Contents

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard product is warranted against defects in materials and workmanship for a period of three years from date of shipment. Duration and conditions of warranty for this product may be superseded when the product is integrated into (becomes a part of) other HP products. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard (HP). Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with a product will execute its programming instructions when properly installed on that product. HP does not warrant that the operation of the product, or software, or firmware will be uninterrupted or error free.

Limitation Of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied products or int`erfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

The design and implementation of any circuit on this product is the sole responsibility of the Buyer. HP does not warrant the Buyer's circuitry or malfunctions of HP products that result from the Buyer's circuitry. In addition, HP does not warrant any damage that occurs as a result of the Buyer's circuit or any defects that result from Buyer-supplied products.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

Notice

The information contained in this document is subject to change without notice. HEWLETT-PACKARD (HP) MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. HP assumes no responsibility for the use or reliability of its software on equipment that is not furnished by HP.

U.S. Government Restricted Rights

The Software and Documentation have been developed entirely at private expense. They are delivered and licensed as "commercial computer software" as defined in DFARS 252.227-7013 (October 1988), DFARS 252.211.7015 (May 1991) or DFARS 252.227-7014 (June 1995), as a "commercial item" as defined in FAR 2.101(a), or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) (or any equivalent agency regulation or contract clause), whichever is applicable. You have only those rights provided for such Software and Documentation by the applicable FAR or DFARS clause or the HP standard software agreement for the product involved.

HP E1406A User's Manual
Edition 4

Documentation History

All Editions and Updates of this manual and their creation date are listed below. The first Edition of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct or add additional information to the current Edition of the manual. Whenever a new Edition is created, it will contain all of the Update information for the previous Edition. Each new Edition or Update also includes a revised copy of this documentation history page.

Edition 1 . . . . . June 1992
Edition 2 . . . . . February 1993
Edition 3 . . . . . October 1994
Edition 4 . . . . . May 1996

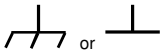
Trademark Information

Mircosoft® and MS-DOS® are U.S. registered trademarks of Microsoft Corp.

Safety Symbols

Instruction manual symbol affixed to product. Indicates that the user must refer to the manual for specific WARNING or CAUTION information to avoid personal injury or damage to the product.

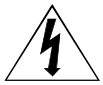
Indicates the field wiring terminal that must be connected to earth ground before operating the equipment—protects against electrical shock in case of fault.



Frame or chassis ground terminal—typically connects to the equipment's metal frame.

Alternating current (AC).

Direct current (DC).



Indicates hazardous voltages.

WARNING Calls attention to a procedure, practice, or condition that could cause bodily injury or death.

CAUTION Calls attention to a procedure, practice, or condition that could possibly cause damage to equipment or permanent loss of data.

WARNINGS

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground the equipment: For Safety Class 1 equipment (equipment having a protective earth terminal), an uninterruptible safety earth ground must be provided from the mains power source to the product input wiring terminals or supplied power cable.

DO NOT operate the product in an explosive atmosphere or in the presence of flammable gases or fumes.

For continued protection against fire, replace the line fuse(s) only with fuse(s) of the same voltage and current rating and type. DO NOT use repaired fuses or short-circuited fuse holders.

Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers or shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.

DO NOT operate damaged equipment: Whenever it is possible that the safety protection features built into this product have been impaired, either through physical damage, excessive moisture, or any other reason, REMOVE POWER and do not use the product until safe operation can be verified by service-trained personnel. If necessary, return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

DO NOT service or adjust alone: Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT substitute parts or modify equipment: Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Declaration of Conformity

according to ISO/IEC Guide 22 and EN 45014

Manufacturer's Name: Hewlett-Packard Company
Loveland Manufacturing Center

Manufacturer's Address: 815 14th Street S.W.
Loveland, Colorado 80537

declares, that the product:

Product Name: HP-IB Command Module

Model Number: E1406A

Product Options: All

conforms to the following Product Specifications:

Safety: IEC 1010-1 (1990) Incl. Amend 1 (1992)/EN61010-1 (1993)
CSA C22.2 #1010.1 (1992)
UL 3111

EMC: CISPR 11:1990/EN55011 (1991): Group 1 Class A
IEC1000-4-2:1995/prEN50082-1 (1995): 4 kV CD, 8 kV AD
ENV50140:1993/prEN50082-1 (1995): 3 V/m
IEC1000-4-4:1995/prEN50082-1 (1995): 1 kV Power Line
0.5 kV Signal Lines
ENV50141:1993/prEN50082-1 (1995): 3 Vrms
ENV50142:1994/prEN50082-1 (1995): 1 kV CM, 0.5 kV DM
IEC1000-4-8:1993/prEN50082-1 (1995): 3 A/m
EN61000-4-11:1994/prEN50082-1 (1995): 30%, 10 ms 60%, 100 ms

Supplementary Information: The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (inclusive 93/68/EEC) and carries the "CE" marking accordingly.

Tested in a typical HP C-Size VXI Mainframe configuration.

December 30, 1995 **Jim White, QA Manager**

European contact: Your local Hewlett-Packard Sales and Service Office or Hewlett-Packard GmbH, Department HQ-TRE, Herrenberger Straße 130, D-71034 Böblingen, Germany (FAX +49-7031-14-3143).

Notes

Please fold and tape for mailing

Reader Comment Sheet

HP E1406A User's Manual
Edition 4

You can help us improve our manuals by sharing your comments and suggestions. **In appreciation of your time, we will enter you in a quarterly drawing for a Hewlett-Packard Palmtop Personal Computer** (U.S. government employees cannot participate in the drawing).

Your Name	City, State/Province
Company Name	Country
Job Title	Zip/Postal Code
Address	Telephone Number with Area Code

Please list the system controller, operating system, programming language, and plug-in modules you are using.

fold here

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 37 LOVELAND, CO

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
Measurement Systems Division
Learning Products Department
P.O. Box 301
Loveland, CO 80539-9984

cut along this line

fold here

Please pencil-in one circle for each statement below:

	Disagree ←				→ Agree
• The documentation is well organized.	O	O	O	O	O
• Instructions are easy to understand.	O	O	O	O	O
• The documentation is clearly written.	O	O	O	O	O
• Examples are clear and useful.	O	O	O	O	O
• Illustrations are clear and helpful.	O	O	O	O	O
• The documentation meets my overall expectations.	O	O	O	O	O

Please write any comments or suggestions below--be specific.

Chapter 1
HP E1406A Command Module Overview

About This Chapter

This chapter contains WARNINGS and CAUTIONS, a functional and physical overview of the HP E1406A Command Module, and instructions on installing the command module in a mainframe. Chapter contents are as follows:

Warnings and Cautions

WARNING **SHOCK HAZARD. Only qualified, service-trained personnel who are aware of the hazards involved should install, configure, or remove the multiplexer module. Disconnect all power sources from the mainframe, the terminal modules, and installed modules before installing or removing a module.**

CAUTION **STATIC ELECTRICITY.** Static electricity is a major cause of component failure. To prevent damage to the electrical components in the multiplexer, observe anti-static techniques whenever removing, configuring, and installing a module. The multiplexer is susceptible to static discharges. Do not install the multiplexer module without its metal shield attached.

Using HP VIC

Our goal is to make installation of your C-size VXI system as easy as possible. To o so, HP VIC (HP VXI Installation Consultant) is provided with the HP E1406A Command Module. HP VIC is a Microsoft® Windows™ program that helps you configure and install the HP E1406A Command Module - based on VXI systems. If your system contains an HP E1406A Command Module to be controlled by a computer external to the VXI mainframe, it is recommended you configure your system using HP VIC.

Command Module Functional Description

The HP E1406A Command Module is the foundation of a VXIbus system (see Figure 1-1). Though its role in a VXIbus system is largely transparent (for example, the user need not program its functions) it provides the following key functions:

- Translates SCPI (Standard Commands for Programmable Instruments) commands for HP register-based instruments.
- Provides the VXIbus slot 0 and resource manager capabilities.
- Can drive the VXIbus TTLTRG0-7 and ECLTRG0-1 trigger lines. The module contains SMB connectors for placing an external trigger onto the selected line(s), and for routing an internal trigger to a device external to the mainframe.
- Contains an internal clock that allows you to set and read the time and date.
- Is the Hewlett-Packard Interface Bus (HP-IB) to VXIbus interface.

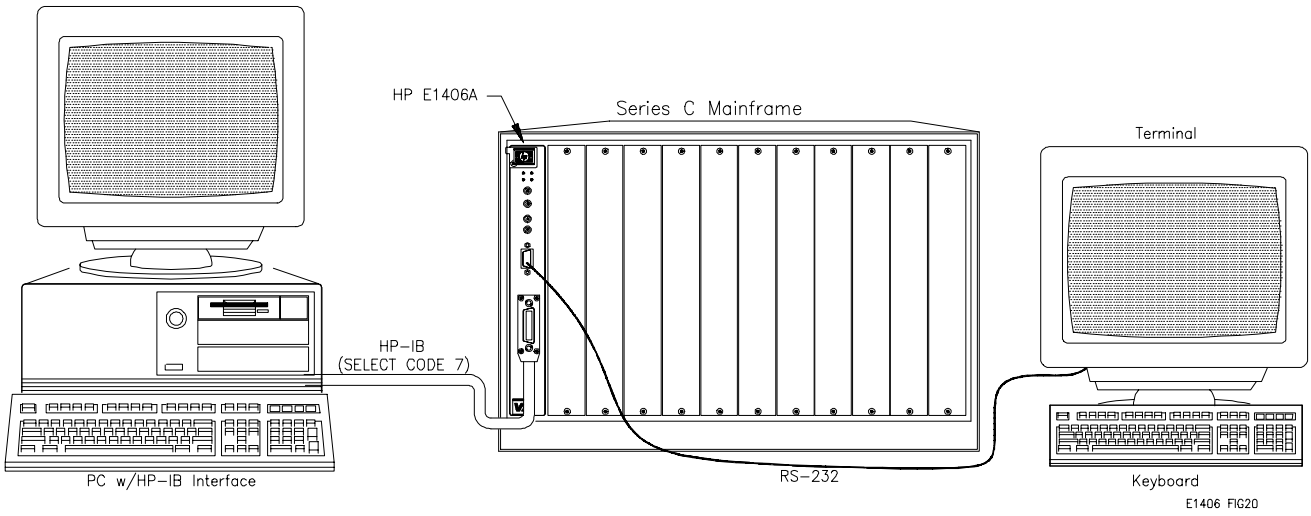


Figure 1-1. VXIbus System

Command Module Physical Description

The HP E1406A Command Module occupies one C-size mainframe slot. The faceplate has annunciators, clock and trigger connectors, interface ports, and extraction levers that are described below.

Faceplate Annunciators

There are four annunciators on the HP E1406A faceplate which show the following:

Failed Shows that the command module has failed its power-on self-test or has stopped working at some point in time.

SYSFAIL Shows that the SYSFAIL line on the VXIbus backplane is being asserted by the command module when it fails.

Access Shows that the command module is accessing, or being accessed by the VXIbus backplane.

Ready Shows that the command module is in the VXIbus normal operation state.

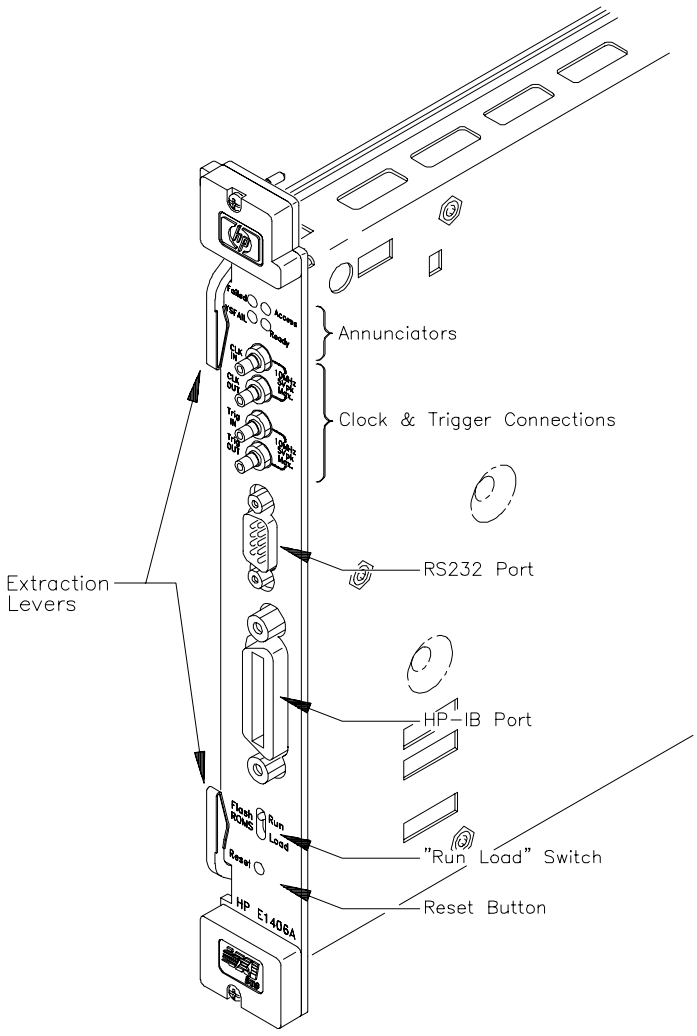


Figure 1-2. HP E1406A Command Module Faceplate

Faceplate CLK10 and Trigger Connectors

There are four signal connectors on the HP E1406A faceplate which function as follows:

Clk In This SMB connector allows an external 10 MHz clock to function as the system's slot 0 CLK10 resource. This is a high impedance input with an input range from ± 40 mV to ± 42.5 V.

Clk Out This SMB connector allows the internal slot 0 CLK10 resource to be routed to other VXIbus mainframes. This output is a TTL level output and drives 50 Ω.

Trig In This SMB connector allows an external trigger signal (TTL levels) to be applied to the system on the trigger line selected (TTLTRG0-7/ECLTRG0-1). The input impedance is 5 kΩ.

Trig Out This SMB connector allows an internal trigger on the trigger line specified (TTLTRG0-7/ECLTRG0-1) to be applied to an external device. This output is a TTL level output and drives 50 Ω.

The HP-IB and RS-232 Ports

The HP-IB port allows an HP-IB cable to be connected from the HP E1406A to a computer, or to an external disk drive. The RS-232 port can be used as a user interface, or used for peripheral control if the HP E1406A contains Instrument BASIC (IBASIC). The RS-232 port is a 9-pin DTE connector. Supported terminals include: HP 700/92, HP 700/94, HP 700/22, HP 700/43, Wyse WY-30, DEC VT 100, and DEC VT 220.

The Run/Load Switch

The run/load switch is located beneath the HP-IB port. This switch lets you activate the loader instrument so that you can reprogram the Flash ROM or download device drivers to the Flash ROM.

The Reset Button

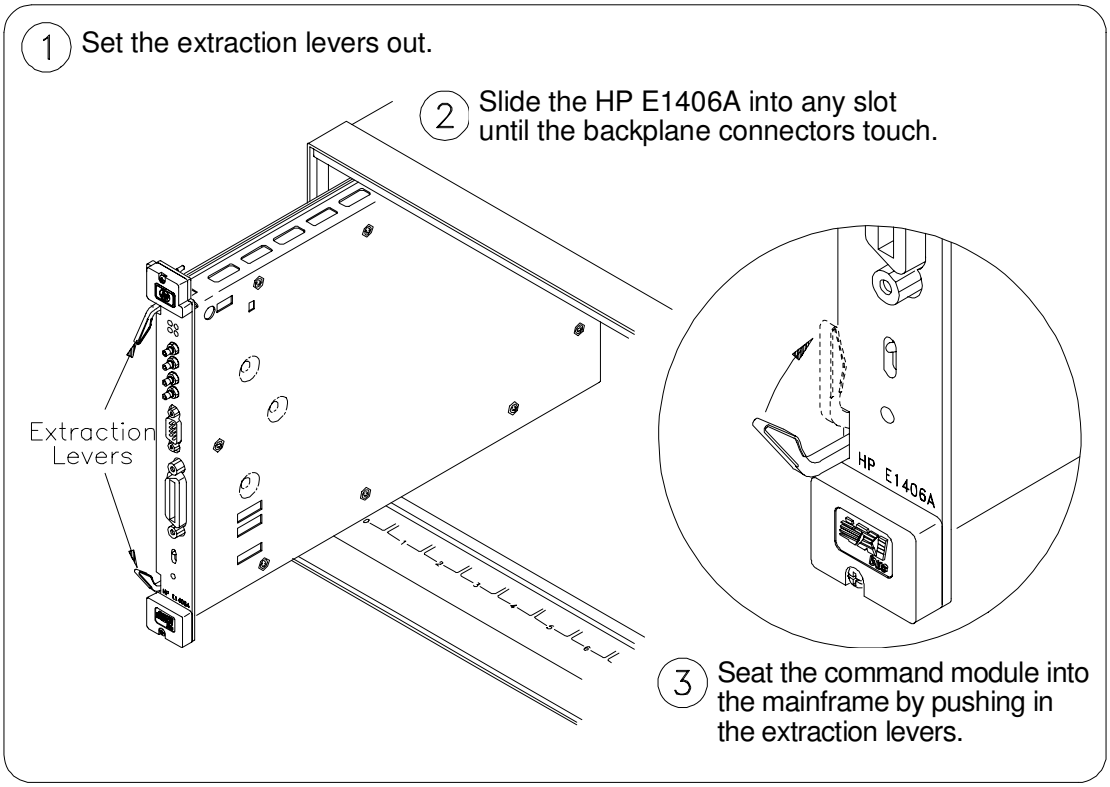
The reset button is located beneath the run/load switch. This button is used to reconfigure your VXIbus system and return it to the power-on state.

Extraction Levers

The extraction levers provide easy insertion into and extraction from the C-size mainframe.

Installing the Command Module in a Mainframe

Refer to Figure 1-3 to install the HP E1406A Command Module in a C-size mainframe.



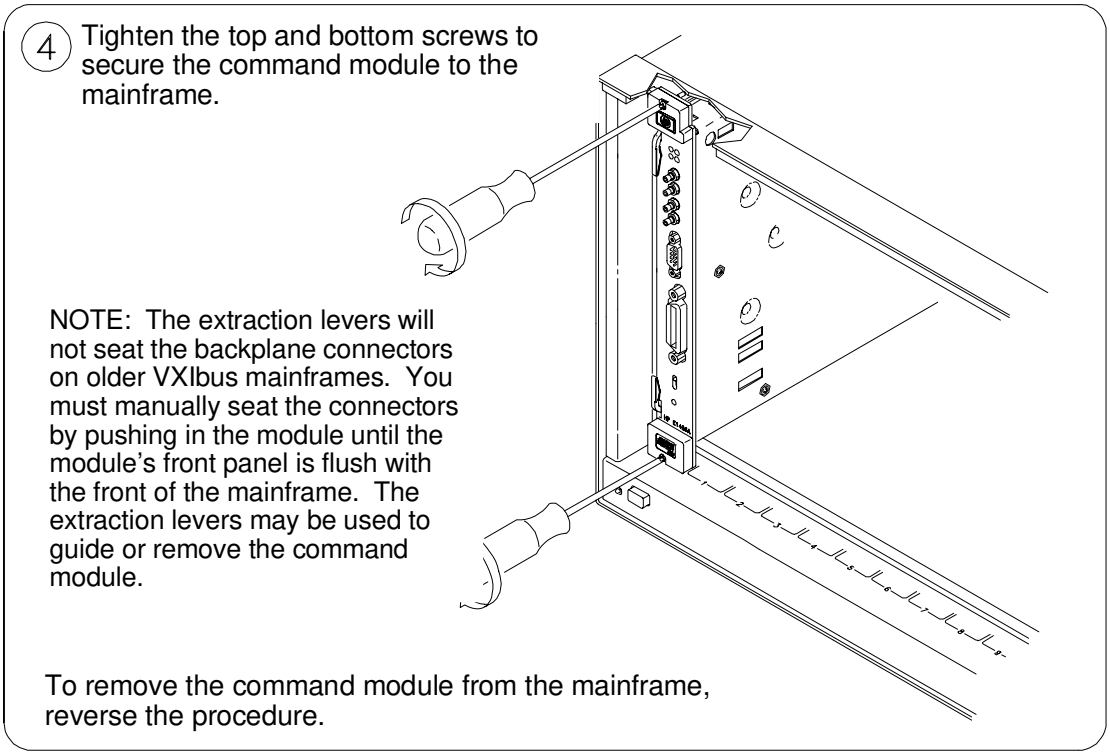


Figure 1-3. Installing the Command Module in a VXIbus Mainframe

Command Module Memory

The HP E1406A comes from the factory equipped with 512 KB of RAM and 1.25 MB of Flash ROM. HP E1406A Option 010 provides 1.75 MB of Flash ROM and 1 MB of RAM.

For applications which do not require shared RAM, the non-volatile RAM can be configured to a full 2 MB if the extra 512 KB of RAM and 512 KB of Flash ROM has been installed.

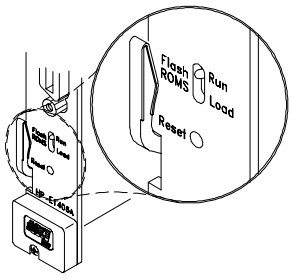
Battery Backed Functions

The HP E1406A clock and calendar functions, the user non-volatile RAM (NRAM), and the device driver RAM (DRAM) are backed up by a NiCad battery. For systems with 512 KB of memory this battery has a ten month lifetime and is fully recharged when the command module is in the mainframe and the power has been on for fifteen continuous hours. This battery has a five month lifetime for systems with 1 MB of RAM and 2.5 month lifetime for systems with 2 MB of RAM.

Chapter 2
Configuring the HP E1406A Command Module

About This Chapter

One purpose of the HP E1406A is to provide the resource manager function required by VXIbus systems. This chapter describes the resource manager's function and shows you how to modify the configuration process with user tables you download into non-volatile user RAM. All of these functions require the Flash ROMS Run/Load switch be set to "Run".



The main sections of this chapter include:

System Configuration Sequence

As mentioned in the *C-Size VXIbus Systems Configuration Guide*, the resource manager within the HP E1406A Command Module performs the following system configuration sequence when power is applied:

- Identify all statically and dynamically configured plug-in modules installed in the C-size mainframe.
- Set commander/servant hierarchies whereby one or more plug-in modules *control* other plug-in modules.
- Perform A24/A32 address mapping so modules requiring additional addressing can receive it.
- Allocate interrupt lines to manage communication between interrupt handler modules and interrupter modules.
- Start system operation.

Once the power-on sequence is completed and the system is started, the resource manager is no longer used.

The following sections describe each step of the configuration sequence. Included are examples on how to change the sequence using configuration tables stored in non-volatile user RAM.

Note Refer to the *C-Size VXIbus Systems Configuration Guide* for information on configuring the HP E1406A Command Module as the resource manager.

Modules Configured Statically and Dynamically

Statically configured modules are plug-in modules whose logical addresses are set with logical address switches. Dynamically configured modules are plug-in modules whose logical addresses are programmed (set) by the resource manager.

Identifying Statically Configured Modules

Once all power-on self tests have completed, the resource manager identifies all statically configured modules. The resource manager retains information such as the module's logical address, slot number, model number, manufacturer's code, and so forth.

Identifying Dynamically Configured Modules

Once all statically configured modules have been located in a mainframe and none have a logical address of 255, the resource manager identifies all dynamically configured modules and assigns them logical addresses as follows.

- The resource manager locates dynamically configured modules by scanning each mainframe slot. Refer to the plug-in module manual for additional information on setting up the module prior to its dynamic configuration.
- Beginning with the lowest mainframe slot (excluding slot 0), the resource manager scans each slot via the module identification (MODID) bus until a dynamically configured module is located. The module is assigned a logical address that is the lowest available multiple of 8.
- The resource manager continues scanning until the next dynamically configured module is located. The module is assigned a logical address that is the next available multiple of 8. The process continues until all dynamically configured devices have been assigned logical addresses. If all multiples of 8 are used, the dynamically configured module is assigned the first available address.
- Logical addresses used by statically configured devices will not be assigned to dynamically configured devices.
- Dynamically configured devices will not be assigned logical address 255.
- A set of address blocked dynamically configured devices will be assigned successive logical addresses beginning with the lowest available multiple of 8.

User-Defined Dynamic Configuration

If your system contains instruments comprised of multiple modules that must have successive logical addresses, then the modules must be statically configured using their logical address switches, or be dynamically configured with the user-defined dynamic configuration table. The dynamic configuration table covered in this section allows you to override the default configuration process by assigning logical addresses as you choose.

The Dynamic Configuration Table

User-defined dynamic configurations are specified with a dynamic configuration table created in the command module. The table is created as follows:

1. Table space in the command module's non-volatile user RAM is made available by allocating a segment of RAM with the command:

   DIAGnostic:NRAM:CREate <*size*>

2. Reset the command module. NRAM is created during the boot-up process:

   DIAGnostic:BOOT:WARM

3. The location (starting address) of the table in RAM is determined with the command:

   DIAGnostic:NRAM:ADDRess?

4. Data is downloaded into the table with the command:

   DIAGnostic:DOWNload <*address*>,<*data*>

5. The table is linked to the appropriate algorithm in the command module processor with the command:

   VXI:CONFigure:DCTable <*address*>

Table Format

The format of the dynamic configuration table is shown in Table 2-1.

Table 2-1. Dynamic Configuration Table Format

Valid Flag	Number of Entries		
Slot Number	Slot 0 Laddr	Laddr	Block Size
Slot Number	Slot 0 Laddr	Laddr	Block Size
•	•	•	•
•	•	•	•
Slot Number	Slot 0 Laddr	Laddr	Block Size

The table parameters are:

- **Valid Flag (1/0)** 1 (one) indicates the table is valid and the modules can be configured accordingly. 0 (zero) will cause an error message (Error 39). Valid Flag is part of the table header and is one byte.
- **Number of Entries (1 - 254)** is the number of entries in the table. Number of Entries is part of the table header and is one byte.

- **Slot Number (1 - 12)** is the mainframe slot the module to be assigned an address is installed in. Field is one byte.
- **Slot 0 Laddr** is the logical address of the slot 0 device. This is 0 (zero) in mainframe #1 but will be different in any additional mainframes. Field is one byte.
- **Laddr (1 - 254)** is the logical address to which the module in **Slot Number** is set. Field is one byte.
- **Block Size (1 - 128)** is the number of devices in an address block. When there is more than one device, **Laddr** specifies the logical address of the first device in the set. The remaining devices are assigned sequential logical addresses beginning with the next highest address. When there are multiple devices in a slot that are not address blocked, there must be an entry in the table for each device. Field is one byte.

Determining the Table Size

The dynamic configuration table has a two byte header and each of the four fields are one byte. The amount of RAM to allocate with DIAGnostic:NRAM:CREate is computed as:

```
2 + 4(N)
```

where N is the number of modules to be configured. For example, to dynamically configure three modules based on logical addresses you have selected, the table size would be: 2 + 4(3) = 14 bytes. DIAGnostic:NRAM:CREate would be executed as:

```
OUTPUT  @E1406;"DIAG:NRAM:CRE  14"
```

Data Format

Data can be sent to the dynamic configuration table in any convenient format, as long as the binary data is preserved. This can be accomplished using DIAGnostic:PEEK? and DIAGnostic:POKE, by reading the data into a variable in the computer and then downloading the data to the table using the Arbitrary Block Program Data format, and so forth. In the following example, this is accomplished by reading the data into 16-bit integer variables in the computer and then downloading the data to the table using the ANSI/IEEE 488.2-1987 Arbitrary Block Program Data format. More information on the Arbitrary Block Program format can be found on page 121 of this manual and in the *ANSI/IEEE 488.2-1987* document.

CAUTION **When downloading data into the dynamic configuration table, DIAGnostic:DOWNload does not determine if the table is large enough to store the data. If the amount of data sent by DIAGnostic:DOWNload is greater than the (table) space allocated by DIAGnostic:NRAM:CREate, system errors will occur. You can recover from these errors by executing DIAGnostic:BOOT:COLD, or by pressing the "Ctrl-R" keys on an RS-232 terminal while cycling mainframe power.**

Example: Dynamically Configuring a Module

The following program dynamically sets the logical address of the HP E1412A 6½-Digit Multimeter in slot 6 to 32. The program notes each of the steps used to create and load the table.

To dynamically configure the multimeter, its logical address must be set to 255 using the logical address switches.

```
10   !Assign an I/O path and allocate a variable to store dynamic configuration
20   !data to be downloaded to the command module.
30   ASSIGN  @E1406 TO 70900;EOL    CHR$(10)    END
40   INTEGER  Dy_config(1:6)
50   !
60   !Allocate a segment of non-volatile user RAM on the command
70   !module to store the dynamic configuration table (1 module).
80   OUTPUT  @E1406;"DIAG:NRAM:CRE  6"
90   !
100  !Restart the system instrument to allocate the user RAM. Wait for the
110  !restart to complete before continuing.
120  OUTPUT  @E1406;"DIAG:BOOT:WARM"
130  ON TIMEOUT 7,.1 GOTO Complete
140  Complete:    B=SPOLL(70900)
150  OFF TIMEOUT 7
160  !
170  !Return the starting address of the table in non-volatile user RAM.
180  OUTPUT  @E1406;"DIAG:NRAM:ADDR?"
190  ENTER  @E1406;A
200  !
210  !Download the following bytes: the table is valid, one module is dynamically
220  !configured, it's installed in slot 6, the logical address of the slot 0 module
230  !is 0, the logical address to be set is 32, and the block size is 1.
240  DATA  257,1,6,0,32,1
250  READ  Dy_config(*)
260  OUTPUT  @E1406 USING    "#,3(K)";"DIAG:DOWN    ";A;" ,#0"
270  OUTPUT  @E1406 USING    "B";Dy_config(*)
280  !
290  !Link the dynamic configuration table to the appropriate algorithm.
300  OUTPUT  @E1406;"VXI:CONF:DCT    ";A
310  !
320  !Restart the system instrument to set the user-defined configuration.
330  OUTPUT  @E1406;"DIAG:BOOT:WARM"
340  END
```

Comments

- Errors associated with dynamic configurations are:

  ERROR 1: FAILED DEVICE
  This error occurs when a dynamically configured device at logical address 255 failed during its power-on sequence.

  ERROR 4: DC DEVICE ADDRESS BLOCK TOO BIG
  This error occurs when the block size specified in the table is greater than 127.

  ERROR 7: DC DEVICE MOVE FAILED
  This error occurs when a dynamically configured device was not set to the logical address specified, possibly due to a hardware failure on the module. The error also occurs when **all** devices in an address block did not move.

  ERROR 9: UNABLE TO MOVE DC DEVICE
  This error occurs when there are not enough successive logical addresses available for the specified block size, or if the logical address specified is already occupied by another static or dynamic module.

  ERROR 39: INVALID UDEF DC TABLE
  This error occurs when the user-defined dynamic configuration table is not true (valid flag does not equal 1).

  ERROR 40: INVALID UDEF DC TABLE DATA
  This error occurs when there are 0, or greater than 254 entries in the user-defined dynamic configuration table.

- The logical addresses assigned by the dynamic configuration table are used by the system until DIAGnostic:BOOT:COLD or VXI:CONFigure:DCTable 0 is executed.

Setting VXI-MXI Configuration

During configuration, ***if an MXI extender device is present*** the resource manager will attempt to assign logical addresses and memory according to the rules listed below. You can override these rules by creating a user-defined extender table. This table will be ignored if there are no MXI extender devices present.

Logical Address Configuration

The following rules and recommendations apply to assigning logical addresses. For a more detailed discussion of how to assign logical addresses please refer to the *HP E1482B VXI-MXI Bus Extender User's Manual*.

- The window of a local extender must include the logical addresses of all remote extenders on its interconnect bus.
- The downward window of a local extender cannot include any devices which are not its descendants, except its own address. It must include all devices on all of its own descendant busses.
- A local extender should have a higher logical address than any statically or dynamically configured devices on its VMEbus (excluding other local extenders).
- A local extender should have a lower logical address than any of its corresponding remote extenders and stand alone devices on its interconnect bus.
- A remote extender should have the lowest logical address on its own VMEbus.
- The logical address of a remote extender can be lower than the address of its corresponding local extender on its interconnect bus.

Default Logical Address Assignments

The resource manager will attempt to assign logical addresses to dynamically configured devices according to the following rules:

- The window for a *local* extender will be set outward to the minimum possible size to include all of the logical addresses found on all of its descendant busses. This includes all stand alone devices and all *remote* extenders that are descendants of the *local* extender.

Note The window for a *local* extender may or may not include the logical address of the *local* extender itself.

- The window for a *remote* extender will be set inward to the minimum possible size to include all of the devices on its VMEbus and all of its descendants.

Note The window for a *remote* extender may or may not include the logical address of the *remote* extender itself.

- A dynamically configured device will be assigned a logical address as follows:
  - Dynamically configured devices on a given VMEbus will be assigned logical addresses after all descendant busses of that VMEbus have been configured.
  - Dynamically configured devices on a given VMEbus will be assigned addresses in the range defined by the statically configured device with the lowest logical address on that VMEbus and the maximum allowable logical address for that VMEbus.
  - Each dynamically configured device will be assigned an address that is a multiple of 8 within the allowable range for that VMEbus until all of these addresses have been used.
  - Any additional dynamically configured devices will be assigned the lowest available addresses within the allowable range for that VMEbus.

A16/A24/A32 Address Window Configuration

The following rules and recommendations apply to assigning A16/A24/A32 logical addresses. Refer to the *HP E1482B VXI-MXI Bus Extender User's Manual* for a more detailed discussion of how to assign logical addresses.

