Revision Query	38
CSET?	Control Loop Parameter Query	30	SCAL	Generate SoftCal Curve Cmd	39
DFLT	Factory Defaults Cmd	30	SETP	Control Loop Setpoint Cmd	39
DISPFLD	Displayed Field Cmd	30	SETP?	Control Loop Setpoint Query	39
DISPFLD?	Displayed Field Query	31	SRDG?	Sensor Units Reading Query	39
EMUL	330 Emulation Mode Cmd	31	TEMP?	Room-Temp Comp. Temp. Query	40
EMUL?	330 Emulation Mode Query	31	TUNEST?	Control Loop 1 Tuning Query	40
FILTER	Input Filter Parameter Cmd	31	ZONE	Control Loop Zone Table Cmd	40
FILTER?	Input Filter Parameter Query	31	ZONE?	Control Loop Zone Table Query	40

6.3.1 Interface Commands (Alphabetical Listing)

QCLS Clear Interface Command

Input: `QCLS[term]`

Remarks: Clears the bits in the Status Byte Register and Standard Event Status Register and terminates all pending operations. Clears the interface, but *not* the controller. The related controller command is Q**RST**.

QESE Event Status Enable Register Command

Input: `QESE <bit weighting>[term]`

Format: nnn

Remarks: Each bit is assigned a bit weighting and represents the enable/disable mask of the corresponding event flag bit in the Standard Event Status Register. To enable an event flag bit, send the command Q**ESE** with the sum of the bit weighting for each desired bit. Refer to Paragraph 6.1.3.2 for a list of event flags.

Example: To enable event flags 0, 3, 4, and 7, send the command Q**ESE 143[term]**. 143 is the sum of the bit weighting for each bit.

Bit	Bit Weighting	Event Name
0	1	OPC
3	8	DDE
4	16	EXE
7	128	PON
	143	

QESE? Event Status Enable Register Query

Input: `QESE?[term]`

Returned: <bit weighting>[term]

Format: nnn Refer to Paragraph 6.1.3.2 for a list of event flags.

QESR? Standard Event Status Register Query

Input: `QESR?[term]`

Returned: <bit weighting>

Format: nnn

Remarks: The integer returned represents the sum of the bit weighting of the event flag bits in the Standard Event Status Register. Refer to Paragraph 6.1.3.2 for a list of event flags.

QIDN? Identification Query

Input: `QIDN?[term]`

Returned: <manufacturer>,<model>,<serial>,<date>[term]

Format: aaaa,aaaaaaaa,aaaaaa,mmddyy

<manufacture>	Manufacturer ID
<model>	Instrument model number
<serial>	Serial number
<date>	Instrument firmware revision date

Example: `LSCI,MODEL332,123456,020301`

Q**OPC** Operation Complete Command

Input: `QOPC[term]`

Remarks: Generates an Operation Complete event in the Event Status Register upon completion of all pending selected device operations. Send it as the last command in a command string.

Q**OPC?** Operation Complete Query

Input: `QOPC?[term]`

Returned: 1[term]

Remarks: Places a "1" in the controller output queue upon completion of all pending selected device operations. Send as the last command in a command string. *Not* the same as Q**OPC**.

Q**RST** Reset Instrument Command

Input: `QRST[term]`

Remarks: Sets controller parameters to power-up settings.

Q**SRE** Service Request Enable Register Command

Input: `QSRE <bit weighting>[term]`

Format: nnn

Remarks: Each bit has a bit weighting and represents the enable/disable mask of the corresponding status flag bit in the Status Byte Register. To enable a status flag bit, send the command Q**SRE** with the sum of the bit weighting for each desired bit. Refer to Paragraph 6.1.3.1 for a list of status flags.

Example: To enable status flags 0, 2, 4, and 6, send the command Q**SRE 89[term]**. 89 is the sum of the bit weighting for each bit.

Bit	Bit Weighting	Event Name
0	1	New A&B
3	8	Alarm
4	16	Error
6	64	SRQ
	89	

Q**SRE?** Service Request Enable Register Query

Input: `QSRE?[term]`

Returned: <bit weighting>[term]

Format: nnn Refer to Paragraph 6.1.3.1 for a list of status flags.