- Systems with multiple VMEbus devices should be configured so that the VMEbus devices in mainframes whose remote extenders have the highest logical addresses should also have the highest logical addresses.
- VMEbus devices should be configured to have the lowest addresses on their particular VMEbus.

Default A16/A24/A32 Address Window Assignments

The resource manager will not attempt to perform any A16 address window configuration as a default. It will attempt to configure A24 and A32 memory according to the following rules:

- A memory page is $1/256$ of the total memory space. The minimum size of an A24 or A32 memory window is 2 pages and the maximum size of the window is 256 pages as defined in *VXI-6 Specifications*. For A24 memory a single page is 65,536 bytes and the minimum window size is 131,072 bytes. For A32 memory a single page is 16,777,216 bytes and the minimum window is 33,554,432 bytes.
- The base address of a memory window must be zero or an even multiple of the size of the window.
- The window for a local extender will be set to the minimum possible size to include all of the memory addresses found on all of its descendants.
- The window for a remote extender will be set to the minimum possible size to include all of the memory on its VMEbus and all of its descendants.
- A VXIbus device will be assigned a memory location in the following manner:
  - VXIbus devices on a given VMEbus will be assigned memory locations after all descendant busses of the VMEbus have been configured.
  - VXIbus devices on a given VMEbus will be assigned memory locations in the range defined by the lowest and highest memory pages available for that bus.
  - The first available page for a VMEbus will be the first page that is higher than any reserved page on any of its ancestors.
  - VXIbus devices will be assigned the lowest memory locations available on the current bus.
  - VXIbus devices will be assigned locations according to memory size and logical address in that order. The device with the largest memory size on a given bus will be assigned an address first. For devices with the same size, the device with the lowest logical address will be assigned a memory location first.
  - If possible, no devices will be assigned to memory locations in the bottom or top $1/8$ of the total memory (for example, in A24 memory addresses 000000_{16} - 200000_{16} or $E00000_{16}$ - $FFFFFF_{16}$).

- VMEbus reserved memory must be placed in locations that will not interfere with windows previously configured. The only way the resource manager can know the location(s) of VMEbus memory is for you to provide this information in the user-defined memory table (see "A24/A32 Address Mapping" on page 44 for more details).

Interrupt Register Configuration

The rules listed below will be used to assign the configuration of the INTX Interrupt Register during system start-up unless you override them with entries in the user-defined extender table.

- The interrupt enable bits in the INTX Interrupt Register on every extender will be enabled for each VMEbus interrupt line that has a VXIbus handler assigned.
- The interrupt enable bits in the INTX Interrupt Register on every extender will be disabled for each VMEbus interrupt line that has no VXIbus handler assigned.
- For every VMEbus interrupt line that has a VXIbus interrupt handler assigned, the direction will be set on each extender such that an interrupt on that line will be routed towards the VMEbus backplane that contains the handler.

TTL Trigger Register Configuration

The TTL Trigger Register will be set to $C0C0_{16}$ (TTL Triggers disabled) for all remote and local extenders that support TTL Triggers. You may enable TTL Triggers and set the TTL Trigger directions with the extender table.

ECL Trigger Register Configuration

The ECL Trigger Register will be set to $C0C0_{16}$ (ECL Triggers disabled) for all remote and local extenders that support ECL Triggers. You can enable ECL Triggers and set the ECL Trigger directions with the extender table.

Utility Register Configuration

The default Utility Register configuration is shown in Table 2-2. Since the resource manager may have to reboot during the system configuration process (for example, to download a driver) the Utility Register is not a part of the extender table. This will help ensure that the SYSRESET signal will propagate throughout the system during a reboot so that all of the cards will receive a hard reset.

If you wish to alter the contents of the Utility Register you can use DIAGnostic:POKE commands directly to the registers. Keep in mind that this may alter the default system reboot process.

Table 2-2. Utility Register Default Configuration

Extender Type	ACFIN	ACFOUT	SFIN	SFOUT	SRIN	SROUT
Local Extender	enabled	enabled	enabled	enabled	enabled	enabled
	(1)	(1)	(1)	(1)	(1)	(1)
Remote Extender	enabled	enabled	enabled	enabled	enabled	enabled
	(1)	(1)	(1)	(1)	(1)	(1)

User-Defined Logical Address and Memory Windows

In many systems that use extenders, the standard boot-up algorithms will not be suitable for your configuration. In such systems it will be necessary to unambiguously define your logical address and memory mapping for the boot-up configuration routine.

The User-Defined Extender Table

You can define your own logical address and memory mapping in a system with extenders by using the user-defined extender table. This table is created as follows:

1. Table space in the command module's non-volatile user RAM is made available by allocating a segment of RAM with the command:

   DIAGnostic:NRAM:CREate <*size*>

2. Reset the command module. NRAM is created during the boot-up process:

   DIAGnostic:BOOT:WARM

3. The location (starting address) of the table in RAM is determined with the command:

   DIAGnostic:NRAM:ADDRess?

4. Data is downloaded into the table with the command:

   DIAGnostic:DOWNload <*address*>, <*data*>

5. The table is linked to the appropriate algorithm in the command module processor with the command:

   VXI:CONFigure:ETABle <*address*>

Table Format

The user-defined extender table consists of a two byte header followed by the required number of extender records. The first byte of the header is a table Valid Flag (1 = valid) and the second byte specifies the number of records in the table.

Table 2-3. Extender Table Format

valid flag (0 | 1)
# of records (N)
extender record 1
extender record 2
•
•
extender record N

Any single item in an extender record can be disabled so that the resource manager will perform the default configuration for the item. For example, to use the resource manager default algorithm for interrupt enable, set the appropriate field in the extender record (see Table 2-4) to 255.

Table 2-4. User-Defined Extender Table Record

Field	Description	Format*	Range	Field Disable Value
1	Logical Address (remote or local extender)	int16	1-255	n/a
2	Logical Address Window Base	int16	0-254 [1]	255
3	Logical Address Window Size	int16	2-256	n/a
4	A16 Memory Base Page	int16	0-254 [1]	255
5	A16 Memory Window Size (number of pages)	int16	2-256	n/a
6	A24 Memory Base Page	int16	0-254 [1]	255
7	A24 Memory Window Size (number of pages)	int16	2-256	n/a
8	A32 Memory Base Page	int16	0-254 [1]	255
9	A32 Memory Window Size (number of pages)	int16	2-256	n/a
10	Interrupt Enable	int16	n/a [2]	255
11	TTL Trigger Enable	int16	n/a [3]	255
12	ECL Trigger Enable	int16	n/a [4]	255

1 The upper byte of this field (bits 15-8) is reserved.

2 This is Mainframe Extender Register 12_{16}. See the *VXI-6 Specification* or your mainframe extender manual for a definition of this register. Interrupts may not be supported by all mainframe extender cards.

3 This is Mainframe Extender Register 14_{16}. See the *VXI-6 Specification* or your mainframe extender manual for a definition of this register. TTL Triggers may not be supported by all mainframe extender cards.

4 This is Mainframe Extender Register 16_{16}. See the *VXI-6 Specification* or your mainframe extender manual for a definition of this register. ECL Triggers may not be supported by all mainframe extender cards.

* int16 is a 16-bit integer, or two bytes.

Determining the Table Size

The user-defined extender table has a one word header and each of the 12 fields is also one word. The amount of RAM allocated with DIAGnostic:NRAM:CREate is specified in bytes. Since one word is two bytes, the amount of RAM to allocate is computed as:

```
2 + 24(N)
```

where N is the number of modules to be configured. For example, to provide information for three extender devices, the table size would be:

```
2 + 24(3) = 74 bytes
```

DIAGnostic:NRAM:CREate would be executed as:

```
OUTPUT  @E1406;"DIAG:NRAM:CRE  74"
```

Data Format

Data can be sent to the extender table in any convenient format, as long as the binary data is preserved. This can be accomplished using DIAGnostic:PEEK? and DIAGnostic:POKE, by reading the data into a variable in the computer and then downloading the data to the table using the Arbitrary Block Program Data format, and so forth. In the following example, this is accomplished by reading the data into 16 bit integer variables in the computer and then downloading the data to the table using the ANSI/IEEE 488.2-1987 Arbitrary Block Program Data format. More information on the Arbitrary Block Program format can be found on page 121 of this manual and in the *ANSI/IEEE 488.2-1987* document.

The table header is sent as a single 16-bit word which must contain the Valid Flag and the number of modules involved. **For a valid table, the header is 256 plus the number of modules.** For example, to indicate a valid table with seven entries, the header is 263 (256 + 7 = 263).

CAUTION **When downloading data into the user-defined extender table, DIAGnostic:DOWNload does not determine if the table is large enough to store the data. If the amount of data sent by DIAGnostic:DOWNload is greater than the table space allocated by DIAGnostic:NRAM:CREate, system errors will occur. You can recover from these errors by executing DIAG:BOOT:COLD, or by pressing the "Ctrl-R" keys on an RS-232 terminal while cycling mainframe power.**

Example: User-Defined Extender Table

This example shows a single interconnect bus with a local extender at logical address 63 in the root mainframe and a remote extender at logical address 64 in the secondary mainframe.

258	valid (upper byte) + 2 records (lower byte)
63	local extender logical address
128	logical address window base
64	logical address window size (128 to 191)
255	specify no A16 memory
0	A16 memory size (ignored)
64	A24 memory base page
64	A24 memory size (pages 64 to 127)
0	A32 memory base page
128	A32 memory size (pages 0 to 127)
257	interrupt line 1 enabled (IN)
769	TTL Triggers (TTL1 OUT, TTL0 IN)
-15936	ECL Triggers ($C1C0_{16}$ = ECL0 enabled OUT)
64	remote extender logical address
128	logical address window base
64	logical address window size (128 to 191)
255	specify no A16 memory
0	A16 memory size (ignored)
64	A24 memory base page
64	A24 memory size (pages 64 to 127)
255	specify no A32 memory
0	A32 memory size (ignored)
256	interrupt line 1 enabled (OUT)
770	TTL Triggers (TTL1 IN, TTL0 OUT)
-15935	ECL Triggers (ECL0 IN)

The program on the next page downloads the table shown above into user non-volatile memory. The program notes each of the steps used to create and load the table.

```
10   !Assign an I/O path and allocate a variable to store MXI configuration
20   !data to be downloaded to the command module.
30   ASSIGN  @E1406 TO 70900;EOL  CHR$(10)  END
40   INTEGER  MXI_config(1:25)
50   !
60   !Allocate a segment of non-volatile user RAM on the command
70   !module to store the user-defined MXI table (1 module).
80   OUTPUT  @E1406;"DIAG:NRAM:CRE  50"
90   !
100  !Restart the system instrument to allocate the user RAM. Wait for the
110  !restart to complete before continuing.
120  OUTPUT  @E1406;"DIAG:BOOT:WARM"
130  ON  TIMEOUT 7,.1 GOTO Complete
140  Complete:      B=SPOLL(70900)
150  OFF TIMEOUT 7
160  !
170  !Return the starting address of the table in non-volatile user RAM.
180  OUTPUT  @E1406;"DIAG:NRAM:ADDR?"
190  ENTER @E1406;A
200  !
210  !Download the required bytes.
220  !See the user-defined extender table for the meaning of these bytes.
230  DATA 258, 63, 128, 64, 255, 0, 64, 64, 0, 128, 257, 769, -15936, 64,
     128, 64, 255, 0, 64, 64, 255, 0, 256, 770, -15935
240  READ  MXI_config(*)
250  OUTPUT  @E1406 USING  "#,3(K)";"DIAG:DOWN  ";A;" ,#0"
260  OUTPUT  @E1406 USING  "W";MXI_config(*)
270  !
280  !Link the user-defined MXI table to the appropriate algorithm.
290  OUTPUT  @E1406;"VXI:CONF:ETAB  ";A
300  !
310  !Restart the system instrument to set the user-defined configuration.
320  OUTPUT  @E1406;"DIAG:BOOT:WARM"
330  END
```

Comments

- The following errors are associated with the extender table or indicate that you may need to create an extender table:

ERROR 50: EXTENDER NOT SLOT 0 DEVICE

This error occurs when a remote VXIbus extender in a remote mainframe is not in slot 0 of its mainframe. The resource manager expects all remote VXIbus extenders to be installed in slot 0 of their mainframe.

ERROR 51: INVALID EXTENDER LADD WINDOW

This error occurs when the configuration routine finds an invalid start address or size for an extender logical address window. You should reconfigure the logical addresses of the VXIbus devices or create a user-defined extender table for the system to override the default algorithm.

ERROR 52: DEVICE OUTSIDE OF LADD WINDOW

This error occurs when a device or devices were found outside the default maximum or outside the user-defined range for the extender. You should reconfigure the logical addresses of the VXIbus devices or create a new extender table for the system to override the default algorithm.

ERROR 53: INVALID EXTENDER A24 WINDOW

This error occurs when the configuration routine finds an invalid start address or size for an extender A24 address window. You should reconfigure the VMEbus memory devices or create a user-defined extender table to override the default algorithm.

ERROR 54: DEVICE OUTSIDE OF A24 WINDOW

This error occurs when an A24 memory device is located outside of the allowable logical address range of an MXIbus extender. You should reconfigure the VMEbus memory devices or create a user-defined extender table to override the default algorithm.

ERROR 55: INVALID EXTENDER A32 WINDOW

This error occurs when the resource manager finds an invalid start address or size for an extender A32 address window. You should reconfigure the VMEbus memory devices or create a user-defined extender table to override the default algorithm.

ERROR 56: DEVICE OUTSIDE OF A32 WINDOW

This error occurs when an A32 memory device is located outside of the allowable logical address range of an MXIbus extender. You should reconfigure the VMEbus memory devices or create a user-defined extender table to override the default algorithm.

ERROR 57: INVALID UDEF LADD WINDOW
This error occurs when a user-defined logical address window violates the *VXI-6 Specification* (has an invalid base or size). You should redefine your extender table with correct values.

ERROR 58: INVALID UDEF A16 WINDOW
This error occurs when a user-defined A16 window violates the *VXI-6 Specification* (has an invalid base or size). You should redefine your extender table with correct values.

ERROR 59: INVALID UDEF A24 WINDOW
This error occurs when a user-defined A24 window violates the *VXI-6 Specification* (has an invalid base or size). You should redefine your extender table with correct values.

ERROR 60: INVALID UDEF A32 WINDOW
This error occurs when a user-defined A32 window violates the *VXI-6 Specification* (has an invalid base or size). You should redefine your extender table with correct values.

ERROR 61 INVALID UDEF EXT TABLE
This error occurs when the valid flag is not set to 1 in the extender table. You should redefine your extender table with correct values.

ERROR 62: INVALID UDEF EXT TABLE DATA
This error occurs when there is an incorrect number of records for a user-defined extender table. You should make sure that the number of records shown in the header matches the number of records actually in the table.

ERROR 63: UNSUPPORTED UDEF TTL TRIGGER
This error occurs when there is a user-defined extender table TTL Trigger entry for a MXIbus extender that does not support TTL Triggers.

ERROR 64: UNSUPPORTED UDEF ECL TRIGGER
This error occurs when there is a user-defined extender table ECL Trigger entry for a MXIbus extender that does not support ECL Triggers.

ERROR 66: INTX CARD NOT INSTALLED
This error occurs when the INTX card is not installed on the VXI-MXI extender. You should make sure the INTX card is correctly installed and that it is functioning.

- The system configuration assigned by the extended device table is used by the system until DIAGnostic:BOOT:COLD or VXI:CONFigure:ETABle 0 is executed.

Setting Commander/Servant Hierarchies

In a VXIbus system, a commander is a plug-in module which controls other plug-in modules. "Control" can be a commander such as the HP E1406A Command Module translating SCPI commands, and/or serving as the HP-IB interface for (servant) modules within its servant area.

During the configuration sequence, the resource manager assigns servant modules to a commander module based on the servants' logical addresses and the commander's servant area. The concept of the servant area is shown in Figure 2-1. The *C-Size VXIbus Systems Configuration Guide* shows how to set the command module's servant area.

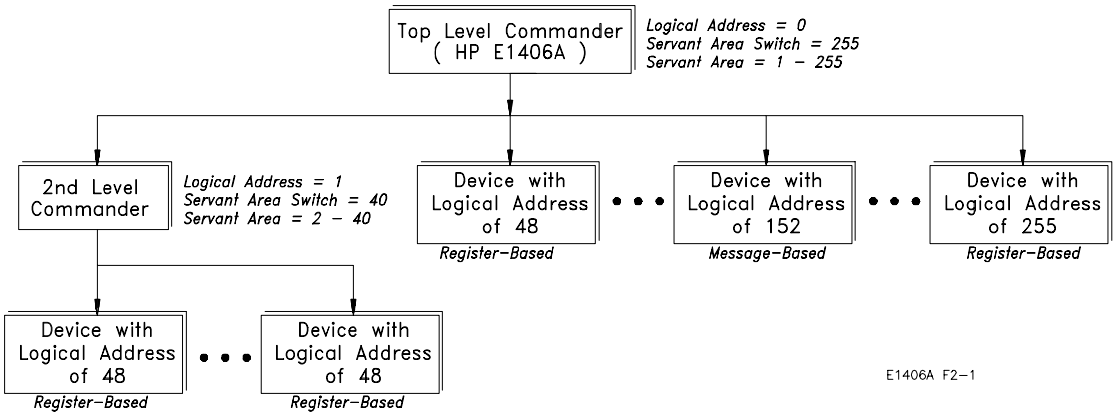


Figure 2-1. Example of Commander/Servant Hierarchy

Note the following regarding commander/servant relationships:

- A commander's servant area is its logical address + 1, through its logical address + its servant area switch setting.
- If within a given commander's servant area (Figure 2-1) there is another lower-level commander(s) (logical address 1), the given commander will control the lower-level commander. However, all modules within the servant area of the lower-level commander (logical addresses 2 - 41) will be controlled by the lower-level commander.
- If there is a commander outside the servant area of the command module/resource manager, that commander becomes a top level commander. The resource manager will assign all modules within the commander's servant area to that commander, or to that commander's lower-level commanders.
- The command module will always be the commander for IBASIC even if IBASIC's logical address (240) is outside the module's servant area. There can be multiple IBASICs in the same system since each is a servant to its respective command module. Note that there are no VXIbus registers for IBASIC.

User-Defined Commander/Servant Hierarchies

In some systems you may need to assign a servant to a commander that is outside the commander's servant area. In other systems, it may be necessary to change a module's secondary HP-IB address, or assign secondary addresses to modules whose logical addresses are not instrument identifiers. These tasks can be accomplished with the user-defined commander/servant hierarchy table described in this section.

Note

Register-based instrument drivers that support multiple card sets normally require that the cards in the set have sequential logical addresses. When instrument drivers support non-sequential logical addresses, instruments that consist of non-sequential card sets must be created using the user-defined commander/servant hierarchy table. There must be an entry in the table for every card in the instrument card set.

The User-Defined Commander/Servant Hierarchy Table

User-defined commander/servant hierarchies and secondary HP-IB addresses are specified with a commander/servant hierarchy table created in the command module. The table is created as follows:

1. Table space in the command module's non-volatile user RAM is made available by allocating a segment of RAM with the command:

   DIAGnostic:NRAM:CREate <*size*>

2. Reset the command module. NRAM is created during the boot-up process:

   DIAGnostic:BOOT:WARM

3. The location (starting address) of the table in RAM is determined with the command:

   DIAGnostic:NRAM:ADDRess?

4. Data is downloaded into the table with the command:

   DIAGnostic:DOWNload <*address*>, <*data*>

5. The table is linked to the appropriate algorithm in the command module processor with the command:

   VXI:CONFigure:CTABle <*address*>

Table Format

The format of the commander/servant hierarchy table is shown in Table 2-5.

Table 2-5. Commander/Servant Hierarchy Table Format

Valid Flag/ Number of Modules		
Laddr	Cmdr Laddr	Sec Addr
Laddr	Cmdr Laddr	Sec Addr
•	•	•
Laddr	Cmdr Laddr	Sec Addr

The table parameters are:

- **Valid Flag (1/0)** 1 indicates the table is valid and the modules should be configured accordingly. 0 (zero) will cause an error message (Error 38). Valid Flag is part of the table header and is represented by the upper eight bits of the header word.
- **Number of Modules (1 - 254)** is the number of entries in the table. Number of Modules is part of the table header and is represented by the lower eight bits of the header word.
- **Laddr** is the logical address of the module which is assigned a new commander or new secondary HP-IB address. Field is one word.
- **Cmdr Laddr** is the logical address of the commander to which the module specified by **Laddr** is assigned. If -1 is specified, the module is not assigned to a commander. Field is one word.
- **Sec Addr (1 - 30)** is the secondary HP-IB address assigned to the module specified by **Laddr**. If -1 is specified, the secondary address is assigned by default. Field is one word.

Determining the Table Size

The commander/servant hierarchy table has a one word header and three one word fields. The amount of RAM allocated with DIAGnostic:NRAM:CREate is specified in bytes. Since one word is two bytes, the amount of RAM to allocate is computed as:

```
2 + 6(N)
```

where N is the number of modules to be configured. For example, to assign three modules to a particular commander, the table size would be:

```
2 + 6(3) = 20 bytes
```

DIAGnostic:NRAM:CREate would be executed as:

```
OUTPUT  @E1406;"DIAG:NRAM:CRE  20"
```

Data Format

Data can be sent to the commander/servant hierarchy table in any convenient format, as long as the binary data is preserved. This can be accomplished using DIAGnostic:PEEK? and DIAGnostic:POKE, by reading the data into a variable in the computer and then downloading the data to the table using the Arbitrary Block Program Data format, and so forth. In the following example, this is accomplished by reading the data into 16 bit integer variables in the computer and then downloading the data to the table using the ANSI/IEEE 488.2-1987 Arbitrary Block Program Data format. More information on the Arbitrary Block Program format can be found on page 121 of this manual and in the *ANSI/IEEE 488.2-1987* document.

The table header is sent as a single 16-bit word which must contain the Valid Flag and the number of modules involved. **For a valid table, the header is 256 plus the number of modules.** For example, to indicate a valid table with seven entries, the header is 263 (256 + 7 = 263).

CAUTION **When downloading data into the commander/servant hierarchy table, DIAGnostic:DOWNload does not determine if the table is large enough to store the data. If the amount of data sent by DIAGnostic:DOWNload is greater than the (table) space allocated by DIAGnostic:NRAM:CREate, system errors will occur. You can recover from these errors by executing DIAGnostic:BOOT:COLD, or by pressing the "Ctrl-R" keys on an RS-232 terminal while cycling mainframe power.**

Example: Assigning a Secondary HP-IB Address

The following program assigns secondary HP-IB address 01 to the HP E1411B 5½-Digit Multimeter at logical address 25. The program notes each of the steps used to create and load the table.

```
10   !Assign an I/O path and allocate a variable to store commander/servant
20   !hierarchy data to be downloaded to the command module.
30   ASSIGN  @E1406 TO 70900;EOL  CHR$(10)  END
40   INTEGER  Cs_hier(1:4)
50   !
60   !Allocate a segment of non-volatile user RAM on the command module
70   !to store the commander/servant hierarchy table.
80   OUTPUT  @E1406;"DIAG:NRAM:CRE  8"
90   !
100  !Restart the system instrument to allocate the user RAM.  Wait for the
110  !restart to complete before continuing.
120  OUTPUT  @E1406;"DIAG:BOOT"
130  ON TIMEOUT 7,.1 GOTO Complete
140  Complete:    B=SPOLL(70900)
150  OFF TIMEOUT 7
160  !
170  !Return the starting address of the table in non-volatile user RAM.
180  OUTPUT  @E1406;"DIAG:NRAM:ADDR?"
190  ENTER  @E1406;A
200  !
210  !Download the following: the table is valid and one module is being
220  !assigned a secondary address, the logical address of the module is 25,
230  !its commander's logical address is 0, the secondary address is 01.
240  DATA  257,25,0,1
250  READ  Cs_hier(*)
260  OUTPUT  @E1406 USING    "#,3(K)";"DIAG:DOWN    ";A;" ,#0"
270  OUTPUT  @E1406 USING    "W";Cs_hier(*)
280  !
290  !Link the commander/servant hierarchy table to the appropriate algorithm.
300  OUTPUT  @E1406;"VXI:CONF:CTAB    ";A
310  !
320  !Restart the system instrument to set the user-defined configuration.
330  OUTPUT  @E1406;"DIAG:BOOT"
340  END
```

Comments

- The following errors are associated with the commander/servant hierarchy table:

  ERROR 12: INVALID UDEF COMMANDER LADD
  This error occurs when the user-defined commander logical address specified in the table (Cmdr Laddr) is not a valid commander. Either the commander does not exist, or it is not a message-based device.

  ERROR 14: INVALID UDEF SECONDARY ADDRESS
  This error occurs when the user-defined secondary address (Sec Addr) is invalid in the commander/servant hierarchy table. Valid secondary addresses are -1, 1 - 30. The error also occurs if the device to which the secondary address is assigned is outside the servant area of the command module.

  ERROR 15: DUPLICATE SECONDARY ADDRESS
  This error occurs when the same secondary address is specified for more than one module in the commander/servant hierarchy table.

  ERROR 18: INVALID COMMANDER LADD
  This error occurs when the commander specified in the user-defined commander/servant hierarchy table is not a valid message-based commander, or the device does not exist.

  ERROR 37: INVALID UDEF CNFG TABLE
  This error occurs when the user-defined commander/servant hierarchy table is not true (valid flag does not equal 1).

  ERROR 38: INVALID UDEF CNFG TABLE DATA
  This error occurs when there are 0 or greater than 254 entries in the user-defined commander/servant hierarchy table.

- The secondary HP-IB addresses (and/or commanders) assigned by the commander/servant hierarchy table are used by the system until DIAGnostic:BOOT:COLD or VXI:CONFigure:CTABle 0 is executed.

A24/A32 Address Mapping

During the configuration sequence, the resource manager reads each VXIbus device's ID Register to determine if the device requires a block of A24 or A32 addresses. Figure 2-2 shows the address mapping concept.

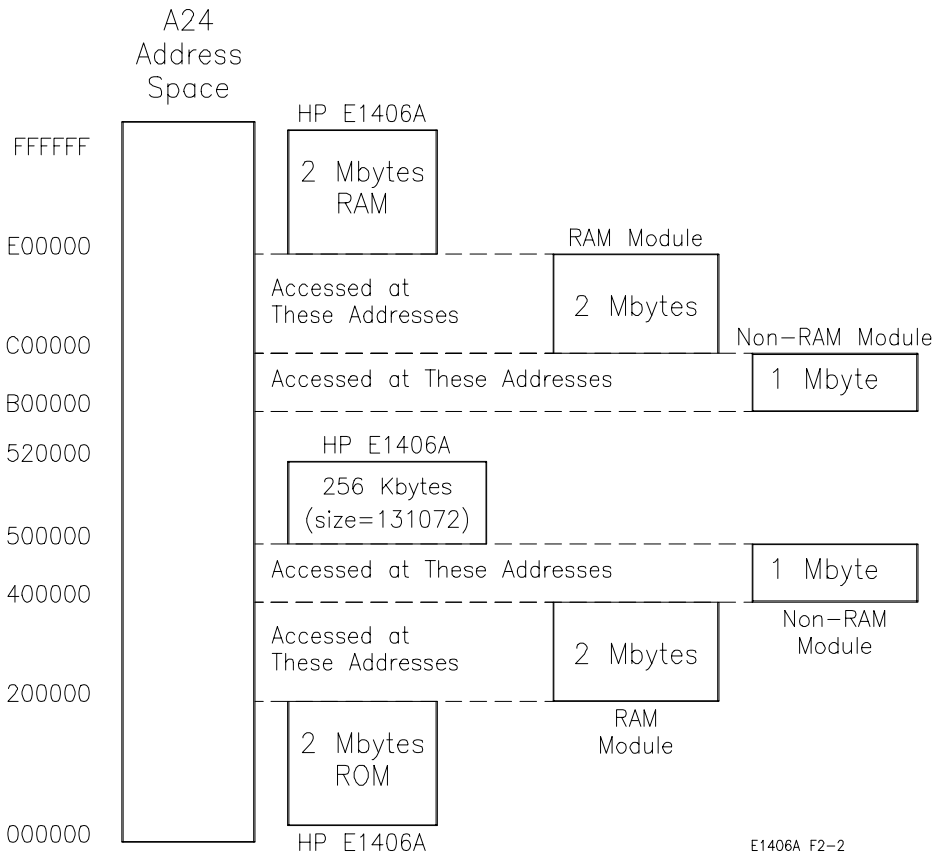


Figure 2-2. A24/A32 Address Mapping Concept

A24/A32 Address Allocation

The resource manager allocates A24 and A32 addresses as follows:

- The top and bottom 2 MB of A24 addresses are used by the command module for its own RAM and ROM.
- VXIbus modules are allocated addresses from the bottom of the address space up.
- The order of address allocation is based on the number of addresses required (memory size) and the logical address. Modules with the largest amount of memory are allocated addresses first. Modules with the same amount of memory are allocated addresses beginning with the lowest logical address.
- The top 2 MB of A24 addresses (used internally by the command module RAM) can be allocated. However, the command module cannot access those addresses on the other device.
- An address allocation table can be used to reserve blocks of A24/A32 addresses for VMEbus devices. This table is also used to assign addresses other than the default addresses assigned by the resource manager.

- A24 address space is 16 MB and A32 address space is 4 GB. The command module does not have A32 address lines and cannot access A32 address space. However, it will allocate A32 address space for devices which can access it. A32 memory allocation is similar to A24 memory allocation.
- A32 address space is 00000000_{16} through $FFFFFFFF_{16}$.

Allocating Address Space for VMEbus Devices

The resource manager (command module) has no way to determine when VMEbus devices have been installed in the system. As a result, the resource manager allocates addresses to VXIbus A24/A32 devices rather than to VMEbus devices.

There are two ways to prevent addresses intended for a VMEbus device from being assigned to VXIbus devices. The first method is described below. The second method uses an address allocation table to "reserve" a block of addresses. The table used for this is described in the section "Reserving A24/A32 Address Space" beginning on page 48.

Allocating Address Space for VMEbus Devices: Method 1

1. Configure and install all modules (except VMEbus devices) in the HP 75000 Series C Mainframe. This process is described in the *C-Size VXIbus Systems Configuration Guide*.

2. Turn on the mainframe and note section 6 of the resource manager's configuration sequence (Figure 2-3).

   Given the starting (offset) A24 addresses assigned to the devices and the size of each device's memory (converted to hexadecimal), the A24 addresses **not allocated** can be determined. For example, in Figure 2-3, the highest offset is 240000_{16} with a size of 20000_{16} (131,072 bytes converted to hexadecimal). Thus, for this system, A24 addresses from 260000_{16} to $DFFFFF_{16}$ are available to VMEbus devices.

Note In systems that include VXI-MXI extenders you should use a table to tell the resource manager where your A24/A32 VMEbus memory is located. The resource manager cannot find VMEbus memory without this table.

Sequence ↓	Display	Explanation
1	Testing ROM Testing 512 KB RAM Passed Testing CPU CPU Self Test Passed Non-volatile Ram Contents Lost HP-IB address: 09 Talk/Listen command module ladd = 0 command module servant area = 255	The HP E1406A operating system performs a series of self-tests and clears its volatile RAM. The command module's HP-IB address, logical address, and servant area (based on the switch settings) are reported.
2	Command Module VMEbus timeout -- ENABLED	The resource manager identifies the status of the command module VMEbus timeout. This must be ENABLED for systems without VXIbus extenders (HP E1406A Command Module HP-IB switch #5 = 0).
3	Searching for static devices in mainframe 0 SC device at ladd 0 in slot 0 SC device at ladd 8 in slot ? SC device in ladd 16 in slot 8 Searching for dynamic devices in mainframe 0 DC device in slot 3 moved to ladd 24, block size = 1	The resource manager identifies all statically configured modules, and then locates and configures all dynamically configurable modules.
4	Searching for pseudo devices	Pseudo devices are instruments such as IBASIC.
5	Configuring Commander/Servant hierarchy ladd = 0, cmdr ladd = -1 ladd = 8, cmdr ladd = 0 ladd = 16, cmdr ladd = 0 ladd = 24, cmdr ladd = 0 ladd = 32, cmdr ladd = 24 ladd = 64, cmdr ladd = 24 Validating Commander/Servant hierarchy Commander ladd 24 granted device ladd 32 Commander ladd 24 granted device ladd 64	The resource manager establishes the VXIbus system's commander/servant hierarchies based on the commander's servant area and the servant's logical address.
6	Mapping A24 Memory ladd 0, offset = 00200000H, size = 131,072 (bytes) ladd 24, offset = 00220000H, size = 131,072 (bytes) ladd 64, offset = 00240000H, size = 131,072 (bytes) Mapping A32 memory in mainframe 0	The resource manager allocates A24 addresses to access the memory located on the modules at logical addresses 0, 24, and 64. The offset is specified in hexadecimal and the size is specified in bytes. In this system, there are no A32 devices.
7	Configuring VME interrupts VME interrupt line 1 assigned to ladd 0, handler ID 1 VME interrupt line 2 assigned to ladd 24, handler ID 1 VME interrupt line 3 assigned to ladd 64, handler ID 1 VME interrupt line 4 - no handler assigned VME interrupt line 5 - no handler assigned VME interrupt line 6 - no handler assigned VME interrupt line 7 - no handler assigned	The resource manager allocates interrupt lines to itself and to the other interrupt handlers in the system.
8	SYSTEM INSTALLED AT SECONDARY ADDR 0 VOLTMTR INSTALLED AT SECONDARY ADDR 1 SWITCH INSTALLED AT SECONDARY ADDR 2 MBinstr INSTALLED AT SECONDARY ADDR 3 SYSTEM instrument started BNO issued to ladd 24, BNO response = FFFE Opening HP-IB access for message-based device at sec addr 03	The resource manager identifies the secondary HP-IB addresses used in the system, starts the system instrument (i.e., command module), issues the Begin Normal Operation (BNO) command to its direct message based servant, and opens HP-IB access to the module at secondary HP-IB address 03.