Q**STB?** Status Byte Query

Input: `QSTB?[term]`

Returned: <bit weighting>[term]

Format: nnn

Remarks: Acts like a serial poll, but does not reset the register to all zeros. The integer returned represents the sum of the bit weighting of the status flag bits that are set in the Status Byte Register. Refer to Paragraph 6.1.3.1 for a list of status flags.

QTST? Self-Test Query

Input: `QTST?[term]`

Returned: <status>[term]

Format: n

<status>	0 = no errors found, 1 = errors found

Remarks: The Model 332 reports status based on test done at power up.

QWAI Wait-to-Continue Command

Input: `QWAI[term]`

Remarks: This command is not supported in the Model 332.

ALARM Input Alarm Parameter Command

Input: `ALARM <input>, <off/on>, <source>, <high value>, <low value>, <deadband>, <latch enable>[term]`

Format: a,n,n, ±nnnnnn, ±nnnnnn, ±nnnnnn,n

<input>	Specifies which input to configure: A or B.
<off/on>	Determines whether the instrument checks the alarm for this input, where 0 = off and 1 = on.
<source>	Specifies input data to check. Valid entries: 1 = kelvin, 2 = Celsius, 3 = sensor units, 4 = linear data.
<high value>	Sets the value the source is checked against to activate the high alarm.
<low value>	Sets the value the source is checked against to activate low alarm.
<deadband>	Sets the value that the source must change outside of an alarm condition to deactivate an unlatched alarm.
<latch enable>	Specifies a latched alarm (remains active after alarm condition correction) where 0 = off (no latch) and 1 = on.

Remarks: Configures the alarm parameters for an input.

Example: **ALARM A,0[term] –** Turns off alarm checking for Input A.
ALARM B,1,1,270.0,0,0,1[term] – Turns on alarm checking for input B, activates high alarm if kelvin reading is over 270, and latches the alarm when kelvin reading falls below 270.

ALARM? Input Alarm Parameter Query

Input: `ALARM? <input>[term]`

Format: a

<input>	A or B

Returned: <off/on>, <source>, <high value>, <low value>, <deadband>, <latch enable> [term]

Format: n,n,±nnnnnn,±nnnnnn,±nnnnnn,n (Refer to command for description)

ALARMST? Input Alarm Status Query

Input: `ALARMST? <input>[term]`

Format: a

<input>	A or B

Returned: <high state>, <low state>[term]

Format: n,n

<high state>	0 = Off, 1 = On
<low state>	0 = Off, 1 = On

ALMRST Reset Alarm Status Command

Input: `ALMRST[term]`

Remarks: Clears both the high and low status of all alarms, including latching alarms.

ANALOG Analog Output Parameter Command

Input: `ANALOG <bipolar enable>, <mode>, <input>, <source>, <high value>, <low value>, <manual value>[term]`

Format: n,n,a,n,±nnnnnn,±nnnnnn,±nnnnnn

<bipolar enable>	Specifies analog output is 0 = positive output only or 1 = bipolar.
<mode>	Specifies data the analog output monitors. Valid entries: 0 = off, 1 = input, 2 = manual, 3 = loop.
<input>	Specifies which input to monitor if <mode> = 1.
<source>	Specifies input data. Valid entries: 1 = kelvin, 2 = Celsius, 3 = sensor units, 4 = linear equation.
<high value>	If <mode> is 1 , this parameter represents the data at which the analog output reaches +100% output.
<low value>	If <mode> is 1, this parameter represents the data at which the analog output reaches –100% output if bipolar, or 0% output if positive only.
<manual value>	If <mode> is 2, this parameter is the output of the analog output.

Example: **ANALOG 0,1,A,1,100.0,0.0[term]** – Sets analog output to monitor Input A kelvin reading with 100.0 K at +100% output (+10.0 V) and 0.0 K at 0% output (0.0 V).

ANALOG? Analog Output Parameter Query

Input: `ANALOG?[term]`

Returned: <bipolar enable>, <mode>, <input>, <source>, <high value>, <low value>, <manual value>[term]

Format: n,n,a,n,±nnnnnn,±nnnnnn,±nnnnnn (Refer to command for definition)

AOUT? Analog Output Data Query

Input: `AOUT?[term]`

Returned: <analog output>[term]

Format: ±nnn.n

Remarks: Returns the percentage of output of the analog output. Most often used for input or loop modes when the output value is set by the instrument. Resolution is 0.5%.

BAUD RS-232 Baud Rate Command

Input: `BAUD <bps>[term]`

Format: n

<bps>	Specifies Baud rate: 0 = 300 Baud, 1 = 1200 Baud, 2 = 9600 Baud.

BAUD? RS-232 Baud Rate Query

Input: `BAUD?`

Returned: <bps>`[term]`

Format: n (Refer to command for description)

BEEP
Alarm Beeper Command

Input: `BEEP <state>[term]`

Format: n

<state> 0 = Off, 1 = On.

Remarks: Enables or disables system beeper sound when an alarm condition is met.

BEEP?
Alarm Beeper Query

Input: `BEEP?`

Returned: <state>`[term]`

Format: n (Refer to command for description)

BRIGT
Display Brightness Command

Input: `BRIGT <bright>[term]`

Format: n

<bright> 0 = 25%, 1 = 50%, 2 = 75%, 3 = 100%. Default = 2.

BRIGT?
Display Brightness Query

Input: `BRIGT?[term]`

Returned: <bright>[term]

Format: n (Refer to command for description)

CMODE
Control Loop Mode Command

Input: `CMODE <loop>, <mode>[term]`

Format: n,n

<loop>	Specifies which loop to configure: 1 or 2.
<mode>	Specifies the control mode. Valid entries: 1 = Manual PID, 2 = Zone, 3 = Open Loop, 4 = AutoTune PID, 5 = AutoTune PI, 6 = AutoTune P.

Example: **CMODE 1,4[term]** – Control Loop 1 uses PID AutoTuning.

CMODE?
Control Loop Mode Query

Input: `CMODE? <loop>[term]`

Format: n

<loop> Specifies which loop to query: 1 or 2.

Returned: <mode>[term]

Format: n (Refer to command for description)

CRDG?
Celsius Reading Query

Input: `CRDG? <input>[term]`

Format: a

<input> A or B

Returned: <temp value>[term]

Format: ±nnnnnn

Remarks: Also see the RDGST? command.

CRVDEL Curve Delete Command

Input: `CRVDEL <curve>[term]`

Format: nn

<curve>	Specifies a user curve to delete. Valid entries: 21–41.

Example: **CRVDEL 21[term]** – Deletes User Curve 21.

CRVHDR Curve Header Command

Input: `CRVHDR <curve>, <name>, <SN>, <format>, <limit value>, <coefficient>[term]`

Format: nn,aaaaaaaaaaaaaaa,aaaaaaaaaa,n,±nnn.nnn,n

<curve>	Specifies which curve to configure. Valid entries: 21–41.
<name>	Specifies curve name. Limited to 15 characters.
<SN>	Specifies the curve serial number. Limited to 10 characters.
<format>	Specifies the curve data format. Valid entries: 1 = mV/K, 2 = V/K, 3 = Ω/K, 4 = log Ω/K.
<limit value>	Specifies the curve temperature limit in kelvin.
<coefficient>	Specifies the curves temperature coefficient. Valid entries: 1 = negative, 2 = positive.

Remarks: Configures the user curve header.

Example: **CRVHDR 21,DT-470,00011134,2,325.0,1[term]** – Configures User Curve 21 with a name of DT-470, serial number of 00011134, data format of volts versus kelvin, upper temperature limit of 325 K, and negative coefficient.

CRVHDR? Curve Header Query

Input: `CRVHDR? <curve>[term]`

Format: nn

<curve>	Valid entries: 1–41.

Returned: <name>, <SN>, <format>, <limit value>, <coefficient>[term]

Format: aaaaaaaaaaaaaaa,aaaaaaaaaa,n,±nnn.nnn,n (Refer to command for description)

CRVPT Curve Data Point Command

Input: `CRVPT <curve>, <index>, <units value>, <temp value>[term]`

Format: nn,nnn,±nnnnnnn,±nnnnnnn

<curve>	Specifies which curve to configure. Valid entries: 21–41.
<index>	Specifies the points index in the curve. Valid entries: 1–200.
<units value>	Specifies sensor units for this point to 6 digits.
<temp value>	Specifies the corresponding temperature in kelvin for this point to 6 digits.