Figure 2-3. Resource Manager Configuration Without Extenders

Sequence	Display	Explanation
1	Testing ROM Testing 512 KB RAM Passed Testing CPU CPU Self Test Passed Non-volatile Ram Contents Lost HP-IB address: 09 Talk/Listen command module ladd = 0 command module servant area = 255	The HP E1406A operating system performs a series of self-tests and clears its volatile RAM. The command module's HP-IB address, logical address, and servant area (based on the switch settings) are reported.
2	Command Module VMEbus timeout -- DISABLED	The resource manager identifies the status of the command module VMEbus timeout. This must be DISABLED for systems without VXIbus extenders (HP E1406A Command Module HP-IB switch #5 = 0).
3	Searching for static devices in mainframe 0 SC device at ladd 0 in slot 0 SC device at ladd 8 in slot ? SC device in ladd 16 in slot 8 SC device at ladd 127 in slot 5 -- VXIbus extender Searching for static devices on interconnect bus 127 SC device at ladd 128 in slot 0 -- VXIbus extender Searching for static devices in mainframe 128 SC device at ladd 144 in slot 7 Searching for dynamic devices in mainframe 128 DC device in slot 3 moved to ladd 136, block size = 1 VXIbus extender 128 Ladd window range: 128 to 159, INWARD VXIbus extender 127 Ladd window range: 128 to 159, OUTWARD Searching for dynamic devices in mainframe 0 DC device in slot 3 moved to ladd 24, block size = 1	The resource manager identifies all statically configured modules, and then locates and configures all dynamically configurable modules.
4	Searching for pseudo devices	Pseudo devices are instruments such as IBASIC.
5	Configuring Commander/Servant hierarchy ladd = 0, cmdr ladd = -1 ladd = 8, cmdr ladd = 0 ladd = 16, cmdr ladd = 0 ladd = 24, cmdr ladd = 0 ladd = 136, cmdr ladd = 0 ladd = 144, cmdr ladd = 0 Validating Commander/Servant hierarchy Commander ladd 24 granted device ladd 32 Commander ladd 24 granted device ladd 64	The resource manager establishes the VXIbus system's commander/servant hierarchies based on the commander's servant area and the servant's logical address.
6	Mapping A24 Memory Searching for A24 memory in mainframe 128 VXIbus extender 128 A24 window range: 00000000 to 00FFFFFF, OUTWARD VXIbus extender 127 A24 window range: 00000000 to 00FFFFFF, INWARD Searching for A24 memory in mainframe 0 ladd 0, offset = 00200000H, size = 131,072 (bytes) Mapping A32 memory Searching for A32 memory in mainframe 128 VXIbus extender 128 A32 window range: 00000000 to FFFFFFFF, OUTWARD VXIbus extender 127 A32 window range: 00000000 to FFFFFFFF, INWARD Searching for A32 memory in mainframe 0	The resource manager allocates A24 addresses to access the memory located on the modules at logical addresses 0, 24, and 64. The offset is specified in hexadecimal and the size is specified in bytes. In this system, there are no A32 devices.
7	Configuring VME interrupts VME interrupt line 1 assigned to ladd 0, handler ID 1 VME interrupt line 2 assigned to ladd 24, handler ID 1 VME interrupt line 3 assigned to ladd 64, handler ID 1 VME interrupt line 4 - no handler assigned VME interrupt line 5 - no handler assigned VME interrupt line 6 - no handler assigned VME interrupt line 7 - no handler assigned VXIbus extender 128 interrupts: 1-OUT 2-DIS 3-DIS 4-DIS 5-DIS 6-DIS 7-DIS VXIbus extender 128 interrupts: 1-IN 2-DIS 3-DIS 4-DIS 5-DIS 6-DIS 7-DIS	The resource manager allocates interrupt lines to itself and to the other interrupt handlers in the system.
8	SYSTEM INSTALLED AT SECONDARY ADDR 0 VOLTMTR INSTALLED AT SECONDARY ADDR 1 SWITCH INSTALLED AT SECONDARY ADDR 2 MBinstr INSTALLED AT SECONDARY ADDR 3 SYSTEM instrument started BNO issued to ladd 24, BNO response = FFFE Opening HP-IB access for message based device at sec addr 03	The resource manager identifies the secondary HP-IB addresses used in the system, starts the system instrument (i.e., command module), issues the Begin Normal Operation (BNO) command to its direct message based servant, and opens HP-IB access to the module at secondary HP-IB address 03.

Figure 2-4. Resource Manager Configuration With Extenders

Reserving A24/A32 Address Space

As previously mentioned, the resource manager cannot determine when VME devices have been installed in the system. To prevent the resource manager from allocating A24/A32 addresses intended for VME devices to VXIbus devices, the address allocation table is used. The A24/A32 address allocation table is also used to assign different addresses to VXIbus devices other than those (default) addresses assigned by the resource manager during power-on.

The A24/A32 Address Allocation Table

The A24/A32 address allocation table is created and stored in the command module as follows:

1. Table space in the command module's non-volatile user RAM is made available by allocating a segment of RAM with the command:

   DIAGnostic:NRAM:CREate <*size*>

2. Reset the command module. NRAM is created during the boot-up process:

   DIAGnostic:BOOT:WARM

3. The location (starting address) of the table in RAM is determined with the command:

   DIAGnostic:NRAM:ADDRess?

4. Data is downloaded into the table with the command:

   DIAGnostic:DOWNload <*address*>, <*data*>

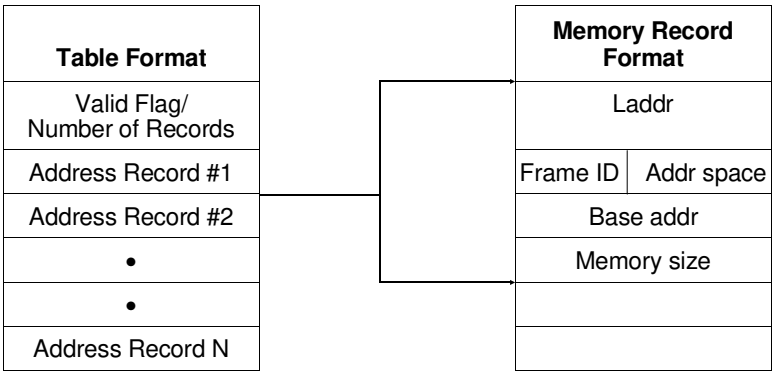
5. The table is linked to the appropriate algorithm in the command module processor with the command:

   VXI:CONFigure:MTABle <*address*>

Table Format

The format of the A24/A32 address allocation table is shown in Table 2-6.

Table 2-6. A24/A32 Address Allocation Table Format



The table parameters are:

- **Valid Flag (0/1)** 1 (one) indicates the table is valid and the addresses reserved accordingly. 0 (zero) will cause an error message (Error 43). Valid Flag is part of the table header and is represented by the upper eight bits of the header word.
- **Number of Records** is the number of address records in the table. You must have one record for each VMEbus or VXIbus device for which memory is reserved. Number of Records is part of the table header and is represented by the lower eight bits of the header word.
- **Laddr** is the logical address of the VXIbus device for which A24/A32 addresses are reserved. -1 specifies a VMEbus device. Field is one word.
- **Addr space (24|32)** is the address space being reserved. 24 specifies A24 addresses are being reserved. 32 specifies A32 addresses are being reserved. Field is one word.
- **Frame ID (0-255)** is the logical address of the slot 0 device for the mainframe containing the VMEbus memory block (8-bit byte). This field must be included.
- **Base addr (0 to 2^{24}-1/ 0 to 2^{32}-1)** is the starting address (offset) of the A24 or A32 addresses to be reserved. Field is two words (4 bytes) and is specified in decimal.
- **Memory size (1 to 2^{24}-1/ 1 to 2^{32}-1)** is the amount of memory for which addresses must be reserved. This field must be specified but is ignored if a VXIbus A24/A32 device is specified (**Laddr**). Field is two words (4 bytes) and is specified in decimal.

Determining the Table Size

The A24/A32 address allocation table has a one word header, the first two entries in the address record are one word each, and the second two entries are two words each. The amount of RAM allocated with DIAGnostic:NRAM:CREate is specified in bytes. Since one word is two bytes, the amount of RAM to allocate is computed as:

```
2 + 12(N)
```

where 2 is the two byte header, 12 is the number of bytes per address record (2+2+4+4), and N is the number of address records. For example, to reserve A24 addresses for two VMEbus devices, the table size would be:
2 + 12(2) = 26 bytes. DIAGnostic:NRAM:CREate would be executed as:

```
OUTPUT  @E1406;"DIAG:NRAM:CRE  26"
```

Data Format

Data can be sent to the A24/A32 address allocation table in any convenient format, as long as the binary data is preserved. This can be accomplished using DIAGnostic:PEEK? and DIAGnostic:POKE, by reading the data into a variable in the computer and then downloading the data to the table using the Arbitrary Block Program Data format, and so forth. In the next example, this is accomplished by reading the data into 16-bit integer variables in the computer and then downloading the data to the table using the ANSI/IEEE 488.2-1987 Arbitrary Block Program Data format. More information on the Arbitrary Block Program format can be found on page 121 of this manual and in the *ANSI/IEEE 488.2-1987* document.

The Table Header

The table header is sent as a single 16-bit word which must contain the Valid Flag and the number of address records. **For a valid table, the header is 256 plus the number of records.** For example, to indicate a valid table with two records, the header is 258 (256 + 2).

CAUTION **When downloading data into the A24/A32 address allocation table, DIAGnostic:DOWNload does not determine if the table is large enough to store the data. If the amount of data sent by DIAGnostic:DOWNload is greater than the (table) space allocated by DIAGnostic:NRAM:CREate, system errors will occur. You can recover from these errors by executing DIAGnostic:BOOT:COLD or by pressing the "Ctrl-R" keys on an RS-232 terminal while cycling mainframe power.**

Example: Reserving A24 Addresses for a VMEbus Device

The following program reserves a block of A24 addresses for a VMEbus device. The program assumes the device has been configured with a starting A24 address of 300000_{16} and a size of 80000_{16}.

Again, this procedure is used when you want to reserve a specific block of A24/A32 addresses for a VMEbus device, or when you want to assign addresses to a VXIbus device that are different from those assigned by the resource manager.

```
 10   !Assign I/O path and allocate variable to store A24/A32 memory
 20   !allocation data to be downloaded to the command module.
 30   ASSIGN  @E1406 TO 70900;EOL    CHR$(10)    END
 40   INTEGER  Mem_alloc(1:7)
 50   !
 60   !Allocate a segment of non-volatile user RAM on the command
 70   !module to store the A24/A32 memory allocation table.
 80   OUTPUT  @E1406;"DIAG:NRAM:CRE  14"
 90   !
100   !Restart the system instrument to allocate the user RAM. Wait for the
110   !restart to complete before continuing.
120   OUTPUT  @E1406;"DIAG:BOOT:WARM"
130   ON TIMEOUT 7,.1 GOTO Complete
140   Complete:    B=SPOLL(70900)
150   OFF TIMEOUT 7
160   !
170   !Return the starting address of the table in non-volatile user RAM.
180   OUTPUT  @E1406;"DIAG:NRAM:ADDR?"
190   ENTER  @E1406;A
200   !
210   !Download the following: the table is valid, there is one memory
220   !record: logical address is -1 (VME card), A24 address space (24)
230   !base address is 300000h (48,0), and memory size is 80000h (8,0).
240   !See Comments.
250   DATA  257,-1,24,48,0,8,0
260   READ  Mem_alloc(*)
270   OUTPUT  @E1406 USING    "#,3(K)";"DIAG:DOWN    ";A;" ,#0"
280   OUTPUT  @E1406 USING    "W";Mem_alloc(*)
290   !
300   !Link the A24/A32 memory allocation table to the appropriate algorithm.
310   OUTPUT  @E1406;"VXI:CONF:MTAB    ";A
320   !
330   !Restart the system instrument to set the user-defined configuration.
340   OUTPUT  @E1406;"DIAG:BOOT:WARM"
350   END
```

Comments

- To download the base address and memory size (line 270) they must each be specified as two 16-bit words (line 250). This can be accomplished as follows:

Memory Size: 300000_{16} =	0030	0000
	1st word	2nd word
	48_{10}	0_{10}
Memory Size: 80000_{16} =	0008	0000
	1st word	2nd word
	8_{10}	0_{10}

- The following errors are associated with the A24/A32 address allocation table:

ERROR 8: INACCESSIBLE A24 MEMORY
This error occurs when all or part of an A24 device overlaps the top 2 MB or bottom 2 MB of the A24 address space. This space becomes inaccessible to the command module.

ERROR 32: INACCESSIBLE A32 MEMORY
This error occurs when all or part of an A32 device overlaps the top 500 MB or bottom 500 MB of the A32 address space.

ERROR 33: INVALID UDEF MEMORY BLOCK
This error occurs when an invalid base address is specified, or when the size of the memory exceeds the A24 or A32 address space (given the base address specified).

ERROR 34: UDEF MEMORY BLOCK UNAVAILABLE
This error occurs when the memory block specified in the A24/A32 address allocation table has already been assigned. Also, in a system with VXI-MXI extenders, A24/A32 window restrictions may force some addresses to be unavailable on a given VMEbus.

ERROR 35: INVALID UDEF ADDRESS SPACE
This error occurs when the address space (Addr space) specified in the table is A24 and an A32 device is installed, or vice versa.

ERROR 36: DUPLICATE UDEF MEMORY LADD
This error occurs when a logical address is specified more than once in the same A24/A32 address allocation table. This does not apply to VMEbus devices (address = -1).

ERROR 43: INVALID UDEF MEM TABLE
This error occurs when the user-defined A24/A32 address allocation table is not true (valid flag does not equal 1).

ERROR 44: INVALID UDEF MEM TABLE DATA
This error occurs when an invalid logical address is specified in the A24/A32 address allocation table.

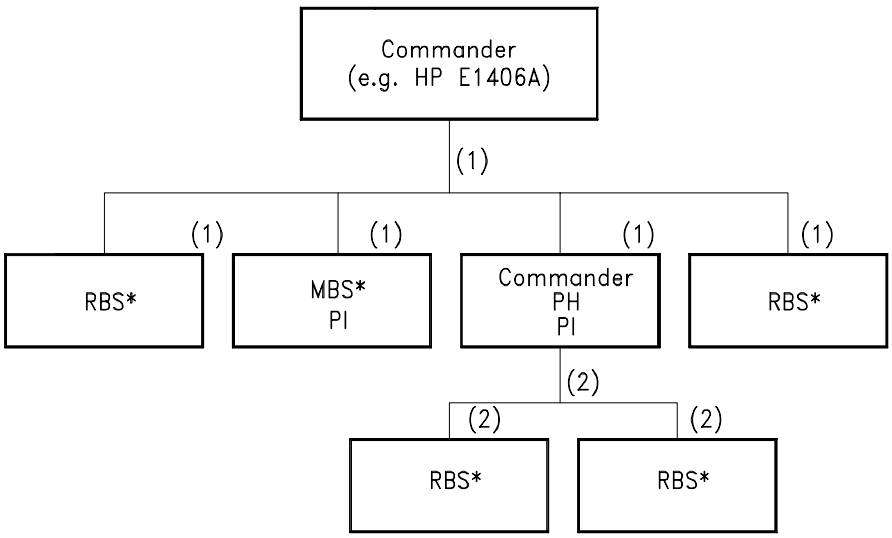
- The A24/A32 addresses reserved by the A24/A32 address allocation table are reserved within the system until DIAGnostic:BOOT:COLD or VXI:CONFigure:MTABle 0 is executed.

Interrupt Line Allocation

In a VXIbus system, communication and coordination between a commander module and its servant module(s) is often achieved using the VXIbus backplane interrupt lines. During the configuration sequence, the resource manager assigns interrupt lines to programmable interrupt handler modules and interrupter modules.

Both commanders and servants can be interrupt handlers and/or interrupters. The command module which is a programmable interrupt handler, is not an interrupter. Thus, in systems where the command module is a servant to another commander, it communicates with the commander through its Response and Data Low Registers (see the *VXIbus System Specification*).

The assignment and use of the interrupt lines is described in Figure 2-5 and with the information which follows.



* Modules which are not programmable interrupters must select the interrupt line using the jumpers on the module.

MBS – Message Based Servant
RBS – Register Based Servant
PI – Programmable Interrupter
(1) – Interrupt Line 1
(2) – Interrupt Line 2

Figure 2-5. Example of Interrupt Line Allocation

Note the following regarding interrupt line allocation:

- There are seven VXIbus backplane interrupt lines. As the resource manager, the HP E1406A Command Module assigns itself interrupt line 1 (default). Additional interrupt lines (up to all seven) can be assigned to the command module using the interrupt line allocation table. Interrupt lines not assigned to programmable handlers remain unassigned.
- Many Hewlett-Packard modules have interrupt line 1 as their factory setting. Thus, they are available for immediate use with the HP E1406A Command Module.
- Commander modules which are programmable interrupt handlers are assigned interrupt lines 2, 3, 4,...7; beginning with the commander with the lowest logical address. Only one interrupt line is assigned per interrupt handler.
- Servant modules which are programmable interrupt handlers are also assigned interrupt lines, beginning with the servant with the lowest logical address. Only one interrupt line is assigned per interrupt handler.
- Servant modules which are programmable interrupters are assigned the same interrupt line assigned to their commander.
- For modules which are not programmable, the interrupt line is selected using jumpers on the modules. The interrupt line allocation table is used to tell the command module which line was selected.

User-Defined Interrupt Line Allocation Table

The interrupt line allocation table allows you to assign additional interrupt lines to a specific handler, reserve interrupt lines for non-programmable interrupt handlers and interrupters, and assign lines to VMEbus devices.

The Interrupt Line Allocation Table

User-defined interrupt line allocations are specified with an interrupt line table created in the command module. The table is created as follows:

1. Table space in the command module's non-volatile user RAM is made available by allocating a segment of RAM with the command:

   DIAGnostic:NRAM:CREate <*size*>

2. Reset the command module. NRAM is created during the boot-up process:

   DIAGnostic:BOOT:WARM

3. The location (starting address) of the table in RAM is determined with the command:

   DIAGnostic:NRAM:ADDRess?

4. Data is downloaded into the table with the command:

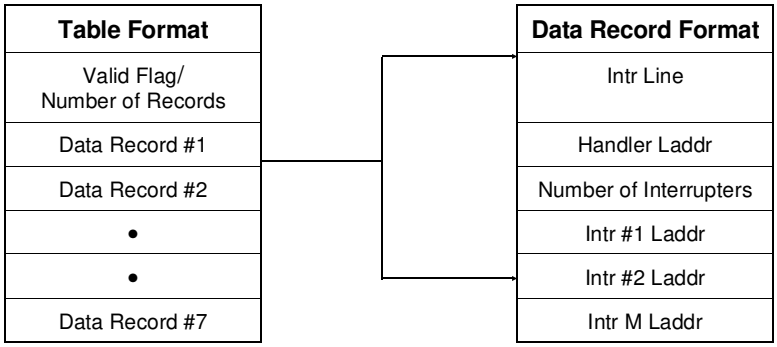
   DIAGnostic:DOWNload <*address*>, <*data*>

5. The table is linked to the appropriate algorithm in the command module processor with the command:

   VXI:CONFigure:ITABle <*address*>

Table Format The format of the interrupt line table is shown in Table 2-7.

Table 2-7. Interrupt Line Allocation Table Format



The table parameters are:

- **Valid Flag (1/0)** 1 (one) indicates the table is valid and the modules should be configured accordingly. 0 (zero) will cause an error message (Error 41). Valid Flag is part of the table header and is represented by the upper eight bits of the header word.
- **Number of Records (1 - 7)** is the number of data records in the table. A data record is required for each interrupt line assigned. Number of Records is part of the table header and is represented by the lower eight bits of the header word.
- **Intr Line (1 - 7)** is the interrupt line to be assigned to the programmable interrupt handler or interrupter, or the line reserved for a non-programmable interrupter/handler or VMEbus device. Field is one word.
- **Handler Laddr** is the logical address of the programmable handler which will handle interrupts on the line specified by **Intr Line**. If -1 is specified, the line is reserved and no handler is assigned. The field is one word.
- **Number of Interrupters** is the number of programmable interrupters on the interrupt line specified by **Intr Line**. If 0 is specified, there are no programmable interrupters. This reserves the line for a non-programmable interrupter. The field is one word.
- **Intr Laddr** is the logical address of the programmable interrupter on the interrupt line specified. The logical address of each programmable interrupter on the line must be specified. Programmable interrupters can be assigned to interrupt lines with no handler. This allows a programmable interrupter to have a non-programmable interrupt handler handle its interrupts. If **Number of Interrupters** is 0, **Intr Laddr** is not specified.

Determining the Table Size

The interrupt line allocation table has a one word header and each data record contains three words, plus one word for each programmable interrupter logical address specified. The amount of RAM allocated with DIAGnostic:NRAM:CREate is specified in bytes. Since one word is two bytes, the amount of RAM to allocate is computed as:

$$2 + 6(N) + 2\sum_{0}^{N} M$$

where 2 is the two byte header, 6 is the number of bytes/data record, N is the number of data records (for example, interrupt lines) and M is the number of programmable interrupters per data record. For example, to create a table for the following:

- one interrupt handler
- two interrupt lines
- one interrupter on one line, three interrupters on second line

the table size would be:

```
2 + 6(2)  +  2(4) = 22 bytes
      |         |
(2 records)  (4 interrupters)
```

DIAGnostic:NRAM:CREate would be executed as:

```
OUTPUT  @E1406;"DIAG:NRAM:CRE  22"
```

Note

When assigning an additional interrupt line to an interrupt handler, you must specify each line. Otherwise, the table will overwrite the line currently assigned, giving the handler only one line. For example, if the resource manager assigns interrupt line 2 to a handler and you want to also assign line 3 to the handler, lines 2 and 3 must be specified in the table. See "Example: Assigning an Interrupt Line" on page 57.

Data Format

Data can be sent to the interrupt line allocation table in any convenient format, as long as the binary data is preserved. This can be accomplished using DIAGnostic:PEEK? and DIAGnostic:POKE, by reading the data into a variable in the computer and then downloading the data to the table using the Arbitrary Block Program Data format, and so forth. In the following example, this is accomplished by reading the data into 16 bit integer variables in the computer and then downloading the data to the table using the ANSI/IEEE 488.2-1987 Arbitrary Block Program Data format. More information on the Arbitrary Block Program format can be found on page 121 of this manual and in the *ANSI/IEEE 488.2-1987* document.

The table header is sent as a single 16-bit word which must contain the Valid Flag and the number of data records. **For a valid table, the header is 256 plus the number of data records.** For example, to indicate a valid table with one data record, the header is 257 (256 + 1 = 257).

CAUTION **When downloading data into the interrupt line allocation table, DIAGnostic:DOWNload does not determine if the table is large enough to store the data. If the amount of data sent by DIAGnostic:DOWNload is greater than the (table) space allocated by DIAGnostic:NRAM:CREate, system errors will occur. You can recover from these errors by executing DIAGnostic:BOOT:COLD, or by pressing the "Ctrl-R" keys on an RS-232 terminal while cycling mainframe power.**

Example: Assigning an Interrupt Line

The following example shows how an additional interrupt line is assigned to a programmable interrupt handler and reserved for a non-programmable interrupter (Figure 2-6).

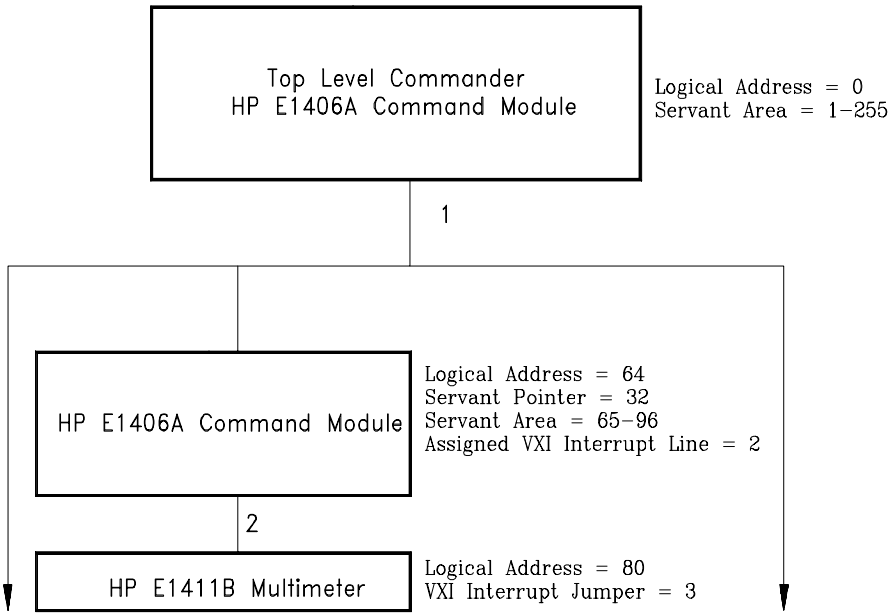


Figure 2-6. Assigning an Additional Interrupt Line

The program assumes that a VXIbus system contains an HP E1411B 5½-Digit Multimeter that is a servant to a second HP E1406A Command Module at logical address 64. Since the command module is the only other commander and is a programmable interrupt handler, it is assigned interrupt line 2 by the resource manager. The HP E1411B, however, has its interrupt jumper set for line 3. For the multimeter to communicate with the command module, the command module must also be assigned to handle interrupt line 3.

```
10   !Assign an I/O path and allocate a variable to store interrupt line
20   !data to be downloaded to the command module.
30   ASSIGN  @E1406 TO 70900;EOL    CHR$(10)    END
40   INTEGER  Intr_line(1:7)
50   !
60   !Allocate a segment of non-volatile user RAM on the command module
70   !to store the interrupt line table (2 data records, no interrupters).
80   OUTPUT  @E1406;"DIAG:NRAM:CRE  14"
90   !
100  !Restart the system instrument to define the user RAM. Wait for the
110  !restart to complete before continuing.
120  OUTPUT  @E1406;"DIAG:BOOT"
130  ON TIMEOUT 7,.1 GOTO Complete
140  Complete:    B=SPOLL(70900)
150  OFF TIMEOUT 7
160  !
170  !Return the starting address of the non-volatile user RAM.
180  OUTPUT  @E1406;"DIAG:NRAM:ADDR?"
190  ENTER  @E1406;A
200  !
210  !Download the following: the table is valid - there are two data records.
220  !Interrupt line 3 (and line 2) is assigned to the handler at logical address 64.
230  !There are no programmable interrupters on either line.
240  DATA  258,2,64,0
250  DATA  3,64,0
260  READ  Intr_line(*)
270  OUTPUT  @E1406 USING    "#,3(K)";"DIAG:DOWN    ";A;" ,#0"
280  OUTPUT  @E1406 USING    "W";Intr_line(*)
290  !
300  !Link the interrupt line table to the appropriate algorithm.
310  OUTPUT  @E1406;"VXI:CONF:ITAB    ";A
320  !
330  !Restart the system instrument to set the user-defined configuration.
340  OUTPUT  @E1406;"DIAG:BOOT"
350  END
```

Comments

- Although interrupt line 2 was assigned to the command module at logical address 64 by the resource manager, the line must be "re-assigned" when line 3 is assigned. Otherwise, line 3 will be assigned in place of line 2.
- The interrupt lines assigned by the interrupt line table are used by the system until DIAGnostic:BOOT:COLD is executed.
- When using multiple command modules, HP-IB cables must be connected from the slot 0 command module, to each command module in the system.

- In this program, the command module at logical address 64 has a primary HP-IB address of 08. It has a servant pointer setting of 32, thus its servant area is from logical address 65 to logical address 96. If the HP E1411B multimeter has a logical address of 80, its secondary HP-IB address is 10. Thus, when programming this multimeter, its HP-IB address is:

  ```
  OUTPUT 70810;"....
  ```

  When programming this command module, its HP-IB address is:

  ```
  OUTPUT 70800;"...
  ```

- The following errors are associated with the Interrupt Line Allocation table:

  ERROR 24: INTERRUPT LINE UNAVAILABLE
  This error occurs when an interrupt line assigned by the user-defined interrupt line allocation table is not available. Either the line has already been assigned or has been reserved. This error also occurs if the line being assigned to an interrupter is not handled by the interrupter's commander.

  ERROR 25: INVALID UDEF HANDLER
  This error occurs when the logical address specified in the user-defined interrupt line allocation table for the interrupt handler (Handler Laddr) is a device that is not a valid interrupt handler.

  ERROR 26: INVALID UDEF INTERRUPTER
  This error occurs when the logical address specified in the user-defined interrupt line allocation table for the interrupter (Intr # Laddr) is a device that is not a valid interrupter.

  ERROR 41: INVALID UDEF INTR TABLE
  This error occurs when the user-defined interrupt line allocation table is not true (valid flag does not equal 1).

  ERROR 42: INVALID UDEF INTR TABLE DATA
  This error occurs when the user-defined interrupt line allocation table has invalid data; the number of records and/or the interrupt line specified is less than 1 or greater than 7, or there is an invalid interrupt handler and/or interrupter logical address (valid addresses are 0 to 255).

- The interrupts assigned by the interrupt line allocation table are used by the system until DIAGnostic:BOOT:COLD or VXI:CONFigure:ITABle 0 is executed.

Starting System Operation

The resource manager completes the configuration sequence by issuing the "Begin Normal Operation" (BNO) command to all top level commanders and to each of its direct message based servants. BNO is not sent to register based modules. The module receiving BNO responds by writing its status to the Data Low Register which is read by the resource manager. More information on BNO and on the Data Low Register can be found in the *VXIbus System Specification*.

If the command module is in a system where it is not the resource manager, it sends BNO to each of its message based servants once it receives BNO from its commander.

VXI SYSFAIL* Line

One of the signals on the VXI backplane is SYSFAIL*. This signal is intended to indicate that some VXI module in the system has failed. During power-on or rebooting the HP E1406A, VXI modules may briefly generate the SYSFAIL* signal. VXI modules that fail to operate may continue to generate SYSFAIL* after the power-on period as an indication of the failure. Similarly, modules that fail during operation of the system may also generate SYSFAIL* when the failure occurs.

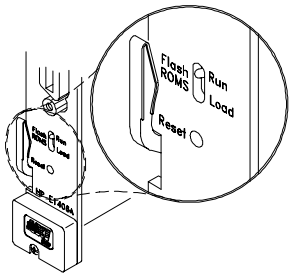
If the HP E1406A Command Module detects the SYSFAIL* after the power-on period, it will automatically reboot. When this occurs, the command module will not enable communication with any of the VXI modules in the system. This is because the HP E1406A cannot determine which VXI module has failed. Also, if IBASIC is installed, it will be disabled. Only the System instrument will be enabled. This behavior is intended to guarantee that you will recognize that a failure has occurred. If this situation occurs, the SYSTem:ERRor? query will return the `Error +2129, "Warning, Sysfail detected"`.

To restore normal operation of the HP E1406A Command Module, you must determine which VXI module has failed and remove it from the system. After removing the failed module and cycling power on your VXI mainframe, your HP E1406A Command Module will work normally.

Chapter 3
Using the Display Terminal Interface

About This Chapter

This chapter shows you how to use the HP E1406A Command Module's display terminal interface to operate instruments in a C-Size mainframe when the Flash ROMS Run/Load switch is set to its "Run" position. The instruments (including the System instrument) are disabled when the Flash ROMS Run/Load switch is in the "Load" position.



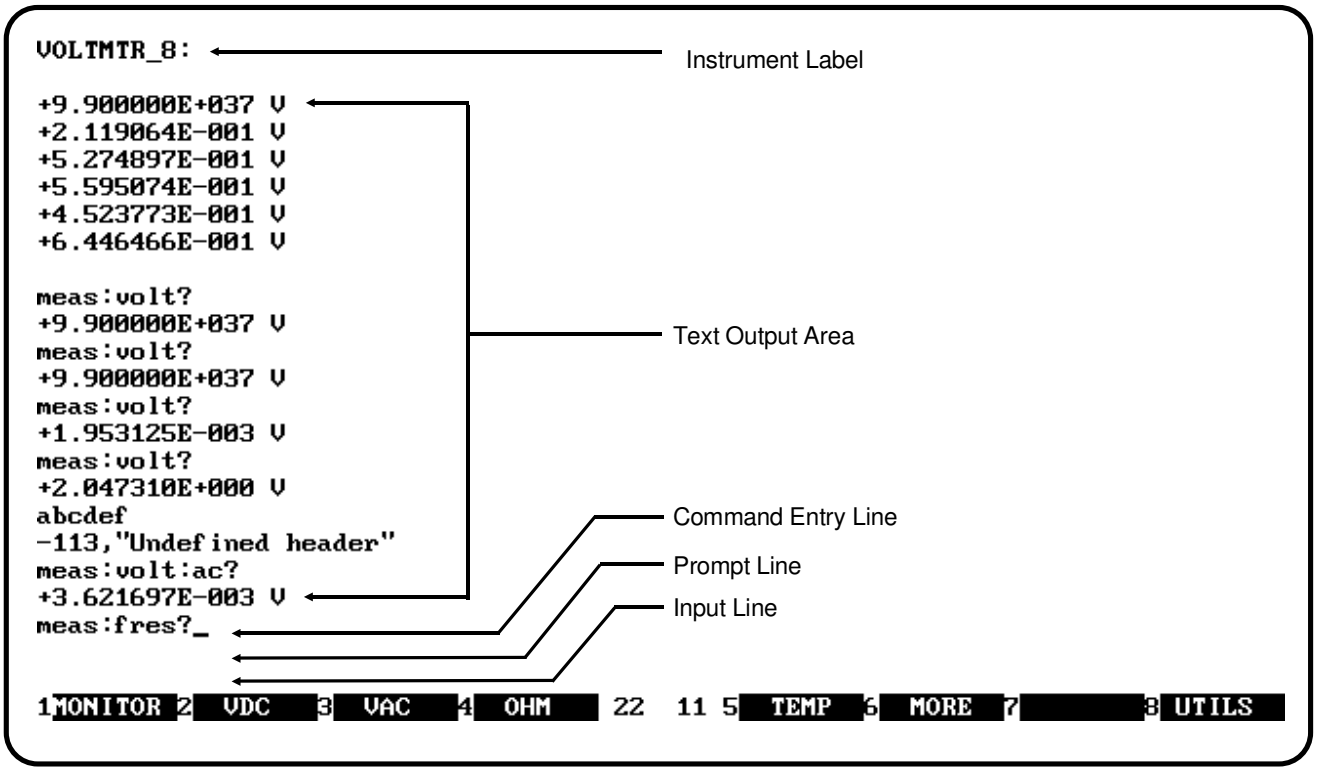
In this position, a special Loader instrument is present, and will let you download drivers or a new operating system to Flash ROM. The terminal interface uses the built-in RS-232 port and/or the optional HP E1324A RS-232C/422 Terminal Interface for Command Modules to provide a front panel for C-size VXIbus systems.