Remarks: Configures a user curve data point.

Example: **CRVPT 21,2,0.10191,470.000,N[term] –** Sets User Curve 21 second data point to 0.10191 sensor units and 470.000 K.

CRVPT? Curve Data Point Query

Input: `CRVPT? <curve>, <index>[term]`

Format: nn,nnn

<curve>	Specifies which curve to query: 1–41.
<index>	Specifies the points index in the curve: 1–200.

Returned: <units value>, <temp value>[term]

Format: ±nnnnnnn,±nnnnnnn (Refer to command for description)

Remarks: Returns a standard or user curve data point.

CSET Control Loop Parameter Command

Input: `CSET <loop>, <input>, <units>, <powerup enable>, <current/power>[term]`

Format: n,a,n,n,n

<loop>	Specifies which loop to configure: 1 or 2.
<input>	Specifies which input to control from: A or B.
<units>	Specifies setpoint units. Valid entries: 1 = kelvin, 2 = Celsius, 3 = sensor units.
<powerup enable>	Specifies whether the control loop is on or off after power-up, where 0 = powerup enable off and 1 = powerup enable on.
<current/power>	Specifies whether the heater output displays in current or power. Valid entries: 1 = current or 2 = power.

Example: **CSET 1,A,1,1[term]** – Control Loop 1 controls off of Input A with setpoint in kelvin.

CSET? Control Loop Parameter Query

Input: `CSET? <loop>[term]`

Format: n

<loop>	Specifies which loop to query: 1 or 2.

Returned: <input>, <units>, <powerup enable>, <current/power>[term]

Format: a,n,n,n (Refer to command for description)

DFLT Factory Defaults Command

Input: `DFLT 99[term]`

Remarks: Sets all configuration values to factory defaults and resets the instrument. The "99" is included to prevent accidentally setting the unit to defaults.

DISPFLD Displayed Field Command

Input: `DISPFLD <field>, <item>, <source>[term]`

Format: n,n,n

<field>	Specifies field to configure: 1–4.
<item>	Specifies item to display in the field: 0 = Off, 1 = Input A, 2 = Input B, 3 = Setpoint, 4 = Heater Output, 5 = Heater Bar.
<source>	If Item is 1 or 2, specifies input data to display. Valid entries: 1 = kelvin, 2 = Celsius, 3 = sensor units, 4 = linear data, 5 = minimum data, and 6 = maximum data.

Example: **DISPFLD 2,1,1[term]** – Displays kelvin reading for Input A in display field 2.

DISPFLD? Displayed Field Query

Input: `DISPFLD? <field>[term]`

Format: n

<field>	Specifies field to query: 1 – 4.

Returned: <item>, <source>[term]

Format: n,n (Refer to command for description)

EMUL 330 Emulation Mode Command

Input: `EMUL <off/on>[term]`

Format: n

<off/on>	Specifies whether 330 Emulation Mode is 0 = Off or 1 = On. Default = 0.

Remarks: The 330 Emulation Mode allows the remote interface of the Model 332 to be compatible with Model 330 commands. The 330 Emulation Mode only affects remote operation; front panel operation of the Model 332 is not changed. In 330 Emulation Mode, curve locations are mapped to match Model 330 locations. For example, the DT-500-D Curve, found at curve location 3 in the Model 332, is mapped to location 0 when in 330 mode. This applies to the following remote commands: ACUR, ACUR?, BCUR, BCUR?. The following Model 330 commands are not supported in 330 Emulation Mode: CUID?, CURV, CURV?, ECUR, KCUR, and SCAL.

EMUL? 330 Emulation Mode Query

Input: `EMUL?[term]`

Returned: <off/on >[term]

Format: n (Refer to command for description)

FILTER Input Filter Parameter Command

Input: `FILTER <input>, <off/on>, <points>, <window>[term]`

Format: a,n,nn,nn

<input>	Specifies input to configure: A or B.
<off/on>	Specifies whether the filter function is 0 = Off or 1 = On.
<points>	Specifies how many data points the filtering function uses. Valid range = 2 to 64.
<window>	Specifies what percent of full scale reading limits the filtering function. Reading changes greater than this percentage reset the filter. Valid range = 1 to 10%.