The main sections of this chapter include:

Note This chapter discusses ***using*** the display terminal interface. It assumes you have already connected your terminal and configured it to communicate with the command module. For information on connecting and configuring your terminal, see the *C-Size VXIbus Systems Configuration Guide*.

Terminal Interface Features

Figure 3-1 shows a typical terminal interface display with its function labels across the bottom of the screen. The first five function keys (**f1** through **f5**) select instrument menu choices. Function keys **f6** through **f8** provide menu control and access to utility functions. The tutorials in this chapter show how to use most of the menu control and utility function keys. See "General Key Descriptions" on page 77 for a complete description of each of these key functions.



Notes:
1. Example screens are from HP AdvanceLink terminal emulator.
2. Later screen examples are shown compressed (only 4 lines high) and may show only part of the screen width.

Figure 3-1. Typical Terminal Interface Display

Using Display Terminal Menus

A System instrument menu and a variety of other instrument menus (depending on the instruments in the command module servant area) are available from the terminal interface. These menus incorporate the most used functions but do not provide access to the complete functionality of an instrument. If a particular function is not available from a menu, you can type the corresponding common command or SCPI command string and execute it from the terminal interface. See "Executing Commands" on page 76 for more information.

When you select an instrument, you are assigning the terminal interface to that instrument. This means that any menu operations, commands executed or recalled, errors displayed, and so forth pertain only to that instrument. Terminal interface operation of an instrument is independent from other instruments and independent from the remote operation of the instrument. To operate another instrument from the terminal interface, you must select that instrument.

How Instruments Appear in the Menu

Instruments in the terminal interface menu are register-based devices which are in the servant area of the command module. **Message-based devices, or register-based devices outside the command module's servant area, do not appear in the menu.**

Note

Message-based instruments, which do not appear in instrument menus, can be programmed using the SYSTEM instrument menu. See "Using the System Instrument Menu" on page 65.

Multiple Command Modules

In systems with multiple command modules, the instruments in the menu depend on the command module whose RS-232 port is connected to the terminal. To change menus (command modules):

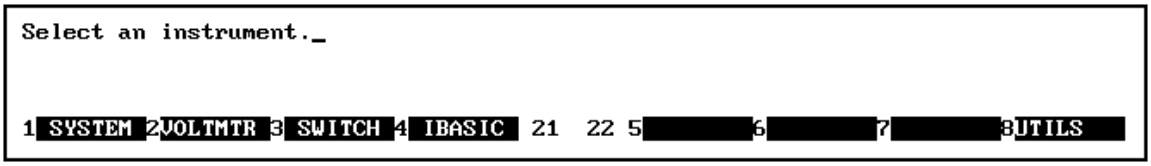
1. Move the RS-232 cable to the desired command module.
2. Press the "Ctrl-D" keys on an RS-232 terminal to guarantee that the display terminal interface is in control of the terminal.
3. Type:

   ST (followed by **Return**) for auto-identification of the terminal.
   or
   ST HP (followed by **Return**) for HP terminals - 700/94, 700/92, 26xx, 23xx
   or
   ST HP70043 (followed by **Return**) for the HP 700/43 terminal
   or
   ST VT100 (followed by **Return**) for VT100 emulators
   or
   ST VT220 (followed by **Return**) for VT220 emulators
   or
   ST WYSE30 (followed by **Return**) for WY-30 emulators
   or
   ST WYSE50 (followed by **Return**) for WY-50 emulators

   This changes the menu to correspond to the instruments in the servant area of the new command module.

Display Terminal Menu Tutorial

Following the power-on sequence or a system reset, the screen shows the `Select an instrument` menu (see Figure 3-2). This menu allows you to select one of the instruments listed.



Note: Typical instruments are shown. Actual choices depend on installed instruments.

Figure 3-2. "Select an Instrument" Menu

Figure 3-2 shows the `Select an Instrument` menu when the Flash ROMS Run/Load switch on the front of the HP E1406A Command Module is set to "Run". If this switch is in the "Load" position, the SYSTEM instrument entry will be replaced by LOADER and the rest of the instruments will disappear from the menu.

The menu select and menu control function keys (usually labeled **f1** - **f8** on their key caps) are defined by eight function labels located across the bottom of the terminal screen. Once you learn how these keys operate, using the menus is easy (key labels are shown in bold text in this chapter):

To select a displayed menu choice, press the function key (**f1** - **f5**) which corresponds to the function key label.

- When there are more than five menu choices, function key **f6** becomes labeled **MORE**. Press **MORE** to display the next group of choices. By repeatedly pressing **MORE** you can display all groups of choices. After you have displayed all groups of choices, pressing **MORE** again returns to the first group of choices.
- Whenever the screen is requesting information (input prompt) such as `Enter the device's logical address`, just type the information and press **Return** (may be **Enter** on a terminal emulator).

  If you pressed the wrong menu key and do not want to enter the requested information, you can escape the input prompt and stay at the same menu level by pressing **ESC** or **PRV_MENU**.

  If you make an incorrect entry in response to an input prompt, the bottom line of the Text Output Area will show an error message. When this happens, just select that menu choice again (**f1 - f5** keys), re-type the correct information, and press **Return**.
- Press **PRV_MENU** or **ESC** to return to the previous menu within an instrument menu or escape from an input prompt. Press **SEL_INST** to return to the `Select an Instrument` menu (see next item). Note that when you leave an instrument and return later, you return to the same menu location you were at when you left. Any information below the Text Output Area will also be redisplayed when you return.

- In addition to the instrument menu keys, **CLR_INST**, **RST_INST** and **SEL_INST** are helpful when operating instruments. These and other utility keys are accessed by pressing the **UTILS** key (see Figure 3-3). Refer to "General Key Descriptions" on page 77 for information on the **RCL_....** keys in this menu.
  - **CLR_INST** clears the instrument's terminal interface input and output buffers (remote buffers are not cleared) and returns to the top level of the instrument menu. Press **CLR_INST** whenever an instrument is busy, is not responding to terminal interface control, or to abort a command being entered from the terminal interface.
  - **RST_INST** clears all terminal interface and remote input and output buffers and resets the instrument.
  - **SEL_INST** returns you to the `Select an Instrument` menu. **SEL_INST** is the key *under* the **UTILS** key. You can easily return to the `Select an Instrument` menu by pressing **f8** twice.

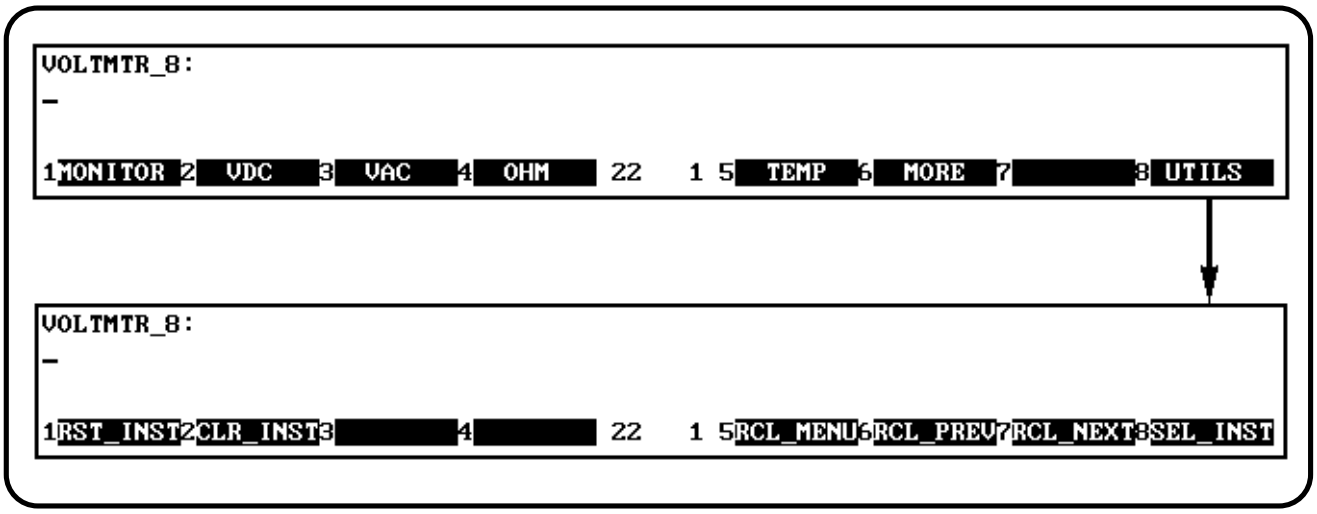


Figure 3-3. Accessing the Utility

Using the System Instrument Menu

The System instrument menu allows you to:

- Read the command module HP-IB address
- Display logical address and instrument information
- Configure the RS-232 port
- Program message-based devices
- Set the system clock and calendar
- Reset the system

The menus on the following pages demonstrate how to do each of the above.

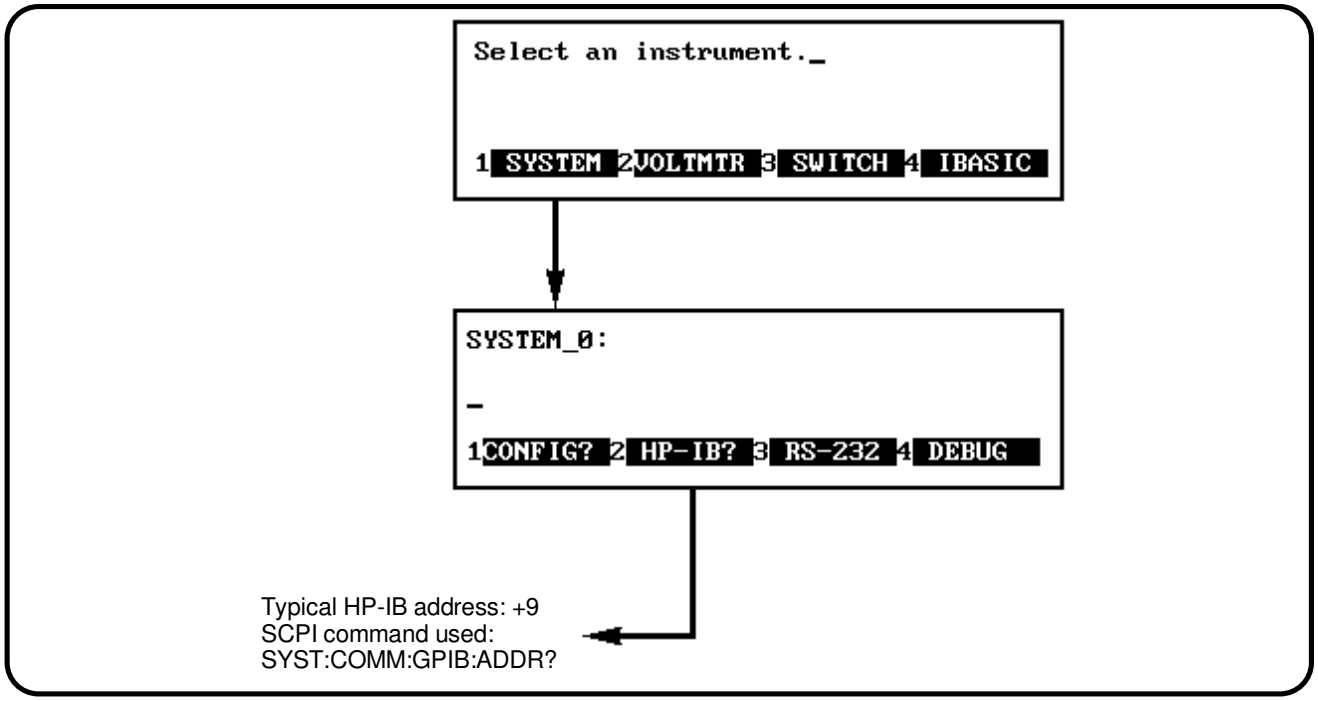
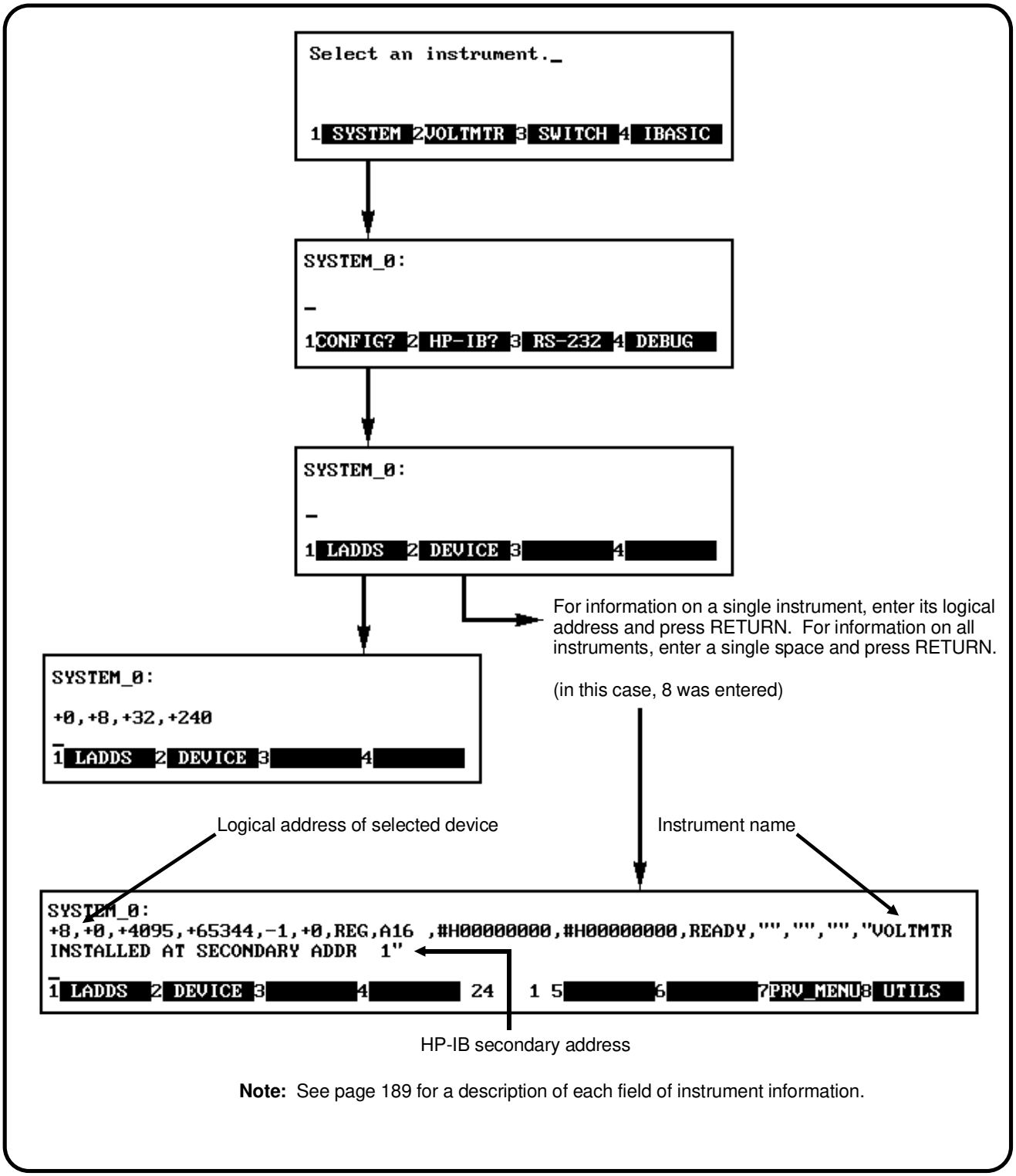
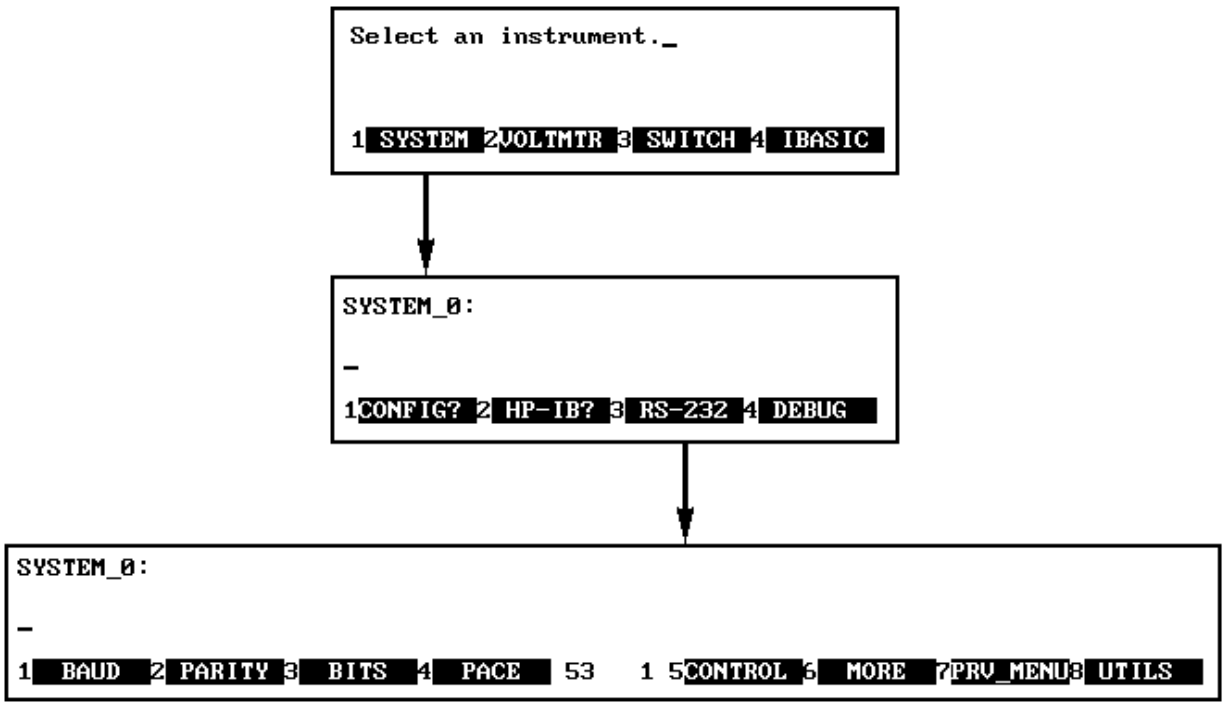


Figure 3-4. Reading the Command Module HP-IB Address



Note: See page 189 for a description of each field of instrument information.

Figure 3-5. Displaying Logical Addresses and System Instrument Information



Note: Configuration of the HP E1406A RS-232 port is covered in the *C-Size VXIbus Systems Configuration Guide*.

Figure 3-6. Configuring the Command Module RS-232 Port

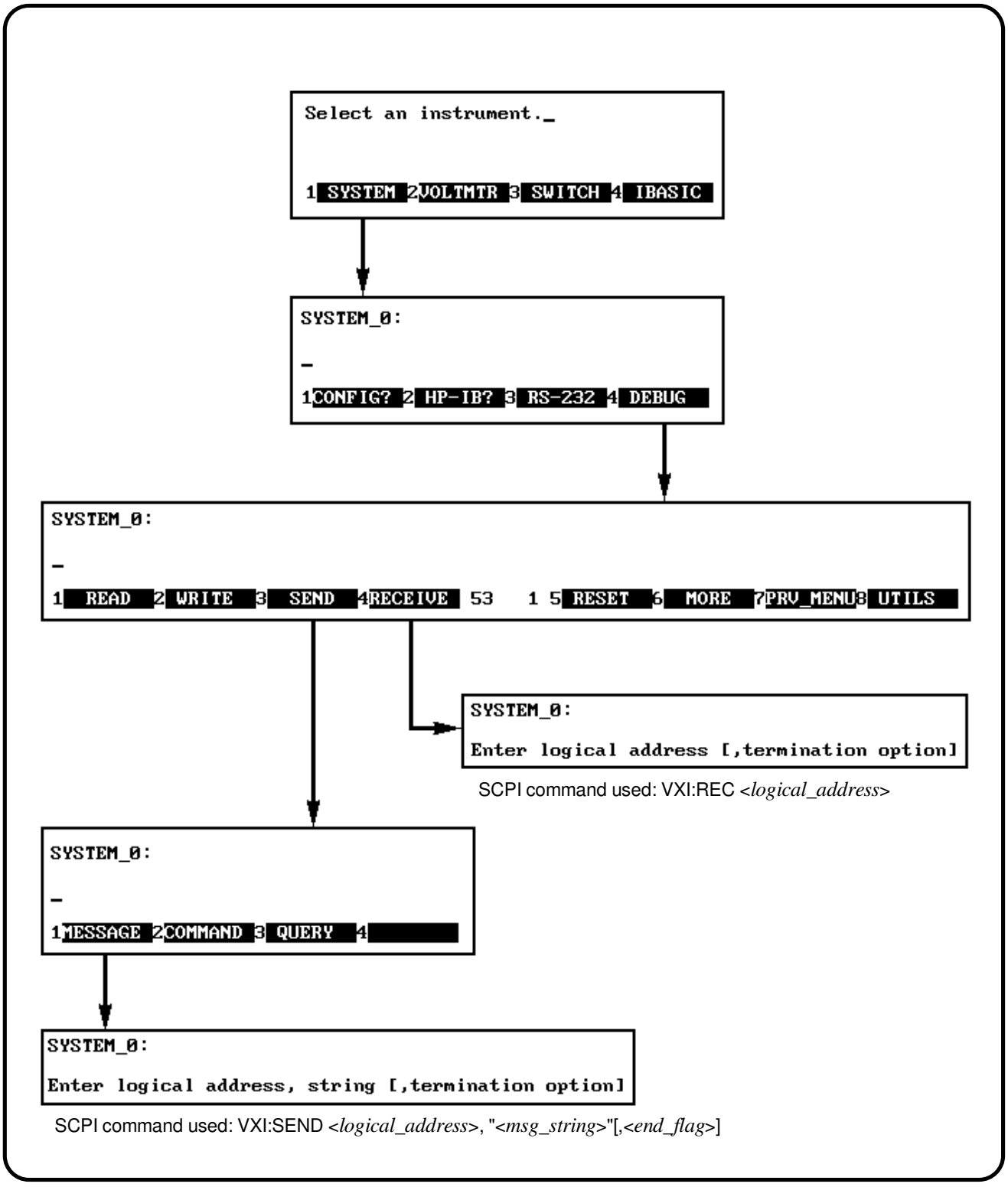


Figure 3-7. Programming Message-Based Devices

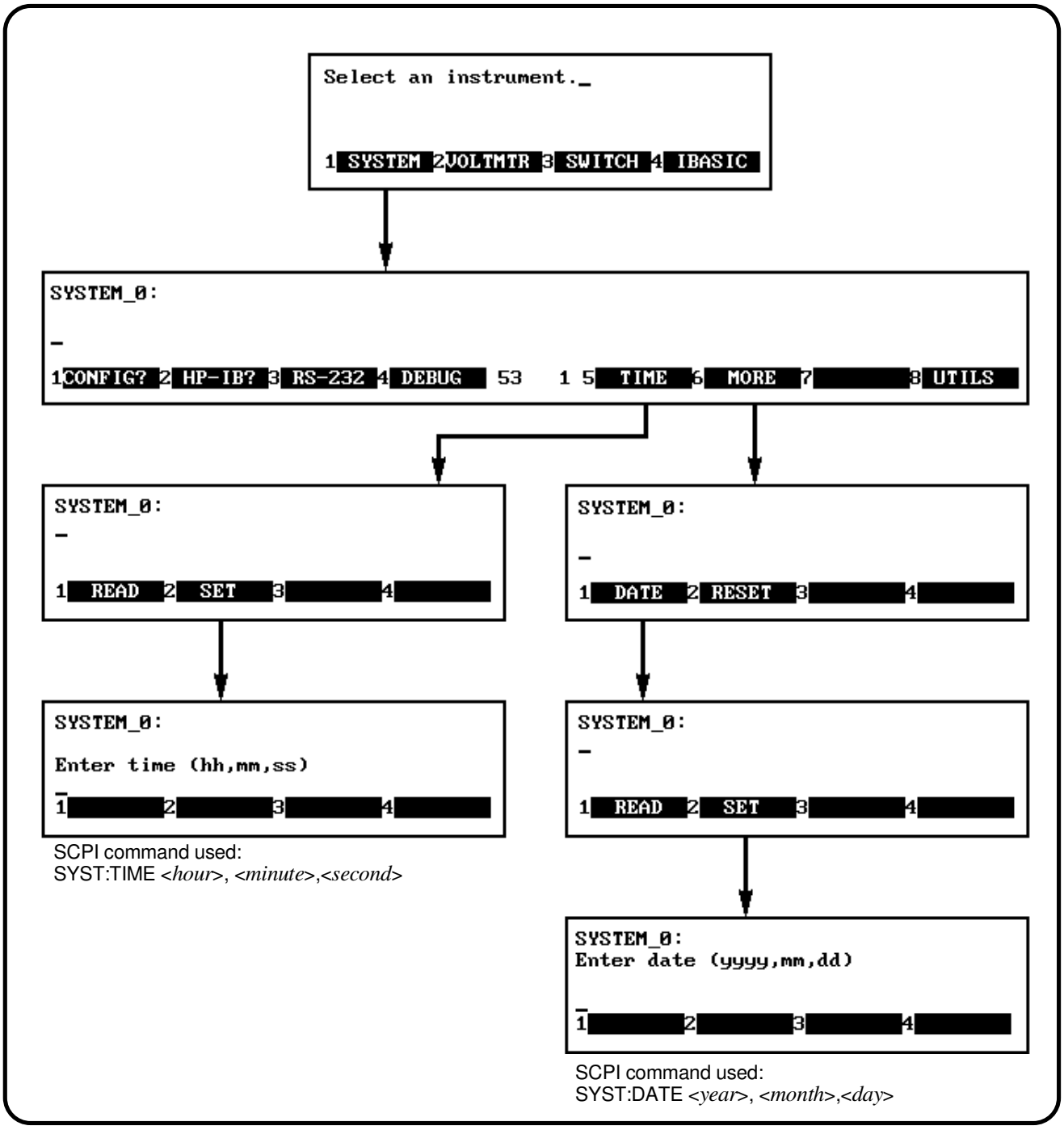
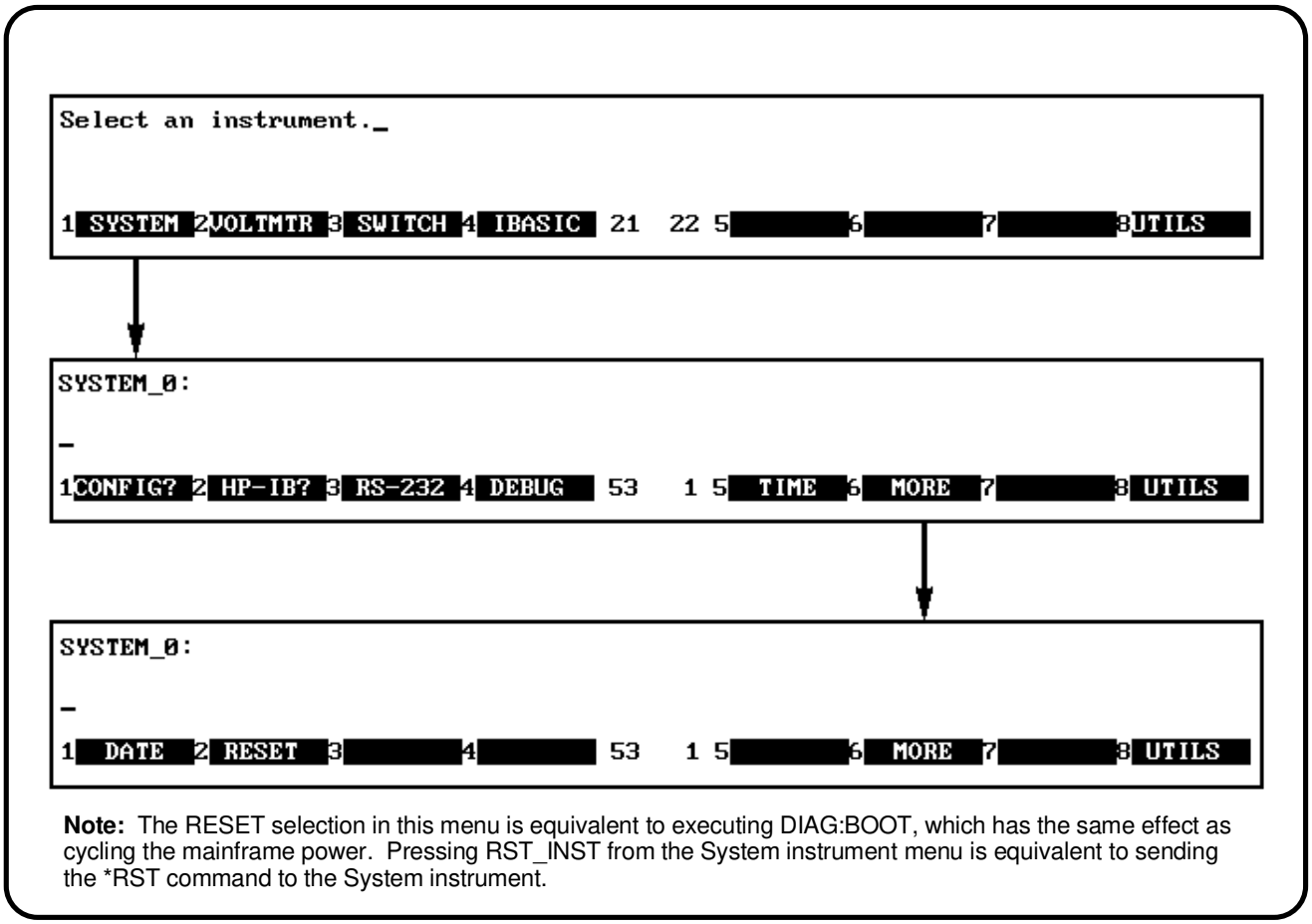


Figure 3-8. Setting the System Clock and Calendar



Note: The RESET selection in this menu is equivalent to executing DIAG:BOOT, which has the same effect as cycling the mainframe power. Pressing RST_INST from the System instrument menu is equivalent to sending the *RST command to the System instrument.

Figure 3-9. Resetting the System

Using the Loader Instrument

The Loader instrument appears on the `Select an instrument` menu when the Flash ROMS Run/Load switch on the front of the HP E1406A Command Module is set to "Load". This instrument allows you to:

- Read the command module HP-IB address
- Configure the RS-232 port(s)
- Set the system clock and calendar
- Reset the system

Using the Switchbox Menu

The instrument menus allow you to access the most-used instrument functions or to monitor an instrument (monitor mode) while it is being controlled from remote. The Switchbox menu is used as an example to show you how to use the instrument menus. Menus are available for many, but not all, instruments. See your instrument user's manual for more information on a particular instrument's menu. The Switchbox menu allows you to:

- Open and close channels
- Scan channels
- Display module (card) type and description
- Reset a selected switch module
- Monitor a switchbox

Selecting the Switchbox

To select the Switchbox, press the function key (**f1 - f5**) which corresponds to the label **SWITCH** in the `Select an instrument` menu. (If the `Select an instrument` menu is not being displayed press **UTILS** then **SEL_INST**.)

Note

After you press the function key for **SWITCH**, the screen may show: `Select SWITCH at logical address:_` while the function key labels show two or more logical addresses. This means more than one switchbox is installed in the mainframe. To select one of the switchboxes, press the function key for the logical address key label.

Figures 3-10 through 3-13 show how to use the switchbox menu. Keep the following points in mind when using the menu:

- The card number identifies a module within the switchbox. The module with the lowest logical address is always card number 01. The module with the next successive logical address is card number 02, and so on.
- The @ character is required preceding a channel list when executing a switchbox command from the terminal interface or remote. When entering a channel list in response to a menu prompt, however, do not precede it with the @ character. Doing so causes a syntax error.

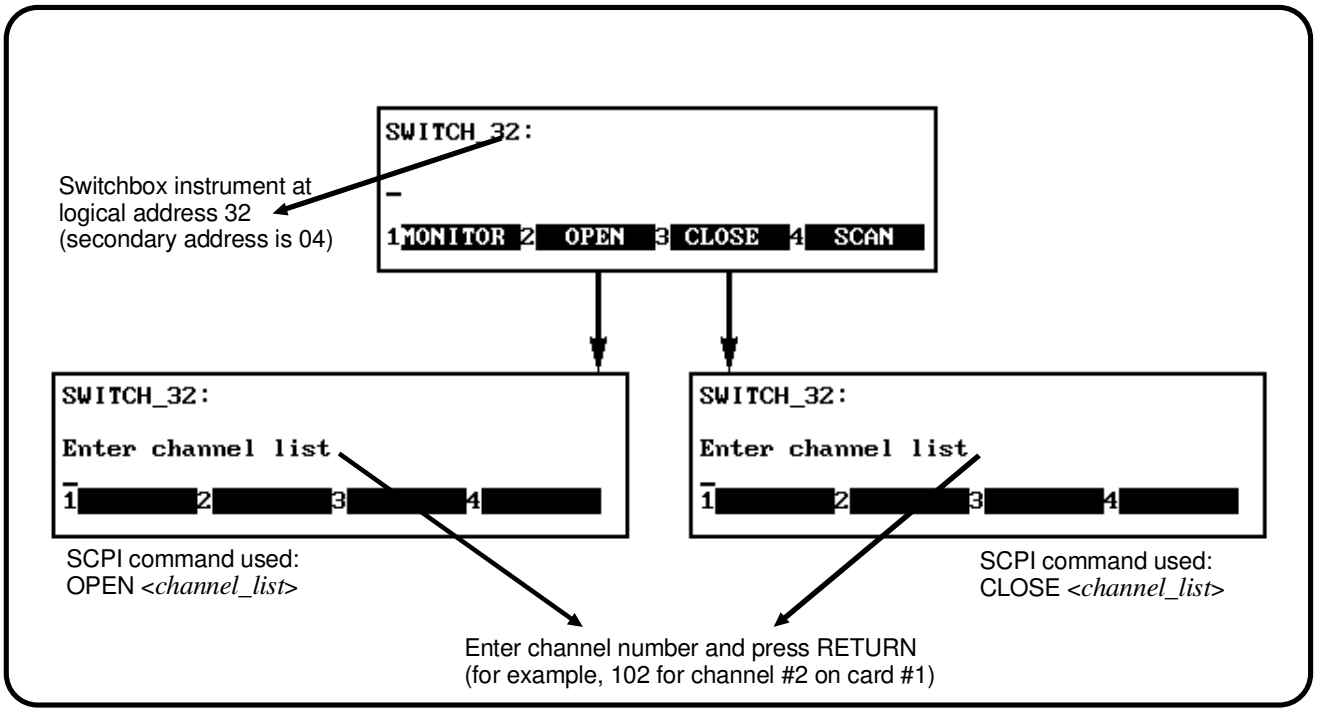


Figure 3-10. Opening and Closing Channels

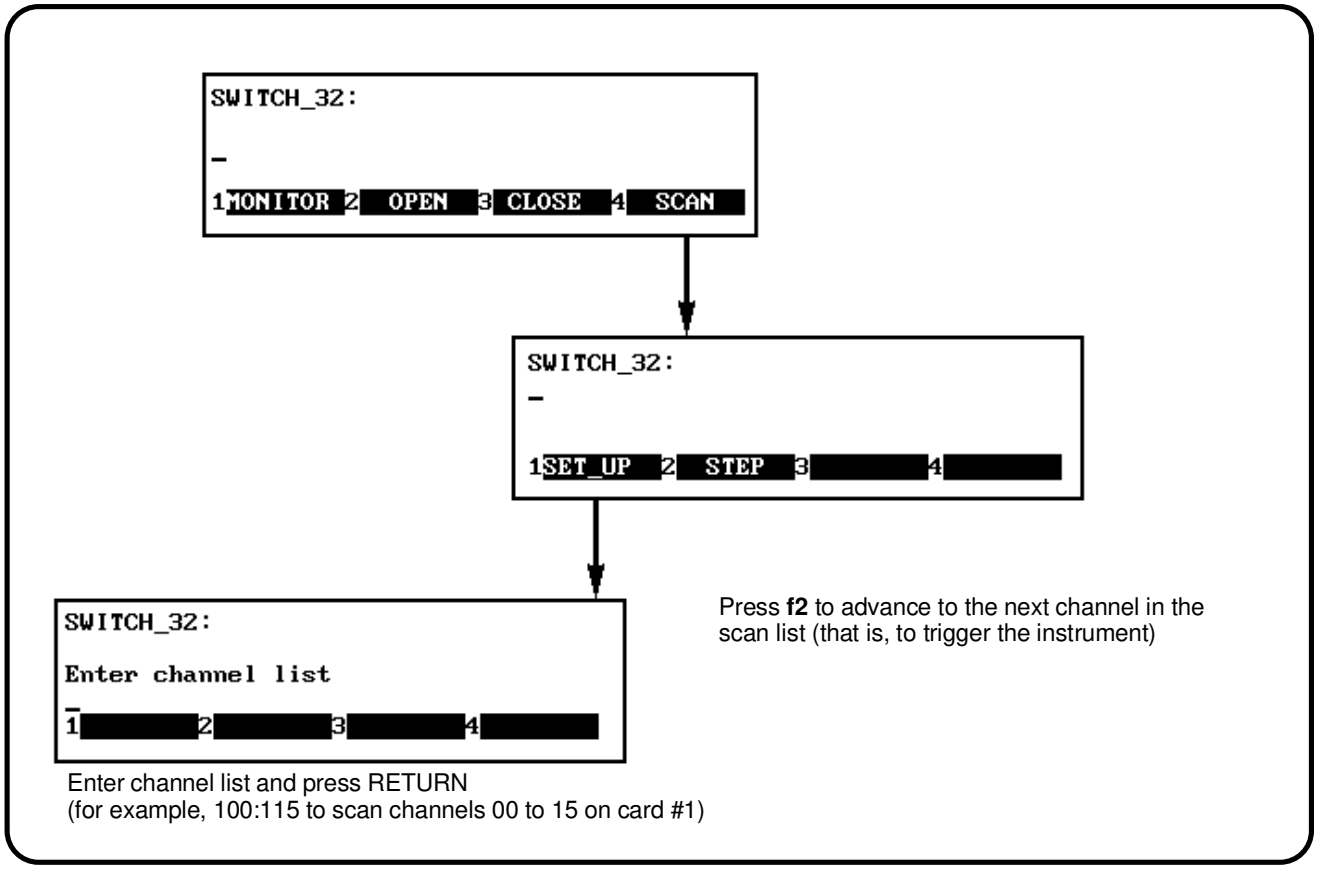


Figure 3-11. Scanning Channels

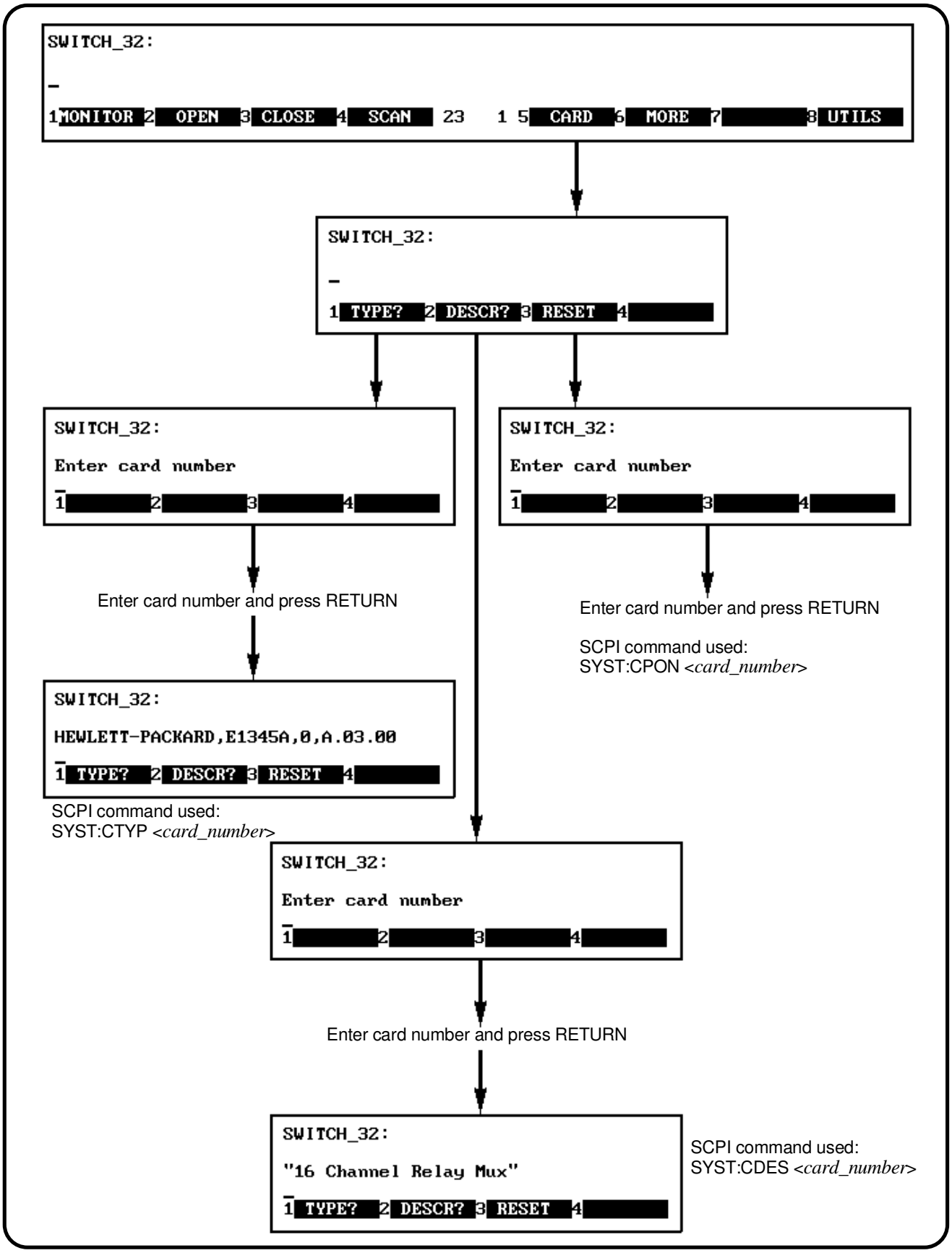


Figure 3-12. Displaying Card Type and Description or Resetting Card

Monitor Mode

Monitor mode displays the status of an instrument while it is being controlled from remote. Monitor mode is useful for debugging programs. You can place an instrument in monitor mode using terminal interface menus, or by executing the DISP:MON:STAT ON command from the terminal interface or by remote. Pressing most terminal interface keys will automatically exit monitor mode and return to the instrument menu. However, you can use the left and right arrow keys in monitor mode to view long displays.

Note Enabling monitor mode slows instrument operations. If the timing or speed of instrument operations is critical (such as making multimeter readings at a precise time interval), you should not use monitor mode.

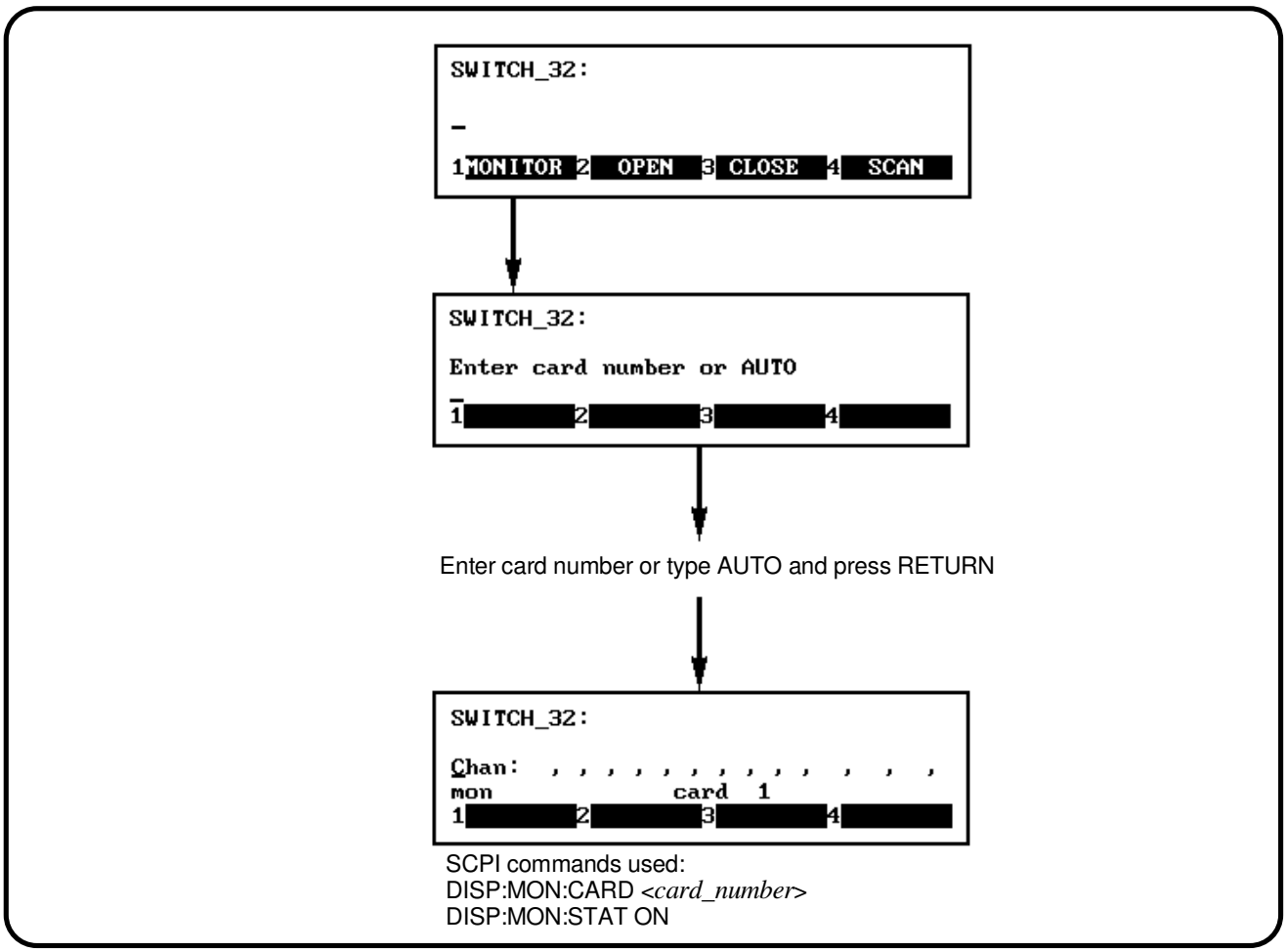


Figure 3-13. Selecting Monitor Mode

Table 3-1 shows the status annunciators that may appear in the bottom line of the screen in monitor mode. Some instruments also have device-specific annunciators (see your specific module user's manual for more information).

Table 3-1. Monitor Mode Display Annunciators

Annunciator	Description
mon	The instrument is in monitor mode.
busy	The instrument is executing a command.
err	An error has occurred (see "Reading Error Messages" below).
srq	A service request has occurred.

Reading Error Messages

Whenever the screen is showing the *err* annunciator, an error has occurred for the instrument being monitored. You can read the error message, although doing so cancels monitor mode. To read an error message, type SYST:ERR? (followed by pressing the **Return** key):

The error message will be displayed in the bottom line of the Text Output Area. To see if another error was logged, repeat the SYST:ERR? command by pressing **UTILS**, **RCL_PREV**, then **Return**.

After you have read all the error messages, executing the SYST:ERR? command causes the screen to show: `+0,"No error"`. After reading the error message(s), press the **f1** key to return to monitor mode.

Executing Commands

From the terminal interface, you can type and execute IEEE 488.2 common commands and SCPI commands for the instrument presently selected by the `Select an instrument` menu. (However, you cannot execute a command when the screen is requesting that you input information.) This is particularly useful for accessing functions not available in an instrument's menu. For example, assume you want to program the HP E1411B 5½-Digit Multimeter for 10 DC voltage measurements. To specify 10 measurements you must type in the necessary command since the command is not on the multimeter menu. After selecting the VOLTMR menu, type the following commands and press the **Return** key after each command.

```
CONF:VOLT:DC
SAMP:COUN  10
READ?
```

These commands configure the multimeter, specify 10 measurements, and display the readings on the terminal.

Editing the Terminal Display

The screen editing keys (shown on page 78) allow you to edit user-entered data or commands. When editing, the screen is in insert mode. That is, typed characters will be inserted into the string at the present cursor position.

Note The key labels shown are found on all HP terminals (except HP terminals supporting ANSI terminal protocol). See "Using Supported Terminals" on page 79 for equivalent key functions on your terminal.

General Key Descriptions

This section explains the function of each of the terminal interface's menu, menu control, and editing keys. If a key is not functional in a particular situation, pressing that key does nothing except to cause a beep.

Menu and Menu Control Keys

[f1] through [f5] Label menu choices for corresponding function keys.

[SEL_INST] Returns to the `Select an instrument` menu.

[PRV_MENU] Returns to the previous menu level within an instrument menu or escapes from an input prompt. When you reach the top of an instrument's menu, the **PRV_MENU** label disappears.

[MORE] The screen can show a maximum of five menu choices at a time. When there are more than five menu choices, function key **f6** becomes labeled **MORE**. Press **MORE** to display the next group of choices. By repeatedly pressing **MORE** you can display all groups of choices. After you have displayed all groups of choices, pressing **MORE** again returns to the first group of choices.

[RCL_PREV] Recalls the last command entered from the terminal interface. After recalling a command, it can be edited or re-executed. You can recall from a stack of previously executed commands by repeatedly pressing **RCL_PREV**. When you reach the bottom of the stack (the last line in the buffer), pressing **RCL_PREV** does nothing except to cause a beep.

[RCL_NEXT] Recalls commands in the opposite order to that of **RCL_PREV**. Pressing **RCL_NEXT** does nothing until you have pressed **RCL_PREV** at least twice.

RCL_MENU

Recalls the last SCPI command generated by a menu operation. For example, reading the time using the menus (SYSTEM, TIME, READ) generates and executes the SYST:TIME? SCPI command. A recalled command can be executed by pressing the **Return** key. You can edit a recalled command before you execute it.

Instrument Control Keys

RST_INST

Resets only the selected instrument (equivalent of executing *RST). **RST_INST** also clears the instrument's terminal interface and remote input and output buffers. **RST_INST** is the only terminal interface key that can affect an instrument being operated from remote.

CLR_INST

Clears the terminal interface input and output buffers (remote buffers are not cleared) of the selected instrument and returns to the top level of the instrument menu. Press **CLR_INST** whenever an instrument is busy, is not responding to terminal interface control, or to abort a command being entered from the terminal interface.

Editing Keys

Moves the cursor one character space to the right while leaving characters intact.

Moves the cursor one character space to the left while leaving characters intact.

Erases the character at the present cursor position (for user-entered data only).

Erases all characters from the present cursor position to the end of the input line (for user-entered data only).

Other Keys

Selects alternate key definitions. These CTRL key sequences provide short-cuts to some of the menu sequences and also provide some functions not directly available from dedicated terminal keys. Some alternate key definitions are:

CTRL

CTRL-R = Instrument Reset
CTRL-C = Clear Instrument
CTRL-D = `Select an instrument` menu.

See Table 3-3 on page 85 for a complete list of all control sequence functions. Users of the optional IBASIC interpreter should refer to their IBASIC manual set for additional editing functions.

Using Supported Terminals

The display terminal interface supports several popular terminal brands and models. This chapter will show you how to access all of the terminal interface functions described previously using your supported terminal.

The Supported Terminals

The following list names the supported terminals and shows where to go for more information. If your terminal is not named in this list, see "Using Other Terminals" on page 82.

- HP 700/92 Menu tutorial
- HP 700/94 Menu tutorial
- HP 700/22 See this page
- HP 700/43 and WYSE WY-30 See page 81

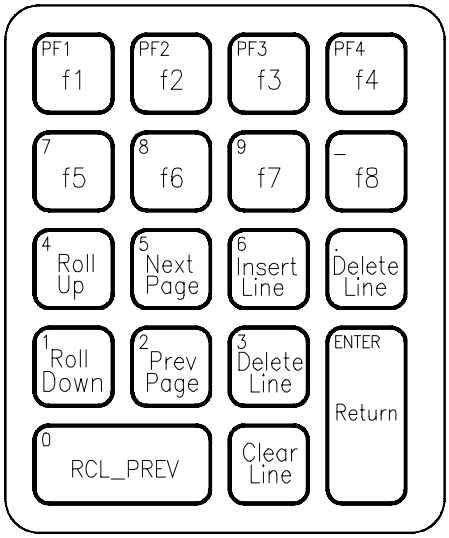
The keyboard guides provided for the listed terminals may be removed or copied, and placed near your keyboard while you go through the menu tutorial sections.

Using the HP 700/22

The HP 700/22 terminal emulates the DEC VT100 or VT220 terminals. Some functions of the display terminal interface have been mapped into keys with other labels. A keyboard map is provided for each of the emulation models. Use these keyboard maps to help locate the terminal interface functions.

VT100 Key Map

The symbols shown in the upper left corner of key each are now mapped with the function labeled in the center of each key.



Selecting VT100 Mode

To use the HP 700/22 in VT100 mode, press the **Set-Up** key and set the following configuration:

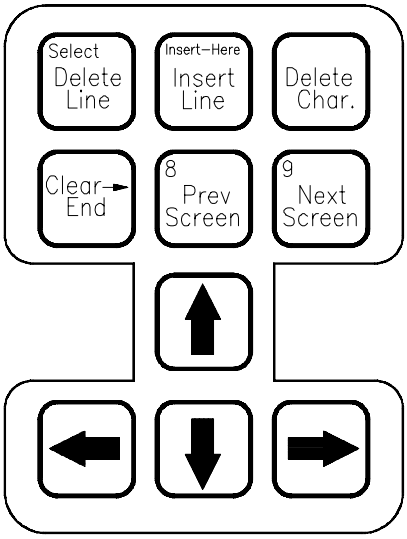
Fields	Value
Terminal Mode	EM100, 7 bit Ctrls
Columns	80
EM100 ID	EM100
Inhibit Auto Wrap	YES

VT220 Key Map

The function keys that are normally labeled **f6** through **f14** are now labeled:

Note Because the HP 700/22 keyboard has nine function keys in the center of the keyboard, f4 is mapped twice.

The symbols shown in the upper left corner of key each are now mapped with the function labeled in the center of each key.



Selecting VT220 Mode

To use the HP 700/22 in VT220 mode, press the **Set-Up** key and set the following configuration:

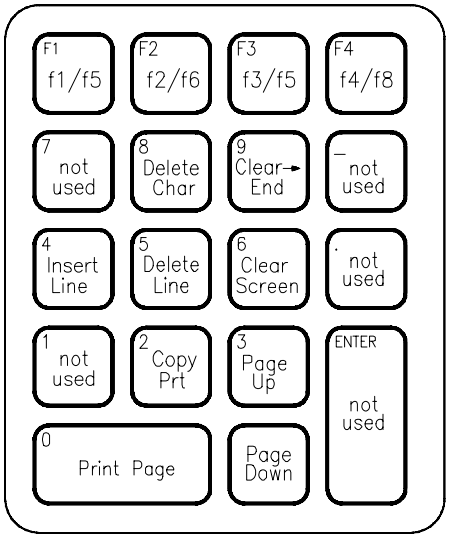
Fields	Value
Terminal Mode	EM200, 7 bit Ctrls
Columns	80
EM100 ID	EM220
Inhibit Auto Wrap	YES

Using the WYSE WY-30

With the WYSE WY-30 terminal, some functions of the display terminal interface have been assigned to keys with other labels. Use this keyboard map to help locate these functions.

The symbols shown in the upper left corner of key each are now mapped with the function labeled in the center of each key.

Where two function key labels are shown, the one following the "/" character is accessed by pressing and holding the CTRL key while pressing the desired function key (for example, to access the **f6** function, press **CTRL-f2/f6**).



Using Other Terminals

This section discusses using terminals which are not on the Supported Terminals list. Primarily this section is to help you use terminals which do not provide programmable soft keys (function keys). Without this capability, a terminal cannot access the display terminal interface's menus. Instead, the terminal interface provides a set of terminal interface commands which allow you to select instruments by name or logical address. Once selected, you can type common commands or SCPI commands to the instrument. In addition, keyboard accessible control codes provide display control for terminals which may not have keys dedicated to those functions.

What "Not Supported" Means

Strictly speaking, a terminal is not supported if it has not been rigorously tested with the display terminal interface. There are several HP terminals which may be compatible with the terminal interface. Terminals such as the DEC VT100, DEC VT220, and WYSE WY-50, or emulations of these may also work properly with the terminal interface. If you have one of these terminals, try it. Here is a list of terminals you should try.

- HP 2392A
- HP 2394A
- DEC VT100
- DEC VT220
- WYSE WY-50
- HP AdvanceLink terminal emulation software (configure as HP 2392A)

Testing Terminals for Compatibility

Here is how you test an unsupported terminal for compatibility with the display terminal interface:

1. Connect your terminal and configure its communication parameters to match the mainframe's serial interface (see Appendix C).
2. With your terminal turned on and set to "remote mode", turn on the mainframe. After the mainframe power-on self-test, the display interface sends sequences of characters to your terminal which should cause it to return its identification. If the terminal ID matches one in a list kept by the terminal interface, it will send character sequences to program the function keys and their labels.
3. If you now see the `Select an instrument` prompt *and* the `Select an instrument` menu labels, your terminal is ready to try. Go to the beginning of this chapter and try the menus.

4. If you see only the `Select an instrument` prompt without the `Select an instrument` menu labels, your terminal did not return a recognized ID. To set the terminal type manually, type the terminal interface command:

   ST HP (followed by **Return**) for HP terminals - 700/94, 700/92, 26xx,23xx

   or

   ST HP70043 (followed by **Return**) for the HP 700/43 terminal

   or

   ST VT100 (followed by **Return**) for VT100 emulators

   or

   ST VT220 (followed by **Return**) for VT220 emulators

   or

   ST WYSE30 (followed by **Return**) for WY-30 emulators

   or

   ST WYSE50 (followed by **Return**) for WY-50 emulators

   If you now see the `Select an instrument` menu labels, go to the beginning of this chapter and try the menus.

   or

   Turn the mainframe off and then on again.

   Continue with this chapter to learn how to use your terminal without menus.

Using a Terminal Without Menus

You can still control instruments installed in your mainframe without using the terminal interface menus. In this case you will send common commands and SCPI commands to your instruments by typing them on your terminal keyboard, or through a computer interface.

Selecting Instruments

To send commands to, and receive responses from an instrument, you must first select that instrument. Two commands are provided to select instruments. They are SI (Select Instrument), and SA (Select Address). These commands only work from the `Select an instrument` prompt. The commands can be typed in upper case or lower case.

SI Command

SI selects an instrument by its name, exactly as it would appear in the `Select an instrument` menu (see Table 3-2). If your mainframe has more than one instrument with the same name, follow the name with a comma (,) and the desired instrument's logical address. Here are some examples of SI commands:

- **si voltmtr** (selects a voltmeter instrument)
- **si switch** (selects a switchbox instrument)
- **SI SWITCH** (same as above)
- **si switch,16** (selects switchbox at logical address 16)

Table 3-2. Instrument Names for the SI Command

Menu Name	Instrument
SYSTEM	The System Instrument (built-in to the command module)
VOLTMTR	HP E1326B Stand-Alone, or HP E1326B Scanning Voltmeter Modules
SWITCH	Switchbox composed of one or more HP Multiplexer Modules
DIG_I/O	HP E1330B Quad 8-Bit Digital Input/Output Module
IBASIC	Optional IBASIC interpreter
COUNTER	HP E1332A 4-Channel Counter/Totalizer, or HP E1333A 3-Channel Universal Counter Modules
D/A	HP E1328A 4-Channel Digital-to-Analog Converter Module

SA Command

SA selects an instrument by its logical address. For multiple module instruments, use the logical address of the first module in the instrument. For example; **SA 8** selects the instrument at logical address 8. When you have selected an instrument, the terminal interface will respond with an instrument prompt which is the instrument's menu name followed by its logical address (e.g., **VOLTMTR_8:**).

To get a list of the logical addresses used in your mainframe, send the SCPI command VXI:CONF:DLAD? to the System instrument. Then, to determine what instrument is at each logical address, send the command VXI:CONF:DLIS? <*logical_address*> for each logical address in the list. Refer to page 189 for information about this command.

Returning to the "Select an Instrument" Prompt

To return to the `Select an instrument` prompt, press and hold the **CTRL** key then press the **D** key.

Control Sequences for Terminal Interface Functions

The terminal interface provides the keyboard control sequences listed in Table 3-3. These can be thought of as keyboard short-cuts for compatible terminals (those which provide menu capability). Only those functions in the table marked with * (asterisk) operate for "UNKNOWN" terminal types (those which do not support menus). An "UNKNOWN" terminal type has very limited editing capability. It will not support the EDIT mode for the optional IBASIC interpreter. In the following table, † = IBASIC only.

Table 3-3. Control Sequence Functions

Terminal Key	Function	Control Sequence
Backspace*	Deletes the character to the left of the cursor and moves cursor left.	CTRL-H
Del char	Delete character at the cursor position.	CTRL-X
Clr →end	Clears line from cursor position to end of line.	CTRL-L
Clear line	Clears line regardless of cursor position.	CTRL-U
Insert line †	Inserts a blank line at the cursor position.	CTRL-O
Delete line †	Deletes the line at the current cursor position.	CTRL-DEL
End of line	Move cursor to the end of current line.	CTRL-Z
Start of line	Move cursor to the beginning of current line.	CTRL-A
Return*	Terminates user entry.	CTRL-M
RCL_MENU	Recalls the last command executed via the menu keys.	CTRL-W
RCL_PREV*	Recalls the last several commands executed via user input.	CTRL-F
RCL_NEXT*	After RCL_PREV, RCL_NEXT may be used to move forward through the recalled commands.	CTRL-B
SEL_INST*	Return to "Select an instrument" menu.	CTRL-D
CLR_INST*	Clear instrument's input and output buffers.	CTRL-C
RST_INST*	Like CLR_INST plus clears.	CTRL-R

In Case of Difficulty

Problem:	Problem Cause/Solution:
Error -113 undefined header error occurs after entering data in response to a menu prompt.	For some commands used by the menus, the data entered is appended to a command header. For example, if you enter "1" as the port number for a digital I/O module, the command used is DIG:HAND1:MODE NONE where HAND1 indicates the port number. If your entry was invalid or incorrect, error -113 occurs.

Following the power-on sequence or system reset the display shows:
 	
`Configuration errors. Select SYSTEM`
`Press any key to continue_`
	An unassigned device (incorrect logical address) was detected., If you cycle power or perform system reset, the display will show the logical address of the unassigned device. You can also check the logical addresses using the CONFIG? -- LADDS branch of the System instrument menu. You can also use SYST:ERR? in the system instrument.

The display shows: `instrument in local lockout`. Menus seem to work but nothing happens when I reach the bottom level or try to execute a command.	The terminal interface has been locked-out (HP-IB local lockout). You can re-enable menu operation by cancelling local lockout (from remote) or by cycling mainframe power.
Display cannot be removed from monitor mode.	Monitor mode was entered (DISP:MON:STAT ON command) and the terminal interface has also been locked out (HP-IB local lockout). Either cancel the local lockout or execute DISP:MON:STAT OFF (from remote).

Display shows:
 	
`Cannot connect to instrument`
`Press any key to continue_`
	A hardware or software problem has occurred in the instrument preventing it from responding to terminal interface control.

After selecting an instrument the display shows: `busy`	The instrument is busy performing an operation. Press **Clear Instr** to abort the instrument operations and allow the terminal interface to access the instrument.

Display shows:
 	
`Instrument in use by another display`
`Press any key to continue_`
	The instrument has already been selected from another terminal interface. An instrument can only be “attached” to one display at a time. At the other terminal interface, press **Select Instr**. The instrument can now be selected from the desired terminal interface.

System Instrument/Switchbox Menus

This section contains charts showing the structure and content for the HP E1406A Command Module's System instrument and switchbox terminal interface instrument menus. The SCPI commands used and descriptions of menu-controlled instrument operations are also included in the charts. You may want to refer to these charts as examples for other instrument menus. See the appropriate instrument user's manual for menus specific to that instrument.