Example: **FILTER B,1,10,2[term]** – Filter input B data through 10 readings with 2% of full scale window.

FILTER? Input Filter Parameter Query

Input: `FILTER? <input>[term]`

Format: a

<input>	Specifies input to query: A or B.

Returned: <off/on >, <points>, <window>[term]

Format: n,nn,nn (Refer to command for description)

HTR? Heater Output Query

Input: `HTR?[term]`

Returned: <heater value>[term]

Format: +nnn.n

<heater value>	Loop 1 heater output in percent (%). Use AOUT? for Loop 2.

HTRST? Heater Status Query

Input: `HTRST?[term]`

Returned: <error code>[term]

Format: n

<error code>	Heater error code: 0 = no error, 1 = heater open load, 2 = heater short.

IEEE IEEE-488 Interface Parameter Command

Input: `IEEE <terminator>, <EOI enable>, <address>[term]`

Format: n,n,nn

<terminator>	Specifies the terminator. Valid entries: 0 = <CR><LF>,1 = <LF><CR>, 2 = <LF>, 3 = no terminator (must have EOI enabled).
<EOI enable>	Sets EOI mode: 0 = enabled, 1 = disabled.
<address>	Specifies the IEEE address: 1 – 30. (Address 0 and 31 are reserved.)

Example: **IEEE 0,0,4[term]** – After receipt of the current terminator, the instrument uses EOI mode, uses <CR><LF> as the new terminator, and responds to address 4.

IEEE? IEEE-488 Interface Parameter Query

Input: `IEEE?[term]`

Returned: <terminator>, <EOI enable>, <address>[term]

Format: n,n,nn (Refer to command for description)

INCRV Input Curve Number Command

Input: `INCRV <input>, <curve number>[term]`

Format: a,nn

<input>	Specifies which input to configure: A or B.
<curve number>	Specifies which curve the input uses. If specified curve parameters do not match the input, the curve number defaults to 0. Valid entries: 0 = none, 1 – 20 = standard curves, 21 – 41 = user curves.

Remarks: Specifies the curve an input uses for temperature conversion.

Example: **INCRV A,23[term]** – Input A uses User Curve 23 for temperature conversion.

INCRV? Input Curve Number Query

Input: `INCRV? <input>[term]`

Format: a

<input>	Specifies which input to query: A or B.

Returned: <curve number>[term]

Format: nn (Refer to command for description)

INTYPE Input Type Parameter Command

Input: `INTYPE <input>, <sensor type>, <compensation>[term]`

Format: a,n,n

<input> Specifies input to configure: A or B.

<sensor type> Specifies input sensor type. Valid entries:

0 = Silicon Diode	8 = NTC RTD 75mV 75 Ω
1 = GaAlAs Diode	9 = NTC RTD 75mV 750 Ω
2 = Platinum 100/250 Ω	10 = NTC RTD 75mV 7.5 kΩ
3 = Platinum 100/500 Ω	11 = NTC RTD 75mV 75 kΩ
4 = Platinum 1000 Ω	12 = NTC RTD 75mV Auto
5 = NTC RTD 75mV 7.5 kΩ	
6 = Thermocouple 25 mV	
7 = Thermocouple 50 mV	

<compensation> Specifies input compensation where 0 = off and 1 = on. Reversal for thermal EMF compensation if input is resistive, room compensation if input is thermocouple. Always 0 if input is a diode.

Remarks: Sensor type NTC RTD 75mV 7.5kΩ listed twice to maintain compatibility with Model 331 INTYPE command.

Example: **INTYPE A,0,0[term]** – Sets Input A sensor type to silicon diode.

INTYPE? Input Type Parameter Query

Input: `INTYPE? <input>[term]`

Format: a

<input> Specifies input to query: A or B.

Returned: <sensor type>, <compensation>[term]

Format: n,n (Refer to command for description)

KEYST? Keypad Status Query

Input: **KEYST?** `[term]`

Returned: <keypad status>[term]

Format: n 1 = key pressed, 0 = no key pressed.

Remarks: Returns keypad status since the last KEYST?. KEYST? returns 1 after initial power-up.