System Instrument Menu

Menu Levels and Content

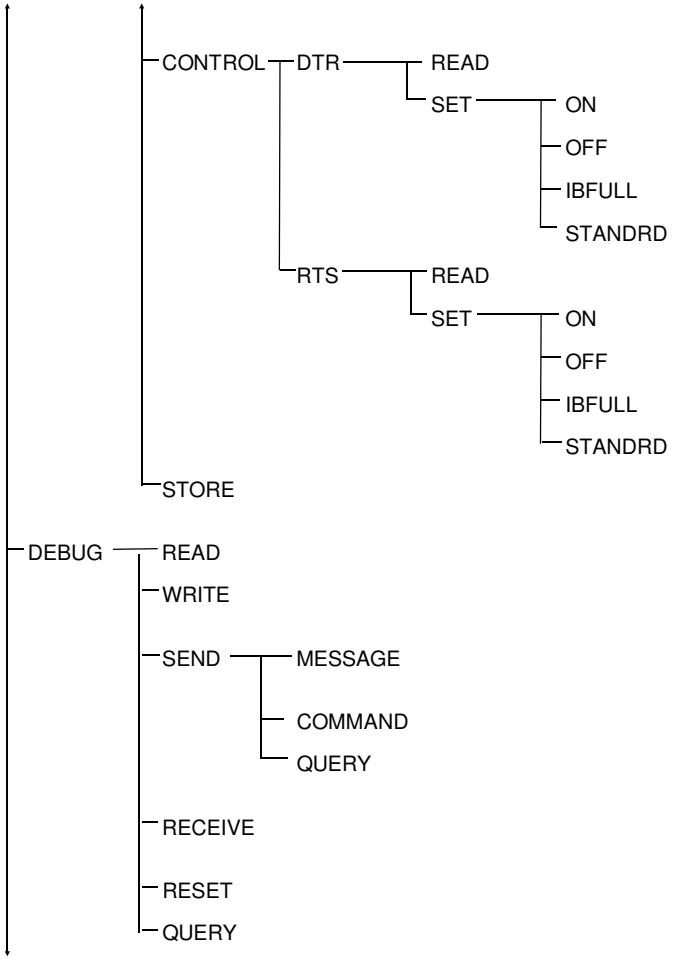
Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	User Entry	Command(s) Used	Description
SYSTEM	CONFIG?	LADDS					VXI:CONF:DLAD?	Displays logical addresses of mainframe instruments.
		DEVICE				logical address	VXI:CONF:DLIS? <*logical_addr*>	Displays information about the device at the specified logical address. (Refer to the Command Reference for details).
	HP-IB?						SYST:COMM:GPIB:ADDR?	Displays HP-IB address.
	RS232	BAUD	READ			card number	SYST:COMM:SER[*n*]:BAUD?	Read current baud rate.
			SET	300		card number	SYST:COMM:SER[*n*]:BAUD 300	Sets the serial interface baud rate to 300.
				1200		card number	SYST:COMM:SER[*n*]:BAUD 1200	Sets the serial interface baud rate to 1200.
				2400		card number	SYST:COMM:SER[*n*]:BAUD 2400	Sets the serial interface baud rate to 2400.
				9600		card number	SYST:COMM:SER[*n*]:BAUD 9600	Sets the serial interface baud rate to 9600.
				19200		card number	SYST:COMM:SER[*n*]:BAUD 19200	Sets the serial interface baud rate to 19200.
		PARITY	READ			card number	SYST:COMM:SER[*n*]:PAR?	Read current parity type.
			SET	EVEN		card number	SYST:COMM:SER[*n*]:PAR EVEN	Sets the serial interface parity to even.
				ODD		card number	SYST:COMM:SER[*n*]:PAR ODD	Sets the serial interface parity to odd.
				ONE		card number	SYST:COMM:SER[*n*]:PAR ONE	Sets the serial interface parity to one.
				ZERO		card number	SYST:COMM:SER[*n*]:PAR ZERO	Sets the serial interface parity to zero.
				NONE		card number	SYST:COMM:SER[*n*]:PAR NONE	Sets the serial interface parity to none.
		BITS	READ			card number	SYST:COMM:SER[*n*]:BITS?	Read current data bit width.
			SET	7		card number	SYST:COMM:SER[*n*]:BITS 7	Sets the data width to 7 bits.
				8		card number	SYST:COMM:SER[*n*]:BITS 8	Sets the data width to 8 bits.
		PACE	READ			card number	SYST:COMM:SER[*n*]:PACE?	Read current pacing type.
			SET	XON/OFF		card number	SYST:COMM:SER[*n*]:PACE XON	Enables XON/XOFF software handshaking.
				NONE		card number	SYST:COMM:SER[*n*]:PACE NONE	Disables XON/XOFF software handshaking.

(*Continued on next page*)

System Instrument Menu

Menu Levels and Content

(*Continued from previous page*)



Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	User Entry	Command(s) Used	Description
		CONTROL	DTR	READ		card number	SYST:COMM:SER[*n*]:CONT:DTR?	Read current setting for DTR line.
				SET	ON	card number	SYST:COMM:SER[*n*]:CONT:DTR ON	Set DTR line to static +V.
					OFF	card number	SYST:COMM:SER[*n*]:CONT:DTR OFF	Set DTR line to static -V.
					IBFULL	card number	SYST:COMM:SER[*n*]:CONT:DTR IBF	Set DTR for hardware handshaking.
					STANDRD	card number	SYST:COMM:SER[*n*]:CONT:DTR STAN	DTR operates to RS-232 standard.
			RTS	READ		card number	SYST:COMM:SER[*n*]:CONT:RTS?	Read current setting for RTS line.
				SET	ON	card number	SYST:COMM:SER[*n*]:CONT:RTS ON	Set RTS line to static +V.
					OFF	card number	SYST:COMM:SER[*n*]:CONT:RTS OFF	Set RTS line to static -V.
					IBFULL	card number	SYST:COMM:SER[*n*]:CONT:RTS IBF	Set RTS for hardware handshaking.
					STANDRD	card number	SYST:COMM:SER[*n*]:CONT:RTS STAN	RTS operates to RS-232 standard.
		STORE				card number	DIAG:COMM:SER[*n*]:STORE	Store current serial communications settings into non-volatile storage.
	DEBUG	READ				laddr, reg_num	VXI:READ? <*laddr*>,<*register_num*>	Read register in A16 address space.
		WRITE				laddr, reg_num, data	VXI:WRIT <*laddr*>,<*register_num*>,<*data*>	Write data to register in A16 address space.
		SEND	MESSAGE			laddr, string	VXI:SEND <*laddr*>,<*string*>	Send SCPI command to message-based instrument at laddr.
			COMMAND			laddr, command	VXI:SEND:COMM <*laddr*>,<*command*>	Send word serial command to laddr.
			QUERY			laddr, query	VXI:SEND:COMM? <*laddr*>,<*query*>	Send word serial command and wait for response.
		RECEIVE				laddr	VXI:REC? <*laddr*>	Receive message from message-based device.
		RESET				laddr	VXI:RES <*laddr*>	Soft reset of device at laddr.
		QUERY				laddr	VXI:QUER? <*laddr*>	Read Data Low register.

(*Continued on next page*)

System Instrument Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	User Entry	Command(s) Used	Description
(*Continued from previous page*)								
	TIME	READ					SYST:TIME?	Read the current system clock.
		SET				time	SYST:TIME *<time>*	Set the system clock.
	DATE	READ					SYST:DATE?	Read the current system calendar.
		SET				date	SYST:DATE *<date>*	Set the system calendar.
	RESET						DIAG:BOOT	Resets mainframe using the configuration stored in non-volatile memory.

Switchbox Menu

Menu Levels and Content

Level 1	Level 2	Level 3	User Entry	Command(s) Used	Description
SWITCH	MONITOR		card number ‡ or AUTO	DISP:MON:CARD <*card_number*> ;STAT ON	Monitor instrument operations.
	OPEN		channel list †	OPEN (@channel_list)	Open channel(s).
	CLOSE		channel list †	CLOS (@channel_list)	Close channel(s).
	SCAN	SET_UP	channel list †	TRIG:SOUR HOLD;:SCAN <*channel_list*>;:INIT	Set up channels to scan.
		STEP	channel list †	TRIG	Step to next channel in scan list.
	CARD	TYPE?	card number ‡	SYST:CTYP? <*card_number*>	Display module ID information.
		DESCR?	card number ‡	SYST:CDES? <*card_number*>	Display module description.
		RESET	card number ‡	SYST:CPON <*card_number*>	Return module to power-on state.
	TEST			*TST?	Runs self-test, displays results (+0 = pass; any other number = fail).

† Channel lists are of the form “ccnn” (single channel), “ccnn,ccnn” (two or more channels) or “ccnn:ccnn” (range of channels); where “cc” is the card number and “nn” is the channel number. For example, to access channel 2 on card number 1 specify 102.

‡ The card number identifies a module within the switchbox. The switch module with the lowest logical address is always card number 01. The switch module with the next successive logical address is card number 02, and so on.

Scanning Voltmeter Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	User Entry	Command(s) Used	Description
VOLTMTR	MONITOR			channel list † or 0 for auto	DISP:MON:CHAN <*channel_list*>;STAT ON	Monitor instrument operations.
	VDC			channel list †	MEAS:VOLT:DC? <*channel_list*>	Measure DC voltage on each channel.
	VAC			channel list †	MEAS:VOLT:AC? <*channel_list*>	Measure AC voltage on each channel.
	OHM			channel list †	MEAS:RES? <*channel_list*>	Measure 2-wire resistance on each channel.
	TEMP	TCOUPLE	B	channel list †	MEAS:TEMP? TC,B, <*channel_list*>	Measure °C of B thermocouple on each channel.
			E	channel list †	MEAS:TEMP? TC,E, <*channel_list*>	Measure °C of E thermocouple on each channel.
			J	channel list †	MEAS:TEMP? TC,J, <*channel_list*>	Measure °C of J thermocouple on each channel.
			K	channel list †	MEAS:TEMP? TC,K, <*channel_list*>	Measure °C of K thermocouple on each channel.
			N14	channel list †	MEAS:TEMP? TC,N14, <*channel_list*>	Measure °C of N14 thermocouple on each channel.
			N28	channel list †	MEAS:TEMP? TC,N28, <*channel_list*>	Measure °C of N28 thermocouple on each channel.
			R	channel list †	MEAS:TEMP? TC,R, <*channel_list*>	Measure °C of R thermocouple on each channel.
			S	channel list †	MEAS:TEMP? TC,S, <*channel_list*>	Measure °C of S thermocouple on each channel.
			T	channel list †	MEAS:TEMP? TC,T, <*channel_list*>	Measure °C of T thermocouple on each channel.
		THERMIS	2252	channel list †	MEAS:TEMP? THER,2252, <*channel_list*>	Measure °C of 2252 Ω thermistor on each channel.
			5K	channel list †	MEAS:TEMP? THER,5000, <*channel_list*>	Measure °C of 5k Ω thermistor on each channel.
			10K	channel list †	MEAS:TEMP? THER,10000, <*channel_list*>	Measure °C of 10k Ω thermistor on each channel.
		RTD	385	channel list †	MEAS:TEMP? RTD,85, <*channel_list*>	Measure °C of 385 RTD on each channel (4-wire).
			392	channel list †	MEAS:TEMP? RTD,92, <*channel_list*>	Measure °C of 392 RTD on each channel (4-wire).
	STRAIN	QUARTER		channel list †	MEAS:STR:QUAR? <*channel_list*>	Measure strain with quarter bridge.
		HALF	BENDING	channel list †	MEAS:STR:HBEN? <*channel_list*>	Measure strain with bending half bridge.
			POISSON	channel list †	MEAS:STR:HPO? <*channel_list*>	Measure strain with Poisson half bridge.
		FULL	BENDING	channel list †	MEAS:STR:FBEN? <*channel_list*>	Measure strain with bending full bridge.
			BENPOIS	channel list †	MEAS:STR:FBP? <*channel_list*>	Measure strain with bending Poisson full bridge.
			POISSON	channel list †	MEAS:STR:FPO? <*channel_list*>	Measure strain with Poisson full bridge.

(***Continued on next page***)

Scanning Voltmeter Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	User Entry	Command(s) Used	Description
(*Continued from previous page*)						
		UNSTRN		channel list †	MEAS:STR:UNST? <*channel_list*>	Measure bridge unstrained.
		DIAG	COMPRES	channel list †	MEAS:STR:QCOM? <*channel_list*>	Compression shunt diagnostic.
			TENSION	channel list †	MEAS:STR:QTEN? <*channel_list*>	Tension shunt diagnostic.
	CARD	TYPE?		card number ‡	SYST:CTYP? <*card_number*>	Displays module ID information.
		DESCR?		card number ‡	SYST:CDES? <*card_number*>	Displays module description.
	TEST				*TST?	Runs self-test, displays results (+0 = pass; any other number = fail).

† Channel lists are of the form “ccnn” (single channel), “ccnn,ccnn” (two or more channels) or “ccnn:ccnn” (range of channels); where “cc” is the card number and “nn” is the channel number. For example, to access channel 2 on card number 1 specify 102.

‡ The card number identifies a module within the switchbox. The switch module with the lowest logical address is always card number 01. The switch module with the next successive logical address is card number 02, and so on.

HP E1326B/E1411B 5½-Digit Multimeter (Stand-Alone) Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	User Entry	Command(s) Used	Description
VOLTMTR	MONITOR				DISP:MON:STAT ON	Display instrument operations.
	VDC				MEAS:VOLT:DC?	Measure DC volts.
	VAC				MEAS:VOLT:AC?	Measure AC volts.
	OHM				MEAS:FRES?	Measure 4-wire ohms.
	TEMP	THERMIS	2252		MEAS:TEMP? FTH,2252	Measure °C of 2252Ω thermistor (4-wire measurement).
			5K		MEAS:TEMP? FTH,5000	Measure °C of 5kΩ thermistor (4-wire measurement).
			10K		MEAS:TEMP? FTH,10000	Measure °C of 10kΩ thermistor (4-wire measurement).
		RTD	385		MEAS:TEMP FRTD,85?	Measure °C of 100Ω RTD with alpha = 385 (4-wire measurement).
			392		MEAS:TEMP FRTD,92?	Measure °C of 100Ω RTD with alpha = 392 (4-wire measurement).
	TEST				*TST?	Run self-test, display results (0 = pass; any other number = fail).

HP E1328A 4-Channel D/A Converter Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	User Entry	Command(s) Used	Description
D/A	MONITOR	CHAN1			DISP:MON:CHAN 1;STAT ON	Monitor instrument operations on channel 1.
		CHAN2			DISP:MON:CHAN 2;STAT ON	Monitor instrument operations on channel 2.
		CHAN3			DISP:MON:CHAN 3;STAT ON	Monitor instrument operations on channel 3.
		CHAN4			DISP:MON:CHAN 4;STAT ON	Monitor instrument operations on channel 4.
		AUTO			DISP:MON:CHAN AUTO;STAT ON	Monitor instrument operations on active channel.
	OUTPUT	VOLTAGE	CHAN1	voltage †	VOLT1 <*voltage*>	Output voltage on channel 1.
			CHAN2	voltage †	VOLT2 <*voltage*>	Output voltage on channel 2.
			CHAN3	voltage †	VOLT3 <*voltage*>	Output voltage on channel 3.
			CHAN4	voltage †	VOLT4 <*voltage*>	Output voltage on channel 4.
		CURRENT	CHAN1	current ‡	CURR1 <*current*>	Output current on channel 1.
			CHAN2	current ‡	CURR2 <*current*>	Output current on channel 2.
			CHAN3	current ‡	CURR3 <*current*>	Output current on channel 3.
			CHAN4	current ‡	CURR4 <*current*>	Output current on channel 4.
	TEST				*TST?	Run self-test, display results (+0 = pass; any other number = fail).

† Enter voltage values in volts. Typical examples are: +3.5, -2, +500E-3.

‡ Enter current values in amps. Typical examples are: .05, +200E-3.

HP E1330A/B Quad 8-Bit Digital Input/Output Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	User Entry	Command(s) Used	Description
DIG_I/O	MONITOR	PORT0			DISP:MON:CHAN 0;STAT ON	Monitor instrument operations on port 0.
		PORT1			DISP:MON:CHAN 1;STAT ON	Monitor instrument operations on port 1.
		PORT2			DISP:MON:CHAN 2;STAT ON	Monitor instrument operations on port 2.
		PORT3			DISP:MON:CHAN 3;STAT ON	Monitor instrument operations on port 3.
		AUTO			DISP:MON:CHAN AUTO;STAT ON	Monitor instrument operations on any active port.
	READ	R_BYTE	PORT0		DIG:HAND0:MODE NONE;:MEAS:DIG:DATA0?	Reads port 0 after handshake.
			PORT1		DIG:HAND1:MODE NONE;:MEAS:DIG:DATA1?	Reads port 1 after handshake.
			PORT2		DIG:HAND2:MODE NONE;:MEAS:DIG:DATA2?	Reads port 2 after handshake.
			PORT3		DIG:HAND3:MODE NONE;:MEAS:DIG:DATA3?	Reads port 3 after handshake.
		R_BIT	PORT0	bit (0-7)	DIG:HAND0:MODE NONE;:MEAS:DIG:DATA0:BIT*m*?	Reads bit m on port 0 after handshake.
			PORT1	bit (0-7)	DIG:HAND1:MODE NONE;:MEAS:DIG:DATA1:BIT*m*?	Reads bit *m* on port 1 after handshake.
			PORT2	bit (0-7)	DIG:HAND2:MODE NONE;:MEAS:DIG:DATA2:BIT*m*?	Reads bit m on port 2 after handshake.
			PORT3	bit (0-7)	DIG:HAND3:MODE NONE;:MEAS:DIG:DATA3:BIT*m*?	Reads bit m on port 3 after handshake.
	WRITE	W_BYTE	PORT0	data (0-255)	DIG:HAND0:MODE NONE;:DIG:DATA0 <*data*>	Writes data to port 0.
			PORT1	data (0-255)	DIG:HAND1:MODE NONE;:DIG:DATA1 <*data*>	Writes data to port 1.
			PORT2	data (0-255)	DIG:HAND2:MODE NONE;:DIG:DATA2 <*data*>	Writes data to port 2.
			PORT3	data (0-255)	DIG:HAND3:MODE NONE;:DIG:DATA3 <*data*>	Writes data to port 3.
		W_BIT	PORT0	bit (0-7), value (0,1)	DIG:HAND0:MODE NONE;:DIG:DATA0:BIT*m* <*value*>	Writes data to bit *m* on port 0.
			PORT1	bit (0-7), value (0,1)	DIG:HAND1:MODE NONE;:DIG:DATA1:BIT*m* <*value*>	Writes data to bit *m* on port 1.
			PORT2	bit (0-7), value (0,1)	DIG:HAND2:MODE NONE;:DIG:DATA2:BIT*m* <*value*>	Writes data to bit *m* on port 2.
			PORT3	bit (0-7), value (0,1)	DIG:HAND3:MODE NONE;:DIG:DATA3:BIT*m* <*value*>	Writes data to bit *m* on port 3.

HP E1332A 4-Channel Counter/Totalizer Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	Level 5	User Entry	Command(s) Used	Description
COUNTER	MONITOR	CHAN1				DISP:MON:CHAN 1;STAT ON	Monitor instrument operations on channel 1.
		CHAN2				DISP:MON:CHAN 2;STAT ON	Monitor instrument operations on channel 2.
		CHAN3				DISP:MON:CHAN 3;STAT ON	Monitor instrument operations on channel 3.
		CHAN4				DISP:MON:CHAN 4;STAT ON	Monitor instrument operations on channel 4.
		AUTO				DISP:MON:CHAN AUTO;STAT ON	Monitor instrument operations on active channel.
	INPUT	LEVEL	CHAN1&2		voltage †	SENS1:EVEN:LEV <*value*>	Set level trigger voltage for channels 1 & 2.
			CHAN3&4		voltage †	SENS3:EVEN:LEV <*value*>	Set level trigger voltage for channels 3 & 4.
		SLOPE	CHAN1	POS		SENS1:EVEN:SLOP POS	Positive level trigger slope for channel 1.
				NEG		SENS1:EVEN:SLOP NEG	Negative level trigger slope for channel 1.
			CHAN2	POS		SENS2:EVEN:SLOP POS	Positive level trigger slope for channel 2.
				NEG		SENS2:EVEN:SLOP NEG	Negative level trigger slope for channel 2.
			CHAN3	POS		SENS3:EVEN:SLOP POS	Positive level trigger slope for channel 3.
				NEG		SENS3:EVEN:SLOP NEG	Negative level trigger slope for channel 3.
			CHAN4	POS		SENS4:EVEN:SLOP POS	Positive level trigger slope for channel 4.
				NEG		SENS4:EVEN:SLOP NEG	Negative level trigger slope for channel 4.
		ISOLATE	ON			INP:ISOL ON	Input isolation on.
			OFF			INP:ISOL OFF	Input isolation off.
		FILTER	ON			INP:FILT ON	Input filter on.
			OFF			INP:FILT OFF	Input filter off.
			FREQ		frequency ‡	INP:FILT:FREQ <*value*>	Set input filter frequency.
	FREQ	CHAN1				TRIG:SOUR IMM;:MEAS1:FREQ?	Frequency measurement on channel 1.
		CHAN3				TRIG:SOUR IMM;:MEAS3:FREQ?	Frequency measurement on channel 3.
	PERIOD	CHAN1				TRIG:SOUR IMM;:MEAS1:PER?	Period measurement on channel 1.
		CHAN3				TRIG:SOUR IMM;:MEAS3:PER?	Period measurement on channel 3.

(***Continued on next page***)

HP E1332A 4-Channel Counter/Totalizer Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	Level 5	User Entry	Command(s) Used	Description
(*Continued from previous page*)							
	TIMEINT	CHAN1				TRIG:SOUR IMM;:MEAS1:TINT?	Time interval measurement on channel 1.
		CHAN3				TRIG:SOUR IMM;:MEAS3:TINT?	Time interval measurement on channel 3.
	POS_PW	CHAN2				TRIG:SOUR IMM;:MEAS2:PWID?	Positive pulse width measurement on channel 2.
		CHAN4				TRIG:SOUR IMM;:MEAS4:PWID?	Positive pulse width measurement on channel 4.
	NEG_PW	CHAN2				TRIG:SOUR IMM;:MEAS2:NWID?	Negative pulse width measurement on channel 2.
		CHAN4				TRIG:SOUR IMM;:MEAS4:NWID?	Negative pulse width measurement on channel 4.
	UDCOUNT	CHAN1	START			TRIG:SOUR IMM;:CONF1:UDC;:INIT1	Up/down count, subtract ch. 2 count from ch. 1 count.
			READ			FETC1?	Get up/down count from channels 1 & 2.
		CHAN3	START			TRIG:SOUR IMM;:CONF3:UDC;:INIT3	Up/down count, subtract ch. 4 count from ch. 3 count.
			READ			FETC3?	Get up/down count from channels 3 and 4.
	TOTALIZ	CHAN1	START			TRIG:SOUR IMM;:CONF1:TOT;:INIT1	Totalize on channel 1.
			READ			FETC1?	Get totalize count on channel 1.
		CHAN2	START			TRIG:SOUR IMM;:CONF2:TOT;:INIT2	Totalize on channel 2.
			READ			FETC2?	Get totalize count on channel 2.
		CHAN3	START			TRIG:SOUR IMM;:CONF3:TOT;:INIT3	Totalize on channel 3.
			READ			FETC3?	Get totalize count on channel 3.
		CHAN4	START			TRIG:SOUR IMM;:CONF4:TOT;:INIT4	Totalize on channel 4.
			READ			FETC4?	Get totalize count on channel 4.
	TEST					*TST?	Run self-test, display results (+0 = pass; any other number = fail).

† Enter voltage values in volts. Typical examples are: +3.5, -2, +500E-3.

‡ Enter frequency value in hertz. Typical examples are: 60, 120, 1E3.

HP E1333A 3-Channel Universal Counter Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	Level 5	User Entry	Command(s) Used	Description
COUNTER	MONITOR	CHAN1				DISP:MON:CHAN 1;STAT ON	Monitor instrument operations on channel 1.
		CHAN2				DISP:MON:CHAN 2;STAT ON	Monitor instrument operations on channel 2.
		CHAN3				DISP:MON:CHAN 3;STAT ON	Monitor instrument operation on channel 3.
		AUTO				DISP:MON:CHAN AUTO;STAT ON	Monitor instrument operations on active channel.
	INPUT	LEVEL	CHAN1		voltage †	SENS1:EVEN:LEV *<value>*	Set trigger level voltage for channel 1.
			CHAN2		voltage †	SENS2:EVEN:LEV *<value>*	Set trigger level voltage for channel 2.
		SLOPE	CHAN1	POS		SENS1:EVEN:SLOP POS	Positive trigger slope for channel 1.
				NEG		SENS1:EVEN:SLOP NEG	Negative trigger slope for channel 1.
			CHAN2	POS		SENS2:EVEN:SLOP POS	Positive trigger slope for channel 2.
				NEG		SENS2:EVEN:SLOP NEG	Negative trigger slope for channel 2.
		COUPLE	AC			INP:COUP AC	AC-coupled input (channels 1 and 2 only).
			DC			INP:COUP DC	DC-coupled input (channels 1and 2).
		IMPED	50_OHM			INP:IMP 50	50Ω input resistance (channels 1 and 2 only).
			1_MOHM			INP:IMP 1e6	1MΩ input resistance (channels 1 and 2 only).
		ATTEN	0dB			INP:ATT 0	No input attenuation (channels 1 and 2 only).
			20dB			INP:ATT 20	20dB input attenuation (channels 1 and 2 only).
		FILTER	ON			INP:FILT ON	Input low-pass filter on (channels 1 and 2 only).
			OFF			INP:FILT OFF	Input low-pass filter off (channels 1 and 2 only).
	FREQ	CHAN1				TRIG:SOUR IMM;:MEAS1:FREQ?	Frequency measurement on channel 1.
		CHAN2				TRIG:SOUR IMM;:MEAS2:FREQ?	Frequency measurement on channel 2.
		CHAN3				TRIG:SOUR IMM;:MEAS3:FREQ?	Frequency measurement on channel 3.
	PERIOD	CHAN1				TRIG:SOUR IMM;:MEAS1:PER?	Period measurement on channel 1.
		CHAN2				TRIG:SOUR IMM;:MEAS2:PER?	Period measurement on channel 2.

(***Continued on next page***)

HP E1333A 3-Channel Universal Counter Menu

Menu Levels and Content

Level 1	Level 2	Level 3	Level 4	Level 5	User Entry	Command(s) Used	Description
(*Continued from previous page*)							
	TIMEINT	CHAN1				TRIG:SOUR IMM;:MEAS1:TINT?	Time interval measurement on channel 1.
		CHAN2				TRIG:SOUR IMM;:MEAS2:TINT?	Time interval measurement on channel 2.
	POS_PW	CHAN1				TRIG:SOUR IMM;:MEAS1:PWID?	Positive pulse width measurement on channel 1.
		CHAN2				TRIG:SOUR IMM;:MEAS2:PWID?	Positive pulse width measurement on channel 2.
	NEG_PW	CHAN1				TRIG:SOUR IMM;:MEAS1:NWID?	Negative pulse width measurement on channel 1.
		CHAN2				TRIG:SOUR IMM;:MEAS2:NWID?	Negative pulse width measurement on channel 2.
	RATIO	CHAN1				TRIG:SOUR IMM;:MEAS1:RAT?	Ratio of channel 1/channel 2.
		CHAN2				TRIG:SOUR IMM;:MEAS2:RAT?	Ratio of channel 2/channel 1.
	TOTALIZ	CHAN1	START			TRIG:SOUR IMM;:CONF1:TOT;:INIT1	Totalize on channel 1.
			READ			FETC1?	Display totalize count.
		CHAN2	START			TRIG:SOUR IMM;:CONF2:TOT;:INIT2	Totalize on channel 2.
			READ			FETC2?	Display totalize count.
	TEST					*TST?	Run self-test, display results (+0 = pass; any other number = fail).

† Enter voltage values in volts. Typical examples are: +3.5, -2, +500E-3.

Chapter 4
Triggering and System Status

About This Chapter

This chapter covers the use of the ECLTRG and TTLTRG* VXI backplane trigger lines and the HP E1406A Command Module's Trig In and Trig Out ports. Also covered is the structure of the status system used by Hewlett-Packard VXI instruments.

The main sections of this chapter include:

Using VXI Backplane Trigger Lines and Ports

Located on the P2 connector of the VXIbus backplane are trigger lines ECLTRG0 - ECLTRG1 and TTLTRG0* - TTLTRG7*. These lines are available for triggering, handshaking, timing, and so forth. The signal characteristics of these trigger lines and of the command module's Trig In port are shown in Figure 4-1.

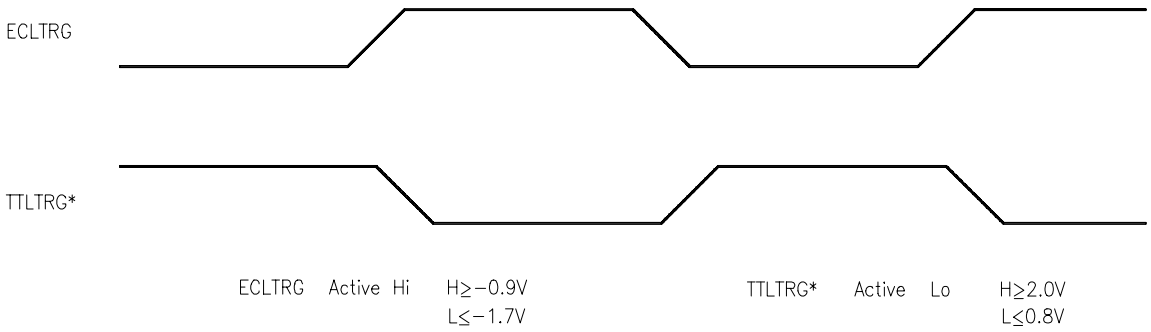


Figure 4-1. ECLTRG and TTLTRG* Signal Characteristics

Programming the Trigger Lines and the Trigger Ports

The programming sequence used to set up the trigger lines and trigger ports is shown in Figure 4-2. Detailed information on the commands used can be found in Chapter 5 of this manual.

Note

In the following commands, <*n*> is 0 or 1 when selecting a ECLTRG trigger line, and 0 to 7 when selecting a TTLTRG* trigger line. Commands in square brackets (**[ ]**) are implied commands and are, therefore, optional. The brackets are not part of the command and are not sent to the instrument.

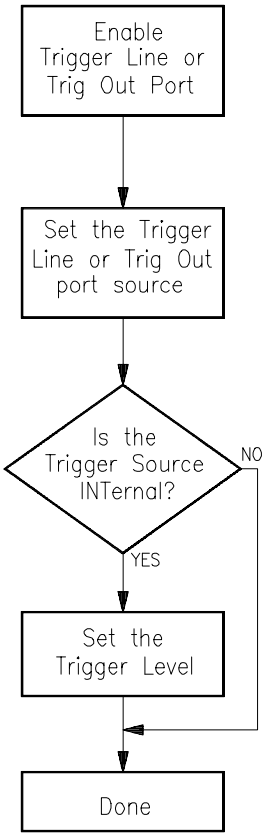


Figure 4-2. Backplane Trigger and Trig Out Port Configuration Sequence

Enabling Trigger Lines and the Trig Out Port

In order to use a trigger line or the Trig Out port, the trigger line or port must be enabled. This is done with the commands:

OUTPut:ECLTrg<*n*>[:STATe] 1 | ON	*Enables a ECL trigger line.*
OUTPut:TTLTrg<*n*>[:STATe] 1 | ON	*Enables a TTL trigger line.*
OUTPut:EXTernal[:STATe] 1 | ON	*Enables the Trig Out port.*

The reset condition for each of these commands is OFF. Therefore, a trigger line or the Trig Out port must be enabled before it can be used.

Setting the Trigger Source

Once the trigger line or the Trig Out port has been enabled, the source which drives the trigger line can be specified. The commands used are:

OUTPut:ECLTrg<*n*>:SOURce INT | EXT | NONE *Selects ECL trigger source.*

OUTPut:TTLTrg<*n*>:SOURce INT | EXT | NONE *Selects TTL trigger source.*

OUTPut:EXTernal:SOURce INT | ECLTrg<*n*> | TTLTrg<*n*> | NONE
Selects Trig Out port source.

When the trigger source is INT, the trigger level is set using the OUTPut...:LEVel commands covered in the next section. When the Trig Out trigger source is ECLTrg<*n*> or TTLTrg<*n*>, the port is driven by the specified trigger line. When the trigger source is EXT, the trigger is supplied through the Trig In port.

Notice that when the source is set, it remains set when the trigger state is set from ON to OFF. To disable a trigger line or the Trig Out port, first set the SOURce to NONE and then set STATe to OFF.

Setting the Trigger Level

When the trigger source is set to INT, the trigger level is controlled with the commands:

OUTPut:ECLTrg<*n*>:LEVel[:IMMediate] 0 | 1 | OFF | ON
Sets ECL trigger level.

OUTPut:TTLTrg<*n*>:LEVel[:IMMediate] 0 | 1 | OFF | ON
Sets TTL trigger level.

OUTPut:EXTernal:LEVel[:IMMediate] 0 | 1 | OFF | ON
Sets Trig Out trigger level.

The commands used to set the TTLTrg and Trig Out port levels use negative logic. Thus, when a 1 or ON level is specified, the trigger line or port is set to a TTL low voltage level.

Sending a Trigger Pulse

In certain VXI applications it may be necessary to send a single (trigger) pulse rather than continuously driving a trigger line. With the trigger line or the Trig Out port enabled (STATe ON) and the trigger source set to INT or NONE, you can send a single pulse using the commands:

OUTPut:ECLTrg<*n*>:IMMediate *Sends a pulse on an ECL trigger line.*

OUTPut:TTLTrg<*n*>:IMMediate *Sends a pulse on a TTL trigger line.*

OUTPut:EXTernal:IMMediate *Outputs a pulse at the Trig Out port.*

The pulse width is typically 60 µs.

Querying the Trigger State, Source, and Level

You can determine the current trigger state, source, and level settings by adding a question mark (?) to the command used to set that parameter. For example:

OUTPut:ECLTrg<*n*>[:STATe]? *Queries state of ECL trigger line.*

OUTPut:TTLTrg<*n*>:LEVel[:IMMediate]? *Queries level of TTL trigger line.*

OUTPut:EXTernal:SOURce? *Queries source of Trig Out port.*

Programming the Status System

This section discusses the structure of the Standard Commands for Programmable Instruments (SCPI) STATus system and how to program the Status Registers. An important feature of SCPI instruments is that they all implement Status Registers in the same way. The status system is explained in the following sections:

- General Status Register Model
  This section explains how Status Registers are structured in SCPI instruments. It also contains an example of how bits in the various registers change with different input conditions.

- Required Status Groups
  This section describes the minimum required Status Registers present in SCPI instruments. These Status Registers cover the most frequently used functions.

Example programs are also provided at the end of this chapter that illustrate how to use Service Requests to monitor events.

General Status Register Model

The generalized Status Register model shown in Figure 4-3 is the building block of the SCPI status system. This model consists of a Condition Register, Transition Filter, an Event Register, and an Enable Register. A set of these registers is called a **status group**.

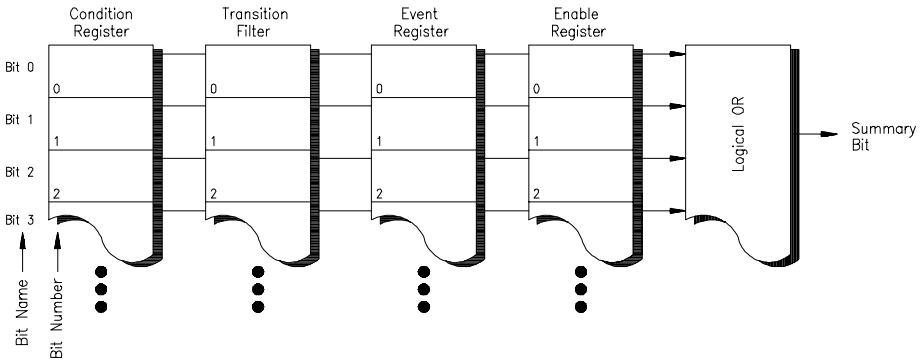


Figure 4-3. Generalized Status Register Model

When a status group is implemented in an instrument, it always contains all of the component registers. However, there is *not* always a corresponding command to read or write to every register.

Condition Register

The **Condition Register** continuously monitors the hardware and firmware status of the instrument. There is no latching or buffering for this register; it is updated in real time. Condition Registers are read-only.

If there is no command to read a particular Condition Register, it is simply invisible to you.

Transition Filter

The **Transition Filter** specifies which types of bit state changes in the Condition Register will set corresponding bits in the Event Register. Transition Filter bits may be set for positive transitions (PTR), negative transitions (NTR), or both. Positive means a condition bit changes from 0 to 1. Negative means a condition bit changes from 1 to 0. Transition Filters are read-write, and are unaffected by *CLS (clear status) or queries. They are set to instrument-dependent values at power on and after *RST (reset).

If there are no commands to access a particular Transition Filter, it has a fixed setting. This setting is specified in the instrument's programming guide or command dictionary. Most of our VXI instruments assign the Transition Filter to detect positive transitions only.

Event Register

The **Event Register** latches transition events from the Condition Register as specified by the Transition Filter. Bits in the Event Register are latched, and, once set, they remain set until cleared by a query or *CLS (clear status). There is no buffering; so while an event bit is set, subsequent events corresponding to that bit are ignored. Event Registers are read-only.

Enable Register

The **Enable Register** specifies which bits in the Event Register can generate a summary bit. The instrument logically ANDs corresponding bits in the Event and Enable Registers, and ORs all the resulting bits to obtain a summary bit. Summary bits are, in turn, recorded in another register, often the Status Byte. Enable Registers are read-write, and are *not* affected by *CLS (clear status). Querying Enable Registers does not affect them. There is always a command to read and write to the Enable Register of a particular status group.

An Example Sequence

Figure 4-4 illustrates the response of a single bit position in a typical status group for various settings. The changing state of the condition in question is shown at the bottom of the figure. A small binary table shows the state of the chosen bit in each Status Register at the selected times T1 – T5.

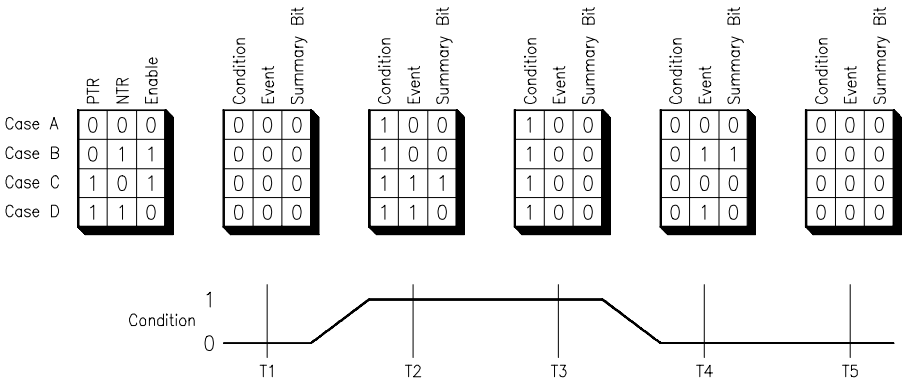


Figure 4-4. Typical Status Bit Changes in a Status Register

Required Status Groups

All SCPI instruments must implement a minimum set of status groups. Some instruments contain additional status groups, consistent with the general status register model. The minimum required status system is shown in Figure 4-5.

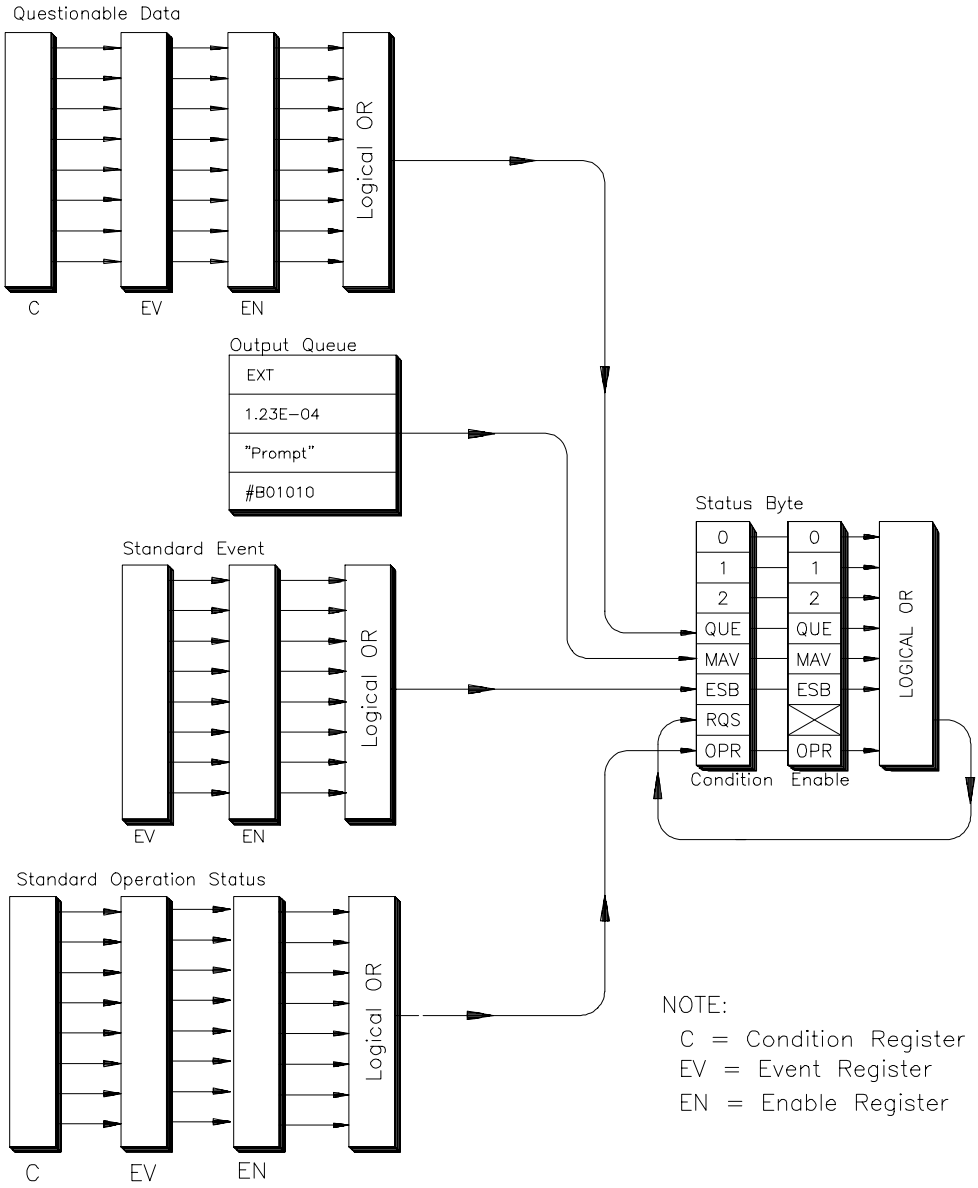


Figure 4-5. Minimum Required Status Register System

The Standard Operation Status and Questionable Data Groups are 16-bits wide, while Status Byte and Standard Event Groups are only 8-bits wide. In all 16-bit groups, the most significant bit (bit 15) is not used. Bit 15 always returns a zero. The commands that set and query bits in the Status Registers all use decimal integers. For example, you send *ESE 4 to set bit 2 of the Standard Event Enable Register. Similarly, a response of "8" to the query *ESE? indicates that bit 3 is set. The remainder of this chapter explains each status group in detail.

Status Byte Group

As Figure 4-6 indicates, the Status Byte is used to summarize information from all the other status groups. The Status Byte differs from the other groups in the way you read it and how its summary bit is processed.

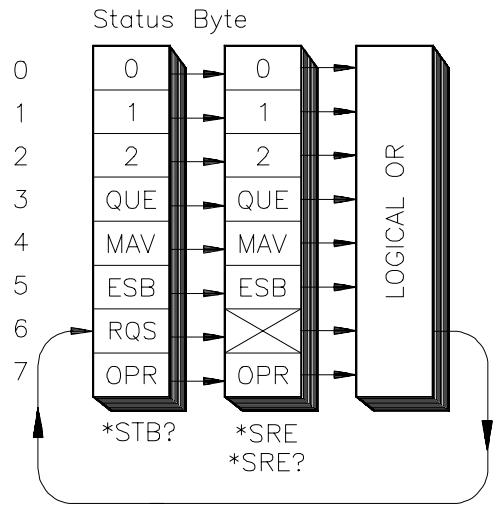


Figure 4-6. Status Byte Register

The Status Byte can be read using either the *STB? common command or by doing a SICL ireadstb function call. The ireadstb function reads the Status Byte from the device specified.

The Status Byte Summary bit actually appears in bit 6 (RQS) of the Status Byte. When bit 6 is set, it generates an SRQ interrupt. This interrupt is a low-level HP-IB message that signals the controller that at least one instrument on the bus requires attention.

There are some subtle differences between *STB? and ireadstb. You can use either method to read the state of bits 0-5 and bit 7. Bit 6 is treated differently depending on whether you use *STB? or ireadstb. With ireadstb, bit 6 returns RQS (request for service) which is cleared after the first ireadstb. *STB? returns the MSS (master state summary). This is the Summary bit of the Status Byte Register. It is like a condition bit and will return to zero only when all enabled bits in the Status Byte are zero. In general, use ireadstb inside interrupt service routines, not *STB?.

Note

In an SRQ interrupt service routine, you must clear the Event Register which caused the SRQ (for example, STAT:QUES:EVEN?, STAT:OPER:EVEN?, or *ESR?). ***Failure to do so will prevent future SRQs from arriving***.

The meaning of each bit in the Status Byte is explained in the following table.

Table 4-1. Status Byte Bit Definitions

Bit	Name	Description
0		Instrument dependent
1		Instrument dependent
2		Instrument dependent
3	QUE	Summary bit from Questionable Data
4	MAV	Messages available in Output Queue
5	ESB	Summary bit from Standard Event
6	RQS	Service request
7	OPR	Summary bit from Standard Operation Status

Example commands using the Status Byte and Status Byte Enable Registers:

*SRE 16 *Generate an SRQ interrupt when messages are available.*

*SRE? *Find out what events are enabled to generate SRQ interrupts.*

*STB? *Read and clear the Status Byte Event Register.*

Standard Event Status Group

The **Standard Event Status Group** is frequently used and is one of the simplest. The unique aspect of Standard Event is that you program it using common commands, while you program all other status groups through the STATus subsystem. Standard Event consists of only two registers: the Standard Event's Event Register and the Standard Event's Enable Register. Figure 4-7 illustrates the structure of Standard Event.

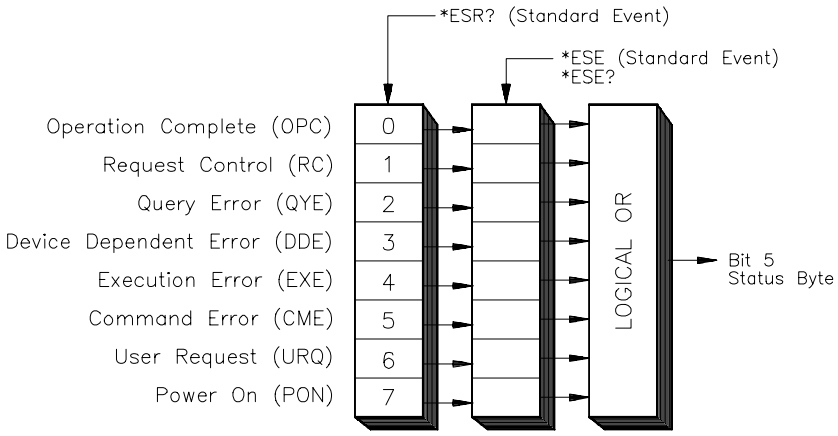


Figure 4-7. Standard Event Status Group

Example commands using Standard Event Registers:

*ESE 48 *Generate a Summary bit on execution or command errors.*

*ESE? *Query the state of the Standard Event's Enable Register.*

*ESR? *Query the state of the Standard Event's Event Register.*

Standard Operation Status Group

The **Standard Operation Status Group** provides information about the state of the measurement systems in an instrument. This status group is accessed through the STATus subsystem. Standard Operation Status includes a Condition Register, Event Register, and an Enable Register. As a beginner, you will rarely need to use this group. Figure 4-8 illustrates the structure of Standard Operation Status.

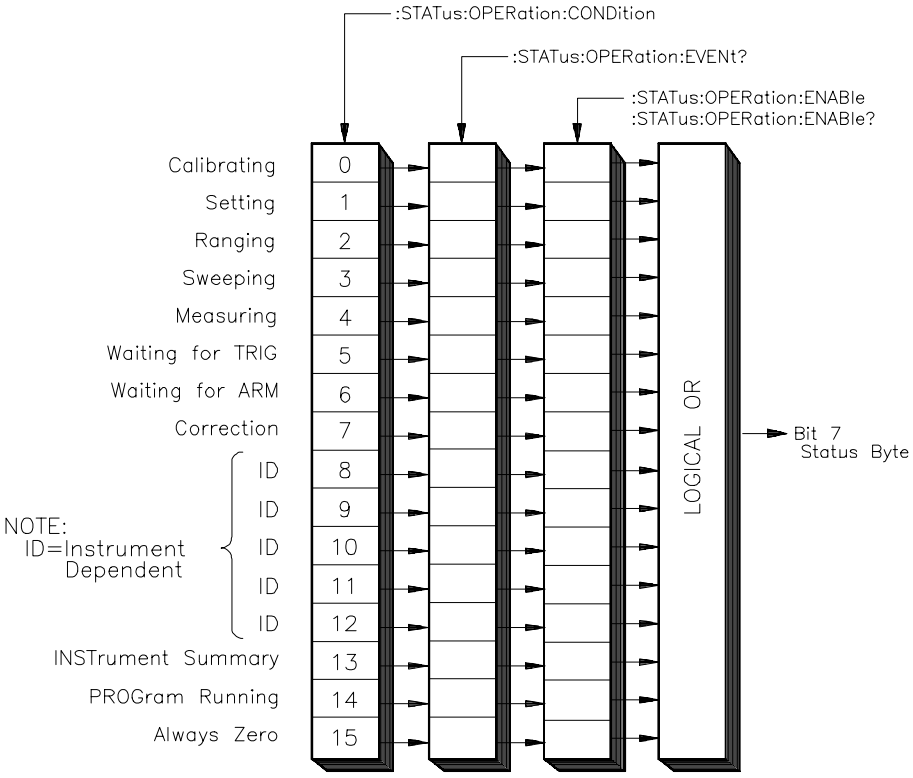


Figure 4-8. Standard Operation Status Group

Questionable Data Group

The **Questionable Data Status Group** provides information about the quality of instrument output and measurement data. Questionable Data is accessed through the STATus subsystem. As a beginner, you will rarely need to use this status group. Figure 4-9 illustrates the structure of Questionable Data.

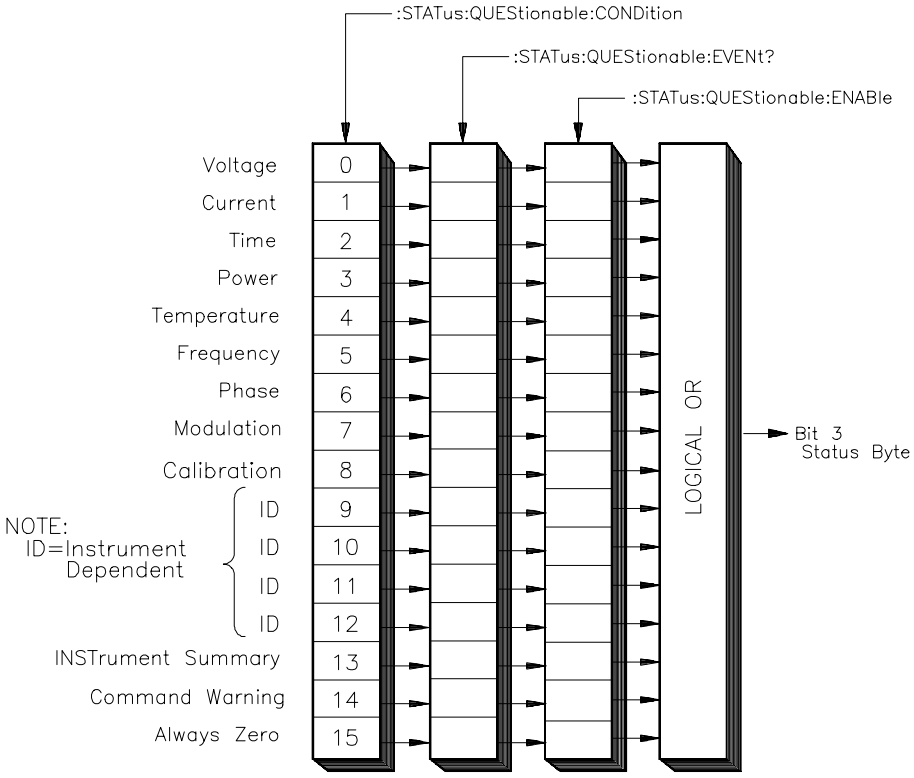


Figure 4-9. Questionable Data Status Group

Status System Programming Examples

This section contains two example programs that use the status system and common commands to monitor when data is available from an instrument and when an error has occurred. Both programming examples are written in C and use the Standard Instrument Control Library (SICL) for I/O operations. The example programs use SCPI (Standard Commands for Programmable Instruments) commands to communicate with the status system. Thus, the instruments must either be message-based or have a SCPI interpreter, such as an HP E1406A Command Module or the SICL **iscpi** interface.

Handling SRQs

The following is a general procedure for handling SRQs:

- Define the SRQ handler to do the following:
  - Read the Status Byte using **ireadstb**. **ireadstb** returns the RQS (request for service) bit in bit 6 of the status byte. After issuing an **ireadstb**, RQS is cleared indicating that the Service Request is being acknowledged. A new SRQ will not be issued unless RQS is cleared. Using *STB? will return the Master State Summary in bit 6 and does not affect RQS, therefore this should not be used in a SRQ handler.
  - Check the status byte to determine which status group(s) requires service.
  - For each status group that requires service, read the Event Register of that status group to determine what caused the SRQ to be generated. It is necessary to clear the Event Register so that if a new event occurs a new SRQ will be generated.
  - Take some action after determining which event caused the SRQ. The action taken is determined by evaluating the contents of the Event Register.
- Enable SRQ Handler in SICL with **ionsrq**.
- Make sure that all the Enable Masks in all the Status Enable Registers are set to the proper values to propagate the Summary bit(s) to the Status Byte. An SRQ is only generated if the MSS (Master State Summary) bit in the status byte is set.

Using Message Available (MAV) Bits

Message Available (MAV) bits can be used to determine when data is available. The following example program sets up an SRQ handler to be called when there is data in the output queue. The program then prompts for SCPI commands. If the SCPI command results in data in the output queue (such as a query command), then the SRQ handler is called and the data is printed.

The following summarizes the procedure used:

- Define an SRQ handler to do the following:
  - Read the Status Byte using **ireadstb**. **ireadstb** returns the RQS (request for service) bit in bit 6 of the status byte. After issuing a **ireadstb**, RQS is cleared indicating that the Service Request is being acknowledged. A new SRQ will not be issued unless RQS is cleared. Using *STB? will return the Master State Summary in bit 6 and does not affect RQS.
  - Check if the MAV bit (bit 4) is set to indicate that a message is available. If the MAV bit is set, then a message is available and the SRQ handler can process the message. In this example, the output queue is read using **iscanf**.
- Enable SRQ Handler in SICL with **ionsrq**.
- Enable Message Available (MAV) bit in the Status Byte Enable Register (e.g. *SRE 16). This will cause an SRQ to arrive when there is a message in the output queue (for example, data is available to be read).

Example Program

```
/* The following program provides an interactive command line interface  */
/* to send SCPI commands to SCPI compatible instruments.  */
/* This utilizes the MAV bit of the Status Byte in order to determine if the  */
/* instrument is returning any output.  */
#include <sicl.h>
#include <stdio.h>

/* Theses are Masks for the Status Byte */
/* all bits start at bit 0 */
#define MAV_MASK 0x10      /* MAV - bit 4 */

/* This is the SRQ handler to check for Message Available (MAV) */
void srq_hdlr( INST id) {
    unsigned char stb;
    char buf[255];
    int esr;
    int errnum;
    char errmsg[100];

    /* read the status byte to determine what caused the SRQ.  */
    /* Note: use ireadstb instead of *STB? because you want to  */
    /* clear RQS instead of reading the MSS bit in the status byte.  */
   ireadstb(id, &stb);
   /* check if MAV caused the SRQ */
  if( MAV_MASK == (stb & MAV_MASK))
  {
```

Continued on next page

```
        /* message is available so read in the result. */
       iscanf( id, "%t", buf);
       printf("%s", buf);
   }
}

void main(){
   INST id;
   char addr[80];
   char cmd[255];
   int opc;
   int idx;

   printf("This program provides an interactive environment for SCPI \n");
   printf("compatible instruments.  \n\n");
   printf("Enter the SICL address of the instrument to open.\n");
   printf("for example:  iscpi,24)\n");

   gets(addr);

   /* install error handler */
   ionerror( I_ERROR_EXIT);

   /* open the instrument specified by the user */
   id = iopen(addr);
   itimeout( id, 20000);/* 20 second timeout */

   /* set up SRQ handler */
   ionsrq( id, srq_hdlr);

   /* enable MAV (bit 4) in status byte to cause an SRQ */
   iprintf( id, "*SRE %d\n", MAV_MASK );

   /* make sure *SRE finished */
   ipromptf( id, "*OPC?\n", "%d", &opc);/* opc value not used */

   printf("\nEnter SCPI Commands/Queries to Instrument at %s\n", addr);
   printf("  (press return to exit)\n\n");
   while(1)
   {
      while(0 == gets(cmd));
      if( 0 == strlen(cmd))
            break;          /* quit sending SCPI Commands */

      /* send command */
      iprintf(id, "%s\n", cmd);

      /* check cmd for a '?', if found assume it is a query */
      for(idx=0; idx<strlen(cmd); idx++)
           if( '?' == cmd[idx])
           {
                /* wait up to 1 minute for srq handler */
                if( 0 != iwaithdlr(60000))
                {
                     printf("ERROR: Failed to process Query\n");
                }
```

Continued on next page

```
                break;
            }
    }/* while - there are commands to send */
    /* remove the handler */
    ionsrq( id, 0);

    /* close the session */
    printf("\nClosing Instrument at %s\n", addr);
    iclose(id);
}
```

Using a Service Request (SRQ)

A Service Request (SRQ) can be used to detect errors. The following example program sets up an SRQ handler to be called when SCPI errors are detected using the Standard Event Status Register. The program then prompts for SCPI commands. If the SCPI command results in data in the output queue (such an query command) or an error, then the SRQ handler is called and the data is printed.

The following summarizes the procedure used:

- Define a SRQ Handler which does the following:
  - Read the Status Byte using **ireadstb**. **ireadstb** returns the RQS (request for service) bit in bit 6 of the status byte. After issuing a **ireadstb**, RQS is cleared indicating that the Service Request is being acknowledged. A new SRQ will not be issued unless RQS is cleared. Using *STB? will return the Master State Summary in bit 6 and does not affect RQS.
  - Check if the MAV bit (bit 4) is set to indicate that a message is available. If the MAV bit is set, then a message is available and the SRQ handler can process the message. In this example, the output queue is read using **iscanf**.
  - Check if the Standard Event Status Summary bit (bit 5) is set. If the bit is set, then read the Standard Event Status Group's Event Register to determine which event(s) caused the SRQ. Check for Command Error (bit 5), Execution Error (bit 4), Device Dependent Error (bit 3), or Query Error (bit 2). If found, read the error queue with SYST:ERR? to print out error messages.
- Enable SRQ Handler in SICL with **ionsrq**.
- Enable MAV bit (Message Available Bit) and Standard Event Status Register Summary bit in the Status Byte Enable Register (for example, *SRE 48). This will cause an SRQ to arrive when there is a message in the output queue or when the summary bit is set in the standard event status register.
- Enable the Command Error, Execution Error, Device Dependent Error, and Query Error Enable bits in the Standard Event Status Enable Register (e.g. *ESE 60). This will cause the Summary bit of the Standard Event Status Register to be set when an error occurs.

Example Program

```
/* The following program provides an interactive command line interface */
/* to send SCPI commands to SCPI compatible instruments. */
/* This utilizes the MAV bit of the Status Byte in order to determine if */
/* the instrument is returning any output.  It also automatically */
/* displays any error conditions that may result by querying the Standard */
/* Event Status Register. */
#include <sicl.h>
#include <stdio.h>

/* Theses are Masks for the Status Byte */
/* all bits start at bit 0 */
#define MAV_MASK 0x10      /* MAV - bit 4 */
#define ESR_MASK 0x20      /* ESR summary - bit 5 */

/* These are Masks for the Standard Event Status Register */
/* all bits start at bit 0 */
#define QRY_ERR_MASK 0x04  /* query error - bit 2 */
#define DEV_ERR_MASK 0x08  /* device dependent error - bit 3 */
#define EXE_ERR_MASK 0x10  /* execution error - bit 4 */
#define CMD_ERR_MASK 0x20  /* command error - bit 5 */

/* This is the SRQ handler to check for Message Available (MAV) */
/* or any error conditions */
void srq_hdlr( INST id)
{
   unsigned char stb;
   char buf[255];
   int esr;
   int errnum;
   char errmsg[100];


   /* read the status byte to determine what caused the SRQ. */
   /* Note: use ireadstb instead of *STB? because we want to */
   /* clear RQS instead of reading the MSS bit in the status byte. */
  ireadstb(id, &stb);

  /* check if MAV caused the SRQ */
  if( MAV_MASK == (stb & MAV_MASK))
  {
      /* message is available so read in the result */
      iscanf( id, "%t", buf);
      printf("%s", buf);
  }
  else /* check if Standard Event Status */
  if( ESR_MASK == (stb & ESR_MASK))
  {
      /* read the standard event register to determine what caused the ESR */
      /* summary bit to be set. This is necessary in order to get future */
      /* SRQ's from the Standard Event status group. */
       ipromptf(id, "*ESR?\n", "%d\n", &esr);

       /* check if an error caused the summary bit to get set */
       if( (CMD_ERR_MASK == (esr & CMD_ERR_MASK )) ||
         (EXE_ERR_MASK == (esr & EXE_ERR_MASK )) ||
         (DEV_ERR_MASK == (esr & DEV_ERR_MASK )) ||
         (QRY_ERR_MASK == (esr & QRY_ERR_MASK ))    )
      {
```

Continued on next page

```
            /* an error occurred, read the error queue to get the error */
            errnum = -1;
            while( errnum != 0)
            {
                ipromptf( id, "SYST:ERR?\n", "%d,%t", &errnum, errmsg);
                if( errnum != 0)
                   printf("%d,%s", errnum, errmsg);
            }
        }
    }
}
void main()
{
   INST id;
   char addr[80];
   char cmd[255];
   int opc;
   int idx;

   printf("This program provides an interactive environment for SCPI \n");
   printf("compatible instruments.  \n\n");
   printf("Enter the SICL address of the instrument to open.\n");
   printf("for example:  iscpi,24)\n");

   gets(addr);


   /* install error handler */
   ionerror( I_ERROR_EXIT);

   /* open the instrument specified by the user */
   id = iopen(addr);
   itimeout( id, 20000);       /* 20 second timeout */

   /* set up SRQ handler */
   ionsrq( id, srq_hdlr);

   /* enable MAV (bit 4) and Standard Event Status Summary (bit 5)
    * in status byte to cause an SRQ */
   iprintf( id, "*SRE %d\n", MAV_MASK | ESR_MASK);

    /* enable ERROR Bits to generate a ESR summary message */
   iprintf( id, "*ESE %d\n", CMD_ERR_MASK | EXE_ERR_MASK |
            DEV_ERR_MASK | QRY_ERR_MASK);

   /* make sure *SRE and *ESE finished */
   ipromptf( id, "*OPC?\n", "%d", &opc);       /* opc value not used */
   printf("\nEnter SCPI Commands/Queries to Instrument at %s\n", addr);
   printf("  (press return to exit)\n\n");

   while(1)
   {
      while(0 == gets(cmd));
      if( 0 == strlen(cmd))
           break;          /* quit sending SCPI Commands */
```

Continued on next page

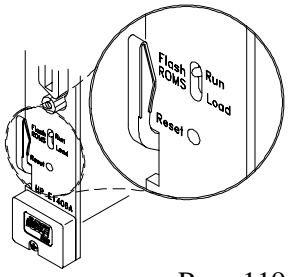
```
        /* send command */
        iprintf(id, "%s\n", cmd);

        /* check cmd for a '?', if found assume it is a query */
        for(idx=0; idx<strlen(cmd); idx++)
              if( '?' == cmd[idx])
              {
                    /* wait up to 1 minute for srq handler */
                    if( 0 != iwaithdlr(60000))
                    {
                          printf("ERROR: Failed to process Query\n");
                    }
                    break;
              }
    }       /* while - there are commands to send */
    /* remove the handler */
    ionsrq( id, 0);
    /* close the session */
    printf("\nClosing Instrument at %s\n", addr);
    iclose(id);
}
```

Chapter 5
HP E1406A Command Reference

About This Chapter

This chapter describes the **Standard Commands for Programmable Instruments** (SCPI) command set and the **IEEE 488.2 Common Commands** for the System instrument and the Loader instrument. The System instrument is part of the HP E1406A Command Module's internal control processor and is, therefore, always present in the command module.

The Flash ROMS Run/Load switch on the front of the HP E1406A Command Module must be in the "Run" position to access the System instrument. The Run/Load switch must be in the "Load" position to access the Loader instrument. This chapter contains the following sections:



Command Types

Commands are separated into two types: IEEE 488.2 Common Commands and SCPI Commands.

Common Command Format

The IEEE 488.2 standard defines the common commands that perform functions like reset, self-test, status byte query, and so forth. Common commands are four or five characters in length, always begin with an asterisk (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of common commands are:

*RST, *ESE <*mask*>, *STB?

SCPI Command Format

SCPI commands perform functions like closing switches, making measurements, and querying instrument states or retrieving data. A subsystem command structure is a hierarchical structure that usually consists of a top level (or root) command, one or more lower level commands, and their parameters.

The following example shows part of a typical subsystem:

```
[ROUTe:]
    CLOSe <channel_list>
    SCAN <channel_list>
        MODE?
```

[ROUTe:] is the root command, CLOSe and SCAN are second level commands with parameters, and :MODE? is a third level command. [ROUTe:] is also an implied command and is, therefore, optional.

Command Separator

A colon (**:**) always separates one command from the next lower level command as shown below:

```
ROUTe:SCAN:MODE?
```

Colons separate the root command from the second level command (ROUTe:SCAN) and the second level from the third level (SCAN:MODE?).

Abbreviated Commands

The command syntax shows most commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, you may send the entire command. The instrument will accept either the abbreviated form or the entire command.

For example, if the command syntax shows DIAGnostic, then DIAG and DIAGNOSTIC are both acceptable forms. Other forms of DIAGnostic, such as DIAGN or DIAGNOS will generate an error. You may use upper or lower case letters. Therefore, DIAGNOSTIC, diagnostic, and DiAgNoStIc are all acceptable.

Implied Commands

Implied commands appear in square brackets ([ ]) in the command syntax. (The brackets are not part of the command, and are not sent to the instrument.) Suppose you send a second level command but do not send the preceding implied command. In this case, the instrument assumes you intend to use the implied command and it responds as if you had sent it. Examine the [SOURce:] subsystem shown below:

```
[SOURce:]
    PULSe
        :COUNt
        :COUNt?
        :PERiod
        :PERiod?
```

The root command [SOURce:] is an implied command. To set the instrument's pulse count to 25, you can send either of the following command statements:

```
SOUR:PULS:COUN 25      or      PULS:COUN 25
```

Variable Command Syntax

Some commands have what appears to be a variable syntax. For example:

DIAG:INT:SETup[*n*]? *and* SYST:COMM:SERial[*n*]:BAUD?

In these commands, the "*n*" is replaced by a number. No space is left between the command and the number because the number is not a parameter. The number is part of the command syntax. The purpose of this notation is to save a great deal of space in the command reference. In the case of …SETup[*n*], [*n*] could range from 1 through 7. In …SERial[*n*]…, [*n*] can be from 0 through 7. You can send the command without the [*n*] and a default value will be used by the instrument. Some examples:

DIAG:INT:SET2?, DIAG:INT:PRI2 5, SYST:COMM:SER1:BAUD 9600

Parameter Types

The following list contains explanations and examples of parameter types you will see later in this chapter.