KRDG? Kelvin Reading Query

Input: `KRDG? <input>[term]`

Format: a

<input> Specifies which input to query: A or B.

Returned: <kelvin value>[term]

Format: ±nnnnnn

Remarks: Also see the RDGST? command.

LDAT? Linear Equation Data Query

Input: `LDAT? <input>[term]`

Format: a

<input>	Specifies which input to query: A or B.

Returned: <linear value>[term]

Format: ±nnnnnn

Remarks: Also see the RDGST? command.

LINEAR Input Linear Equation Parameter Command

Input: `LINEAR <input>, <equation>, <varM value>, <X source>, <B source>, <varB value>[term]`

Format: a,n,±nnnnnn,n,n,±nnnnnn

<input>	Specifies input to configure: A or B.
<equation>	Specifies linear equation to use. Valid entries: 1 = (y = mx + b), 2 = (y = m(x + b)).
<varM value>	Specifies a value for m in the equation.
<X source >	Specifies input data to use. Valid entries: 1 = kelvin, 2 = Celsius, 3 = sensor units.
<B source >	Specifies what to use for b in the equation. To use a setpoint, set its units to the same type specified in <X source>. Valid entries: 1 = a value, 2 = +SP1, 3 = -SP1, 4 = +SP2, 5 = -SP2.
<varB value>	Specifies a value for b in the equation if <B source> is 1.

Example: **LINEAR A,1,1.0,1,3[term] –** The linear data for Input A is calculated from the kelvin reading of the input using the equation: y = 1.0 * x – SP1.

LINEAR? Input Linear Equation Parameter Query

Input: `LINEAR? <input>[term]`

Format: a

<input>	Specifies which input to query: A or B.

Returned: <equation>, <varM value>, <X source>, <B source>, <varB value>[term]

Format: n,±nnnnnn,n,n,±nnnnnn (Refer to command for description)

Remarks: Returns input linear equation configuration.

LOCK Front Panel Keyboard Lock Command

Input: `LOCK <state>, <code>[term]`

Format: n,nnn

<state>	0 = Unlocked, 1 = Locked
<code>	Specifies lock-out code. Valid entries are 000– 999.

Remarks: Locks out all front panel entries except pressing the **Alarm** key to silence alarms. Refer to Paragraph 4.17. Use the CODE command to set the lock code.

Example: **LOCK 1,123[term] –** Enables keypad lock and sets the code to 123.

LOCK? Front Panel Keyboard Lock Query

Input: `LOCK?[term]`

Returned: <state>, <code>[term]

Format: n,nnn (Refer to command for description)

MDAT? Minimum/Maximum Data Query

Input: `MDAT? <input>[term]`

Format: a

<input> Specifies which input to query: A or B.

Returned: <min value>,<max value>[term]

Format: ±nnnnnn,±nnnnnn

Remarks: Returns the minimum and maximum input data. Also see the RDGST? command.

MNMX Minimum and Maximum Input Function Parameter Command

Input: `MNMX <input>, <source>[term]`

Format: a,n

<input> Specifies input to configure: A or B.

<source> Specifies input data to process through max/min. Valid entries: 1 = kelvin, 2 = Celsius, 3 = sensor units, 4 = linear data.

Example: **MNMX B,3[term]** – Input B min/max function is on and processes data from the input sensor units reading.

MNMX? Minimum and Maximum Input Function Parameter Query

Input: `MNMX? <input>[term]`

Format: a

<input> Specifies which input to query: A or B.

Returned: <source>[term]

Format: n (Refer to command for description)

MNMXRST Minimum and Maximum Function Reset Command

Input: `MNMXRST[term]`

Remarks: Resets the minimum and maximum data for all inputs.

MODE Remote Interface Mode Command

Input: `MODE <mode>[term]`

Format: n

<mode> 0 = local, 1 = remote, 2 = remote with local lockout.

Example: **MODE 2[term] –** Places the Model 332 into remote mode with local lockout.

MODE? Remote Interface Mode Query

Input: `MODE?[term]`

Returned: <mode>[term]

Format: n (Refer to command for description)

MOUT Control Loop Manual Heater Power (MHP) Output Command

Input: `MOUT <loop>, <value>[term]`

Format: n,±nnnnnn[term]

<loop>	Specifies loop to configure: 1 or 2.
<value>	Specifies value for manual output.

Example: **MOUT 1,22.45[term]** – Control Loop 1 manual heater power output is 22.45%.

MOUT? Control Loop Manual Heater Power (MHP) Output Query

Input: `MOUT? <loop>[term]`

Format: n

<loop> Specifies which loop to query: 1 or 2.

Returned: <value>

Format: ±nnnnnn[term] (Refer to command for description)

PID Control Loop PID Values Command

Input: `PID <loop>, <P value>, <I value>, <D value>[term]`

Format: n,±nnnnnn,±nnnnnn,±nnnnnn

<loop>	Specifies loop to configure: 1 or 2.
<P value>	The value for control loop Proportional (gain): 0.1 to 1000.
<I value>	The value for control loop Integral (reset): 0.1 to 1000.
<D value>	The value for control loop Derivative (rate): 0 to 200.

Remarks: Setting resolution is less than 6 digits indicated.

Example: **PID 1,10,50[term]** – Control Loop 1 P is 10 and I is 50.

PID? Control Loop PID Values Query

Input: `PID? <loop>[term]`

Format: n

<loop> Specifies which loop to query: 1 or 2.

Returned: <P value>, <I value>, <D value>[term]

Format: ±nnnnnn,±nnnnnn,±nnnnnn (Refer to command for description)

RAMP Control Setpoint Ramp Parameter Command

Input: `RAMP <loop>, <off/on>, <rate value>[term]`

Format: n,n,±nnnnn

<loop>	Specifies which loop to configure: 1 or 2.
<off/on>	Specifies whether ramping is 0 = Off or 1 = On.
<rate value>	Specifies setpoint ramp rate in Kelvin per minute from 0.1 to 100. The rate is always positive, but will respond to ramps up or down.

Example: **RAMP 1,1,10.5[term]** – When Control Loop 1 setpoint is changed, ramp the current setpoint to the target setpoint at 10.5 K/minute.

RAMP? Control Setpoint Ramp Parameter Query

Input: `RAMP? <loop>`

Format: n

<loop> Specifies which loop to query: 1 or 2.

Returned: <off/on>, <rate value>[term]

Format: n,±nnnnn (Refer to command for description)

RAMPST? Control Setpoint Ramp Status Query

Input: `RAMPST? <loop>[term]`

Format: n

<loop> Specifies which loop to query: 1 or 2.

Returned: <ramp status>[term]

Format: n

<ramp status> 0 = Not ramping, 1 = Setpoint is ramping.

RANGE Heater Range Command

Input: `RANGE <range>[term]`

Format: n 0 = Off, 1 = Low (0.5 W), 2 = Medium (5 W), 3 = High (50 W)

RANGE? Heater Range Query

Input: `RANGE?[term]`

Returned: <range>[term]

Format: n (Refer to command for description)

RDGST? Input Reading Status Query

Input: `RDGST? <input>[term]`

Format: a

<input> Specifies which input to query: A or B.

Returned: <status bit weighting>[term]

Format: nnn

Remarks: The integer returned represents the sum of the bit weighting of the input status flag bits. A "000" response indicates a valid reading is present.

Bit	Bit Weighting	Status Indicator
0	1	invalid reading
4	16	temp underrange
5	32	temp overrange
6	64	sensor units zero
7	128	sensor units overrange

RELAY Relay Control Parameter Command

Input: `RELAY <relay number>, <mode>, <input alarm>, <alarm type>[term]`

Format: n,n,a,n

<relay number>	Specifies which relay to configure: 1 or 2.
<mode>	Specifies relay mode. 0 = Off, 1 = On, 2 = Alarms.
<input alarm>	Specifies which input alarm activates the relay when the relay is in alarm mode: A or B.
<alarm type>	Specifies the input alarm type that activates the relay when the relay is in alarm mode. 0 = Low alarm, 1 = High Alarm, 2 = Both Alarms.

Example: **RELAY 1,2,B,0[term]** – Relay 1 activates when Input B low alarm activates.

RELAY? Relay Control Parameter Query

Input: `RELAY? <relay number>[term]`

Format: n

<relay number> Specifies which relay to query: 1 or 2.

Returned: n,a,n (Refer to command for description)

RELAYST? Relay Status Query

Input: `RELAYST? <high/low>`

Format: n

<high/low> Specifies relay type to query: 1 = Low Alarm or 1 = High Alarm.

Returned: <status>[term]

Format: n 0 = Off, 1 = On.

REV? Input Firmware Revision Query

Input: `REV?[term]`

Returned: <revision>[term]

Format: n.n

Remarks: Returns the version number of the input firmware installed in the instrument.

SCAL

Generate SoftCal Curve Command

Input: `SCAL <std>, <dest>, <SN>, <T1 value>, <U1 value>, <T2 value>, <U2 value>, <T3 value>, <U3 value>[term]`

Format: n,nn,aaaaaaaaaa,±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn

<std>	Specifies the standard curve to generate a SoftCal from. Valid entries: 1, 6, 7.
<dest>	Specifies the user curve to store the SoftCal curve. Valid entries: 21–41.
<SN>	Specifies the curve serial number. Limited to 10 characters.
<T1 value>	Specifies first temperature point.
<U1 value>	Specifies first sensor units point.
<T2 value>	Specifies second temperature point.
<U2 value>	Specifies second sensor units point.
<T3 value>	Specifies third temperature point.
<U3 value>	Specifies third sensor units point.

Remarks: Generates a SoftCal curve. Refer to Paragraph 5.3.

Example: **SCAL 1,21,1234567890,4.2,1.6260,77.32,1.0205,300.0,0.5189[term]** – Generates a three-point SoftCal curve from standard curve 1 and saves it in user curve 21.

SETP

Control Setpoint Command

Input: `SETP <loop>, <value>[term]`

Format: n,±nnnnnn

<loop>	Specifies which loop to configure.
<value>	The value for the setpoint (in whatever units the setpoint is using).

Example: **SETP 1,122.5[term] –** Control Loop 1 setpoint is now 122.5 (based on its units).

SETP?

Control Setpoint Query

Input: `SETP? <loop>[term]`

Format: n

<loop>	Specifies which loop to query: 1 or 2.

Returned: <value>[term]

Format: ±nnnnnn

SRDG?

Sensor Units Input Reading Query

Input: `SRDG? <input>[term]`

Format: a

<input>	Specifies which input to query: A or B.

Returned: <sensor units value>[term]

Format: ±nnnnnn

Remarks: Also see the RDGST? command.

TEMP? Thermocouple Junction Temperature Query

Input: `TEMP?`

Returned: <junction temperature>[term]

Format: ±nnnnnnn

Remarks: Temperature is in kelvin.

TUNEST? Control Tuning Status Query

Input: `TUNEST?`

Returned: <tuning status>[term]

Format: n 0 = no active tuning, 1 = active tuning.

ZONE Control Loop Zone Table Parameter Command

Input: **`ZONE <loop>, <zone>, <top value>, <P value>, <I value>, <D value>, <mout value>, <range>[term]`**

Format: n,nn,±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn,n[term]

<loop>	Specifies which loop to configure: 1 or 2.
<zone>	Specifies which zone in the table to configure. Valid entries are: 1– 10.
<top value>	Specifies the top temperature of this zone.
<P value>	Specifies the P for this zone: 0.1 to 1000.
<I value>	Specifies the I for this zone: 0.1 to 1000.
<D value>	Specifies the D for this zone: 0 to 200%.
<mout value>	Specifies the manual output for this zone: 0 to 100%.
<range>	Specifies the heater range for this zone if <loop> = 1. Valid entries: 0– 3.

Remarks: Configures the control loop zone parameters. Refer to Paragraph 2.9.

Example: **ZONE 1,1,25.0,10,20,0,0,2[term]** – Control Loop 1 zone 1 is valid to 25.0 K with P = 10, I = 20, D = 0, and a heater range of 2.

ZONE? Control Loop Zone Table Parameter Query

Input: **`ZONE? <loop>, <zone>[term]`**

Format: n,nn

<loop>	Specifies which loop to query: 1 or 2.
<zone>	Specifies which zone in the table to query. Valid entries: 1– 10.

Returned: <top value>, <P value>, <I value>, <D value>, <mout value>, <range>[term]

Format: ±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn,±nnnnnn,n (Refer to command for description)