- **Arbitrary Block Program Data parameters** are used to transfer blocks of data in the form of bytes. The block of data bytes is preceded by a preamble which indicates either 1) the number of data bytes which follow, or 2) that the following data block will be terminated upon receipt of a New Line message with the EOI signal true. The syntax is:

  ### Definite Length Block

  #<*non-zero digit*><*digit*(*s*)><*data byte*(*s*)>

  Where the value of <*non-zero digit*> equals the number of <*digit*(*s*)>. The value of <*digit*(*s*)> taken as a decimal integer indicates the number of <*data byte*(*s*)> in the block.

  ### Indefinite Length Block

  #0<*data byte*(*s*)><NL^END>

  Examples of sending 4 data bytes:

  #14<*byte*><*byte*><*byte*><*byte*>
  #3004<*byte*><*byte*><*byte*><*byte*>
  #0<*byte*><*byte*><*byte*><*byte*><NL^END>

- **Boolean Parameters** represent a single binary condition that is either true or false (for example, ON, OFF, 1, 0). Any non-zero value is considered true.

- **Discrete Parameters** selects from a finite number of values. These parameters use mnemonics to represent each valid setting. An example is the OUTPut:EXTernal:SOURce <*source*> command where *source* can be INTernal, ECLTrg0, ECLTrg1, TTLTrg0, TTLTrg1, TTLTrg2, TTLTrg3, and so on.

- **Numeric Parameters** are commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation (for example, 123, 123E2, –123, –1.23E2, .123, 1.23E-2, 1.23000E–01). Special cases include MIN, MAX, DEFault, and INFinity.

The "Comments" section within the Command Reference will state whether a numeric parameter can also be specified in hex (#H7B), octal (#Q173), and/or binary (#B1111011).

- **Optional Parameters** are parameters shown within square brackets (**[ ]**), and are optional. (Note that the brackets are not part of the command, and are not sent to the instrument.) If you do not specify a value for an optional parameter, the instrument chooses a default value. For example, consider the ARM:COUNt? [<MIN | MAX>] command. If you send the command without specifying a parameter, the present ARM:COUNt value is returned. If you send the MIN parameter, the command returns the minimum count available. If you send the MAX parameter, the command returns the maximum count available. Be sure to place a space between the command and the parameter.

Linking Commands

Linking IEEE 488.2 Common Commands with SCPI Commands. Use a semicolon (;) between the commands. For example:

```
*RST;OUTP ON      or      TRIG:SOUR HOLD;*TRG
```

Linking Multiple SCPI Commands. Use both a semicolon and a colon between the commands. For example:

```
ARM:COUN 1;:TRIG:SOUR EXT
```

SCPI Command Reference

This section describes the SCPI commands for the System instrument and Loader instrument. Commands are listed alphabetically by subsystem and also within each subsystem.

DIAGnostic

The DIAGnostic subsystem allows control over the System instrument's internal processor system (:BOOT and :INTerrupt), access to the Loader instrument, allocation and contents of user RAM and disc volume RAM (:NRAM and :RDISk), and allocation of the built-in serial interface (DIAG:COMM:SER[0]:OWN).

Subsystem Syntax

```
DIAGnostic
    :BOOT
        :COLD
        [:WARM]
    :COMMunicate
        :SERial[0]
            [:OWNer] <owner> | SYSTem | IBASic | NONE
            [:OWNer]?
        :SERial[n]
            :STORe
    :DOWNload
        :CHECked
            [:MADDress] <address>,<data>
            :SADDress <address>,<data>
        [:MADDress] <address>,<data>
        :SADDress <address>,<data>
    :DRAM
        :AVAilable?
        :CREate <size> | MIN | MAX,<num_drivers> | MIN | MAX | DEF
        :CREate? [<MIN | MAX>,<MIN | MAX | DEF>]
    :DRIVer
        :INSTall
        :LIST
            [:ALL]?
            :FROM?
            :RAM?
            :ROM?
        :LOAD <driver_block>
            :CHECked <driver_block>
    :FROM
        :AVAilable?
        :CREate <num_drivers>
        :CREate?
        :SIZE?
    :INTerrupt
        :ACTivate <mode> | 0 | 1 | OFF | ON
        :PRIority[n] <level> | MIN | MAX | DEF
        :PRIority[n]?
        :RESPonse?
        :SETup[n] <mode> | 0 | 1 | OFF | ON
        :SETup[n]?
    :NRAM
        :ADDRess?
        :CREate <size> | MIN | MAX
        :CREate? [MIN | MAX]
    :PEEK? <address>,<width>
    :POKE <address>,<width>,<data>
```

```
:RDISk
    :ADDRess?
    :CREate <size> | MIN | MAX
    :CREate? [MIN | MAX]
:UPLoad
    [:MADDress]? <address>,<byte_count>
    SADDress? <address>,<byte_count>
```

:BOOT:COLD

DIAGnostic:BOOT:COLD causes the System instrument to restart (reboot). Configurations stored in non-volatile memory and RS-232 configurations are reset to their default states:

- DRAM, NRAM, and RDISk memory segments are cleared.
- Serial Interface parameters for the internal serial interface and for any plug-in serial cards (HP E1324A) that are in the command module's servant area are set to:
  - BAUD 9600
  - BITS 8
  - PARity NONE
  - SBITs 1
  - DTR ON
  - RTS ON
  - PACE XON
- Serial 0 Owner = system

Note Resetting the serial interface parameters takes about 0.01 seconds for the built-in serial port and 0.75 seconds per serial plug-in card. While this is taking place the System instrument will still respond to serial polls. If you are using a serial poll to determine when the cold boot cycle is complete, you should insert a delay of 1 second per plug-in serial card (HP E1324A) before polling the System instrument. This will prevent incorrectly determining that the System instrument has completed its boot cycle.

Comments

- The System instrument goes through its power-up self tests.
- **Related Commands:** DIAG:BOOT[:WARM]

Example **Reboot the System Instrument (cold)**

```
DIAG:BOOT:COLD                    Force boot.
```

:BOOT[:WARM]

DIAGnostic:BOOT[:WARM] causes the System instrument to restart (reboot) using the current configuration stored in non-volatile memory. The effect is the same as cycling power.

Comments

- The System instrument goes through its power-up self tests.
- The Non-volatile system state is used for configuration wherever applicable.
- DRAM, NRAM, and RDISk memory segments remain intact.
- **Related Commands:** DIAG:BOOT:COLD

Example

Boot the System Instrument (warm)

```
DIAG:BOOT                    Force boot. Note that :WARM is implied.
```

:COMMunicate :SERial[0][:OWNer]

DIAGnostic:COMMunicate:SERial[0][:OWNer] ***<owner>*** allocates the built-in serial interface to the System instrument (SYSTem), the optional IBASIC interpreter (IBASic), or to neither (NONE).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*owner*>	discrete	SYSTem | IBASic | NONE	none

Comments

- While the serial interface is allocated to the command module (SYSTem), it can function as the mainframe user interface when connected to a terminal or computer running terminal emulation software.
- When the built-in serial interface is allocated to IBASic, it is controlled only by IBASIC. The serial interface is given a select code of 9, and any RS-232 device connected to the command module's RS-232 port is programmed accordingly. Note that when IBASIC owns the serial interface there is no "front panel" interface to the system.
- If the built-in serial interface is not needed, specifying NONE will release memory for use by other instruments.
- Once the new serial interface owner has been specified (DIAG:COMM:SER:OWN), the change will *not* take effect until you reboot (warm) the system.
- **Related Commands:** DIAG:COMM:SER[0][:OWN]?

Example

Give the Serial Interface to IBASIC

```
DIAG:COMM:SER  IBAS          Note that 0 (zero) and :OWNer are implied.
DIAG:BOOT:WARM               Complete the allocation.
```

:COMMunicate :SERial[0][:OWNer]?

DIAGnostic:COMMunicate:SERial[0][:OWNer]? returns the current "owner" of the built-in serial interface. The values returned will be; SYST, IBAS, or NONE.

Comments

- **Related Commands:** DIAG:COMM:SER[0][:OWN]

Example

Determine Which Instrument has the Serial Interface

DIAG:COMM:SER?	*Note that 0 (zero) and :OWNer are implied.*
enter statement	*Statement returns the string SYST, IBAS, or NONE.*

:COMMunicate :SERial[*n*]:STORe

DIAGnostic:COMMunicate:SERial[*n*]:STORe stores the serial communications parameters (for example, BAUD, BITS, PARity, and so on) into Non-volatile storage for the serial interface specified by [*n*] in SERial[*n*].

Comments

- Until DIAG:COMM:SER[*n*]:STORe is executed, communication parameter values are stored in *volatile* memory, and a power failure will cause the settings to be lost.
- DIAG:COMM:SER[*n*]:STORe stores the serial interface's serial communications parameters. Card number 0 (in place of [*n*]) specifies the command module's built-in interface while 1 through 7 specifies one of up to seven HP E1324A B-size plug-in serial interface modules. Be aware that the HP E1324A module stores its settings in an on-board EEROM. This EEROM write cycle takes nearly one second to complete. Wait for this operation to complete before attempting to use that serial interface.
- The HP E1324A's EEROM used to store its serial communication settings has a finite lifetime of approximately 10,000 write cycles. Even if your application program sent the DIAG:COMM:SER[*n*]:STORe command once every day, the lifetime of the EEROM would still be over 27 years. Be careful that your application program sends the DIAG:COMM:SER[*n*]:STORe command to an HP E1324A no more often than is necessary.
- **Related Commands:** All SYST:COMM:SER[*n*]… commands.

Example

Store the Serial Communications Settings in the Third HP E1324A

```
DIAG:COMM:SER3:STOR
```

:DOWNload:CHECked [:MADDress]

DIAGnostic:DOWNload:CHECked[:MADDress] **<*address*>,<*data*>** writes data into a non-volatile user RAM segment starting at address using error correction. The user RAM segment is allocated by the DIAG:NRAM:CREate or DIAG:DRAM:CREate command.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	0 to 16,777,215 (#HFFFFFE)	none
<*data*>	arbitrary block program data	See "Parameter Types" on page 121	none

Comments

- This command is typically used to send a block of data to a block of user RAM. It is the only way to send binary data to multiple addresses over a serial (RS232C) line.
- **CAUTION:** Be certain that *all* of the data you download will be contained entirely within the allocated NRAM segment. Writing data outside of the NRAM segment will disrupt the operation of the command module. Most computers terminate an OUTPUT, PRINT, or WRITE statement with a carriage return or carriage return and line feed. These End-Of-Line characters must be either accounted for (NRAM segment sized to accommodate them), or suppressed using an appropriate IMAGE or FORMAT statement. Listed below are some helpful methods:
  - Size the NRAM segment a little larger than the expected data block.
  - Control the End-Of-Line characters with format statements.
  - Use the *Definite Length Arbitrary Block Program Data* format (see example on page 121) to send your data rather than the *Indefinite Length Arbitrary Block Program Data* format.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats. DOWNload is done by word (16-bit) access so *address* must be even.
- **Be certain that *address* specifies a location within the user RAM segment allocated using DIAG:NRAM:CREate if you are downloading a configuration table.** DIAG:DOWNload can change the contents of System RAM, causing unpredictable results.
- This command can also be used to write data to a device with registers in the A16 address space. See DIAGnostic:DOWNload:SADDress.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:ADDRess?, DIAG:UPLoad[:MADDress]?, VXI:CONF:CTABle, VXI:CONF:DCTable, VXI:CONF:ITABle, VXI:CONF:MTABle

Byte Format Each byte sent with this command is expected to be in the following format:

Bit #	7	6	5	4	3	2	1	0
	Control Bit	Check Bits			Data Bits			

- ***Control Bit*** is used to indicate the serial driver information such as clear, reset, or end of transmission. This bit is ignored by the regular 488.2 driver. The control bit should be one for regular data.
- ***Check Bits*** are used to detect and correct a single bit error. The control bit is not included in the check. The check bits are a Hamming single bit error correction code, as specified by the following table:

Data Value	Check Bits
0	0
1	7
2	6
3	1
4	5
5	2
6	3
7	4
8	3
9	4
10	5
11	2
12	6
13	1
14	0
15	7

- ***Data Bits*** are the actual data being transferred (four bits at a time). Each word to be written requires four data bytes for transmission. The significance of the data is dependent on the order received. The first data byte received contains the most significant nibble of the 16-bit word to be written (bits 15-12).

  The next data byte received contains the least significant nibble of the most significant byte of the word (bits 11-8). The third data byte received contains the most significant nibble of the least significant byte of the word (bits 7-4). The fourth data byte received contains the least significant nibble of the least significant byte of the word to be written (bits 3-0). Once all four bytes have been received the word will be written.

:DOWNload:CHECked :SADDress

DIAGnostic:DOWNload:CHECked:SADDress **<*address*>,<*data*>** writes *data* to Non-volatile user RAM at a single address specified by *address* using error correction. It can also write to devices with registers in the A16 address space.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	0 to 16,777,215 (#HFFFFFE)	none
<*data*>	arbitrary block program data	See "Parameter Types" on page 121	none

Comments

- This command is typically used to send data to a device which accepts data at a single address. It is the only way to send binary data to single addresses over a serial (RS232C) line.
- Most computers terminate an OUTPUT, PRINT, or WRITE statement with a carriage return or carriage return and line feed. These End-Of-Line characters must be either accounted for (NRAM segment sized to accommodate them), or suppressed using an appropriate IMAGE or FORMAT statement. Listed below are some helpful methods:
  - Control the End-Of-Line characters with format statements.
  - Use the *Definite Length Arbitrary Block Program Data* format (see example on page 121) to send your data rather than the *Indefinite Length Arbitrary Block Program Data* format.
- A register address in A16 address space can be determined by:

  $1FC000_{16}$ + (LADDR * 64) + register_number

  Where $1FC000_{16}$ is the base address in the command module A16 space, LADDR is the device logical address, 64 is the number of address bytes per device, and register_number is the register to which the data is written.

  If the device is an A24 device, the address can be determined using the VXI:CONFigure:DLISt? command to find the base address in A24, and then adding the register_number to that value. A24 memory between address 200000_{16} and address $E00000_{16}$ is directly addressable by the command module.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats. DOWNload is done by word (16-bit) access so *address* must be even.
- **Related Commands:** DIAG:UPLoad:SADDress?

Byte Format Each byte sent with this command is expected to be in the following format:

Bit #	7	6	5	4	3	2	1	0
	Control Bit	*Check Bits*			*Data Bits*			

- ***Control Bit*** is used to indicate the serial driver information such as clear, reset, or end of transmission. This bit is ignored by the regular 488.2 driver. The control bit should be one for regular data.
- ***Check Bits*** are used to detect and correct a single bit error. The control bit is not included in the check. The check bits are a Hamming single bit error correction code, as specified by the following table:

Data Value	Check Bits
0	0
1	7
2	6
3	1
4	5
5	2
6	3
7	4
8	3
9	4
10	5
11	2
12	6
13	1
14	0
15	7

- ***Data Bits*** are the actual data being transferred (four bits at a time). Each word to be written requires four data bytes for transmission. The significance of the data is dependent on the order received. The first data byte received contains the most significant nibble of the 16-bit word to be written (bits 15-12).

  The next data byte received contains the least significant nibble of the most significant byte of the word (bits 11-8). The third data byte received contains the most significant nibble of the least significant byte of the word (bits 7-4). The fourth data byte received contains the least significant nibble of the least significant byte of the word to be written (bits 3-0). Once all four bytes have been received the word will be written.

:DOWNload [:MADDress]

DIAGnostic:DOWNload[:MADDress] <*address*>,<*data*> writes *data* into a Non-volatile user RAM segment starting at *address*. The user RAM segment is allocated by the DIAG:NRAM:CREate command.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	0 to 16,777,215 (#HFFFFFE)	none
<*data*>	arbitrary block program data	See "Parameter Types" on page 121	none

Comments

- **CAUTION:** Be certain that *all* of the data you download will be contained entirely within the allocated NRAM segment. Writing data outside of the NRAM segment will disrupt the operation of the command module. Most computers terminate an OUTPUT, PRINT, or WRITE statement with a carriage return or carriage return and line feed. These End-Of-Line characters must be either accounted for (NRAM segment sized to accommodate them), or suppressed using an appropriate IMAGE or FORMAT statement. Some helpful methods:
  - Size the NRAM segment a little larger than the expected data block.
  - Control the End-Of-Line characters with format statements.
  - Use the *Definite Length Arbitrary Block Program Data* format (see example on page 121) to send your data rather than the *Indefinite Length Arbitrary Block Program Data* format.
- This command is generally used to download data into User Configuration Tables. These tables allow the user to control the system's dynamic configuration, interrupt line allocations, commander/servant hierarchy, address space allocation, and mainframe extender configurations.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats. DOWNload is done by word (16-bit) access so *address* must be even.
- **Be certain that *address* specifies a location within the user RAM segment allocated using DIAG:NRAM:CREate if you are downloading a configuration table.** DIAG:DOWNload can change the contents of System RAM, causing unpredictable results.
- This command can also be used to write data to a device with registers in the A16 address space. See DIAGnostic:DOWNload:SADDress.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:ADDRess?, DIAG:UPLoad[:MADDress]?, VXI:CONF:CTABle, VXI:CONF:DCTable, VXI:CONF:ITABle, VXI:CONF:MTABle

Example — Load Dynamic Configuration Information into an Allocated RAM Segment

DIAG:NRAM:CRE 6	*Allocate a segment of user RAM.*
DIAG:BOOT:WARM	*Reboot system to complete allocation.*
DIAG:NRAM:ADDR?	*Query starting address.*
enter value to variable X	*Get starting address into X.*
DIAG:DOWN <*value of X*>,table data	*Download table data.*
VXI:CONF:DCTAB <*value of X*>	*Link configuration table to configuration algorithm.*
DIAG:BOOT:WARM	*Reboot to set new configuration.*

:DOWNload :SADDress

DIAGnostic:DOWNload:SADDress <*address*>,<*data*> writes *data* to Non-volatile user RAM at a single address specified by *address*, and writes *data* to devices with registers in A16 address space.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	0 to 16,777,215 (#HFFFFFE)	none
<*data*>	arbitrary block program data	See "Parameter Types" on page 121	none

Comments

- Most computers terminate an OUTPUT, PRINT, or WRITE statement with a carriage return or carriage return and line feed. These End-Of-Line characters must be accounted for or suppressed using an appropriate IMAGE or FORMAT statement. Some helpful methods:
  - Control the End-Of-Line characters with format statements.
  - Use the *Definite Length Arbitrary Block Program Data* format (see example on page 121) to send your data rather than the *Indefinite Length Arbitrary Block Program Data* format.
- A register address in A16 address space can be determined by:

  $$1FC000_{16} + (LADDR * 64) + register\_number$$

  where $1FC000_{16}$ is the base address in the command module A16 address space, LADDR is the device logical address, 64 is the number of address bytes per device, and register_number is the register to which the data is written.

  If the device is an A24 device, the address can be determined using the VXI:CONF:DLISt? command to find the base address in A24, and then adding the register_number to that value. A24 memory between address 200000_{16} and address $E00000_{16}$ is directly addressable by the command module.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats. DOWNload is done by word (16-bit) access so *address* must be even.
- **Related Commands:** DIAG:UPLoad:SADDress?

Example Download Data to a Single Address Location

This program downloads an array with the data 1, 2, 3, 4, 5 to register 32 on a device with logical address 40 in VXIbus A16 address space.

```
DIM Dnld_data(1:5)                          ! Dimension controller array.
DATA 1,2,3,4,5
READ Dnld_data(*)                           ! Load data into controller array.
OUTPUT "DIAG:DOWN:SADD #H1FCA20,#210";      ! This line is sent without termination.
Send Dnld_data as 16-bit words              ! Terminate after last word with EOI or LF and EOI.
```

:DRAM:AVAilable?

DIAGnostic:DRAM:AVAilable? returns the amount of RAM remaining (available) in the DRAM (Driver RAM) segment, which is the amount of RAM in the segment minus any previously loaded drivers.

Comments

- DIAG:DRAM:CREate does not allocate the RAM segment until after a subsequent re-boot.
- **Related Commands:** DIAG:DRAM:CREate, DIAG:DRIVer:LOAD, DIAG:DRIVer:LIST[:ALL]?

Example Determine Amount of Space Left for Drivers in the DRAM Segment

```
DIAG:DRAM:AVA?
enter statement                             ! Statement returns available DRAM in bytes.
```

:DRAM:CREate

DIAGnostic:DRAM:CREate ***<size>,<num_drivers>*** creates a Non-volatile RAM area for loading instrument drivers. **DIAG:DRAM:CREate 0** removes the RAM segment when the system is rebooted.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<size>	numeric	0 to available RAM or MIN | MAX	none
<num_drivers>	numeric	0 to available RAM or MIN | MAX | DEF	none

Comments

- *<size>* is the number of bytes to be allocated to DRAM use. A *size* of zero will remove the DRAM segment.
- *<num_drivers>* is the maximum number of drivers to be loaded.
- The DRAM segment will be created only after the System instrument has been rebooted (cycle power or execute DIAG:BOOT).
- Based on the *size* specified, DIAG:DRAM:CRE rounds the *size* up to an even value.
- DRAM will de-allocate previously allocated NRAM and RDISk segments.
- Using all of the available RAM (MAX) for the DRAM segment will limit some functions such as IBASIC program space, instrument reading storage space, and full functionality of the display terminal interface.
- Use DIAG:DRIVer:LOAD... and DIAG:DRIVer:LIST...? to load and manage DRAM.
- **Related Commands:** DIAG:DRAM:AVAilable?, DIAG:DRIVer:LOAD..., DIAG:DRIVer:LIST...?

Example

Allocate a 15 Kbyte Non-Volatile Driver RAM Segment

```
DIAG:DRAM:CRE 15360          Allocate 15 Kbyte segment of driver RAM.
```

:DRAM:CREate?

DIAGnostic:DRAM:CREate? [<MIN | MAX>,<MIN | MAX | DEF>] returns the size (in bytes) of a previously created Non-volatile RAM area for loading instrument drivers, and the number of drivers currently loaded.

Comments

- If you specify one of the parameters, you must specify both.

:DRIVer:INSTall

DIAGnostic:DRIVer:INSTall makes the drivers downloaded to Flash ROM available (installs them) by creating the driver index table.

Comments

- You cannot download any additional drivers into Flash ROM after you have executed this command. To download any new drivers you must recreate the Flash ROM driver area with the DIAG:FROM:CREate command. This will erase any drivers you have already downloaded, which will then have to be reloaded.
- **Related Commands:** DIAG:FROM:CREate

:DRIVer:LIST[:*type*]?

DIAGnostic:DRIVer:LIST[:*type*]? lists all drivers from the specified table found on the system. If no parameter is specified, all driver tables are searched and the data from each driver table is separated from the others by a semicolon.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
[:*type*]	discrete	ALL | RAM | ROM | FROM	ALL

For each driver listed, the system returns `NAME`, `IDN_MODEL`, `REV_CODE`, and `TABLE`.

Parameter	Description
NAME	The instrument name. This is the same label that appears on the instrument selection menu.
IDN_MODEL	The model name. This is the same model name as used in the response to the *IDN? command.
REV_CODE	The revision code. It is in the form A.nn.nn. A is an alpha character.
TABLE	The name of the table the driver was found in. This will be RAM, ROM, or FROM.

Comments

- **DIAGnostic:DRIVer:LIST?** lists all drivers found in the system.
- **DIAGnostic:DRIVer:LIST:FROM?** lists all drivers found in the Flash ROM driver table.
- **DIAGnostic:DRIVer:LIST:RAM?** lists all drivers found in the RAM driver table DRAM.
- **DIAGnostic:DRIVer:LIST:ROM?** lists all drivers found in the ROM driver table.
- **Related Commands:** DIAG:DRAM:AVAilable?, DIAG:DRAM:CREate, DIAG:DRIVer:LOAD...

Example

List All Drivers in the System

DIAG:DRIV:LIST? *Lists all drivers currently loaded.*

Example

List All Drivers in ROM

DIAG:DRIV:LIST:ROM? *Lists all of the drivers in ROM.*

:DRIVer:LOAD

DIAGnostic:DRIVer:LOAD ***<driver_block>*** loads the instrument driver contained in the *driver_block* into a previously created DRAM segment.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<driver_block>	arbitrary block program data	See "Parameter Types" on page 121	none

Comments

- ***driver_block*** is the actual binary driver data to be transferred.
- **Related Commands:** DIAG:DRAM:AVAilable?, DIAG:DRAM:CREate, DIAG:DRIVer:LIST...?

Example

Download a Driver Block

DIAG:DRIV:LOAD ***<driver_block>*** *Downloads the driver <driver_block> to DRAM memory or to Flash ROM.*

:DRIVer:LOAD :CHECked

DIAGnostic:DRIVer:LOAD:CHECked ***<driver_block>*** loads the instrument driver contained in the *driver_block* into a previously created DRAM segment. The *driver_block* is formatted in the same data byte format used by DIAG:DOWNload:CHECked.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<driver_block>	arbitrary block program data	See "Parameter Types" on page 121	none

Comments

- *<driver_block>* is the actual binary driver data to be transferred.
- This is the only way to download a device driver over a serial (RS-232) line.
- **Related Commands:** DIAG:DRAM:AVAilable?, DIAG:DRAM:CREate, DIAG:DRIVer:LIST...?

Example

Download a Driver Named Over RS-232

DIAG:DRIV:LOAD:CHEC ***<checked_driver_block>*** *Downloads the <checked_driver_block> to DRAM memory or Flash ROM.*

:FROM:AVAilable?

DIAGnostic:FROM:AVAilable? returns the amount of Flash ROM remaining to hold new device drivers. This is the amount of Flash ROM in the segment minus any previously loaded drivers and overhead.

Comments

- DIAG:FROM:AVAilable? returns zero if you have not created a valid flash driver area using DIAG:FROM:CREate while the system is in "LOAD" mode.
- **Related Commands:** DIAG:FROM:CREate (LOAD mode command only), DIAG:FROM:SIZE?, DIAG:DRIVer:LOAD, DIAG:DRIVer:LIST[:ALL]?

Example

Determine Amount of Space Left for Drivers in the Flash ROM Segment

```
DIAG:FROM:AVA?
enter statement
```

Statement returns available Flash ROM in bytes.

:FROM:CREate

DIAGnostic:FROM:CREate ***<num_drivers>*** creates a driver area in Flash ROM for loading instrument drivers. **DIAGnostic:FROM:CREate 0** removes the Flash ROM driver area, but does not affect the operating system program that is also in Flash ROM.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*num_drivers*>	numeric	0 to 64	none

Comments

- <*num_drivers*> is the maximum number of drivers to be loaded into Flash ROM.
- Use DIAG:DRIVer:LOAD... to load drivers into Flash ROM when the Flash ROMS Run/Load switch is in the "Load" position.
- **Related Commands:** DIAG:FROM:AVAilable?, DIAG:DRIVer:LOAD…, DIAG:DRIVer:LIST…?

Example

Initialize a Flash ROM Driver Segment for a Maximum of 8 Drivers

```
DIAG:FROM:CRE 8
```

:FROM:CREate?

DIAGnostic:FROM:CREate? returns the maximum number of drivers that a Flash ROM segment was created with.

Comments

- **Related Commands:** DIAG:FROM:CREate (LOAD mode command only).

Example

Determine Maximum Number of Drivers from a Flash ROM Segment

```
DIAG:FROM:CRE?
enter statement
```

Statement returns maximum number of Flash ROM drivers.

:FROM:SIZE?

DIAGnostic:FROM:SIZE? returns the amount of Flash ROM available to be used as Flash ROM driver area. This command does not take into account the size of the driver index table, checksum field, and so forth.

Comments

- **Related Commands:** DIAG:FROM:CREate (LOAD mode command only), DIAG:FROM:AVAilable?, DIAG:DRIVer:LOAD, DIAG:DRIVer:LIST[:ALL]?

Example

Determine Amount of Space Left for Drivers in the Flash ROM Segment

```
DIAG:FROM:SIZE?
enter statement                    Statement returns space available
                                   for FROM in bytes.
```

:INTerrupt:ACTivate

DIAGnostic:INTerrupt:ACTivate ***<mode>*** enables an interrupt on the VXI backplane interrupt line specified by DIAG:INTerrupt:SETup[*n*] to be acknowledged.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<mode>	boolean	0 | 1 | OFF | ON	none

Comments

- When an interrupt occurs and has been acknowledged, the response is read with the DIAG:INTerrupt:RESPonse? command.
- If an interrupt occurs on a VXIbus backplane interrupt line and the interrupt acknowledgment has not been enabled, there is no interrupt acknowledgment response. The interrupt will be held off until the interrupt acknowledge is enabled by either the DIAG:INT:ACTivate command or DIAG:INT:RESPonse? command.
- ON or 1 enables interrupt acknowledgment. OFF or 0 disables interrupt acknowledgment.
- In order for an interrupt to be serviced using the DIAG:INT commands, the interrupt line [*n*] must be assigned to an interrupt handler using the interrupt line allocation table covered in Chapter 2 (see page 54).
- Bit 8 in the Operation Status Register can be used to indicate when an interrupt has been acknowledged (see Chapter 4 for details).
- **Related Commands:** DIAG:INTerrupt:PRIority[*n*], DIAG:INTerrupt:RESPonse?, DIAG:INTerrupt:SETup[*n*]
- ***RST Condition:** DIAG:INTerrupt:ACTivate OFF (for all lines).
- Interrupt acknowledgment must be re-enabled each time an interrupt is acknowledged.

Example

Enable an Interrupt Acknowledgment on Line 2

```
DIAG:INT:SET2                      Set up interrupt line 2.
DIAG:INT:ACT ON                    Enable interrupt acknowledged.
```

:INTerrupt:PRIority[*n*]

DIAGnostic:INTerrupt:PRIority[*n*] <*level*> gives a priority level to the VXI interrupt line specified by [*n*].

Parameter Name	Parameter Type	Range of Values	Default Units
[*n*]	numeric	1 through 7	1
<*level*>	numeric	1-7 | MIN | MAX | DEF	none

Comments

- The priority of an interrupt line determines which line will be acknowledged first when more than one line is interrupting.
- For *level*, lower values have lower priority (level 1 is a lower priority than level 2).
- No parameter, or DEF (default) sets priority to 1.
- PRIority1 through PRIority7 specifies the VXI interrupt lines 1 through 7.
- Sending PRIority without an [*n*] value specifies VXI interrupt line 1.
- In order for an interrupt to be serviced using the DIAGnostic:INTerrupt commands, the interrupt line [*n*] must be assigned to an interrupt handler using the interrupt line allocation table (see page 54).
- This command has no effect if only one interrupt line is to be set up.
- **Related Commands:** DIAG:INTerrupt:ACTivate, DIAG:INTerrupt:SETup[*n*], DIAG:INTerrupt:RESPonse?

Example

Setup, Set a Priority, and Wait for VXI Interrupt Response on Line 2

```
DIAG:INT:SET2  ON          Handle interrupt on line 2.
DIAG:INT:PRI2  5           Set priority to 5 on line 2 code which will initiate an action resulting in an interrupt.
DIAG:INT:RESP?             Read the acknowledge response.
```

:INTerrupt:PRIority[*n*]?

DIAGnostic:INTerrupt:PRIority[*n*]? returns the current priority level set for the VXI interrupt line specified by [*n*].

Comments

- PRIority1? through PRIority7? specifies the VXI interrupt lines 1 through 7.
- Sending PRIority? without an [*n*] value specifies VXI interrupt line 1.
- **Related Commands:** DIAG:INTerrupt:PRIority[*n*], DIAG:INTerrupt:SETup[*n*], DIAG:INTerrupt:RESPonse?

Example

Determine Interrupt Priority for Line 4

```
DIAG:INT:PRI4?
enter statement            Statement returns 1 through 7.
```

:INTerrupt:RESPonse?

DIAGnostic:INTerrupt:RESPonse? returns the interrupt acknowledge response (STATUS/ID word) from the highest priority VXI interrupt line.

Comments

- The value returned is the response from the interrupt acknowledge cycle (STATUS/ID word) of a device interrupting on one of the interrupt lines set up with the DIAG:INT:SETup[*n*] command.
- Bits 0 through 7 of the STATUS/ID word are the interrupting device's logical address. Bits 8 through 15 are Cause/Status bits. Bits 16 through 31 (D32 Extension) are not read by the System instrument.
- If only bits 0 through 7 are used by the device (bits 8 - 15 are FF), the logical address can be determined by adding 256 to the value returned by DIAG:INT:RESPonse?. If bits 0 - 15 are used, the logical address is determined by adding 65,536 to the value returned (if the number returned is negative).
- Only the interrupt lines previously configured with the DIAG:INT:SETup[*n*] commands generate responses for this command.
- If there are interrupts on multiple lines when this command is received, or when the acknowledgment was enabled with DIAG:INT:ACTivate, the response data returned will be from the line with the highest priority set using the DIAG:INT:PRIority[*n*] command.
- If interrupt acknowledge has not been enabled with DIAG:INT:ACTivate, then it will be enabled by DIAG:INT:RESPonse?. System instrument execution is halted until the interrupt acknowledgment response is received.
- DIAG:INT:WAIT? can also be used to wait for the interrupt response.
- **Related Commands:** DIAG:INTerrupt:ACTivate, DIAG:INTerrupt:SETup[*n*], DIAG:INTerrupt:PRIority[*n*]

Example

Setup and Wait for VXI Interrupt Response on Line 2

```
DIAG:INT:PRI2  5                 Set priority to 5 on line 2.
DIAG:INT:SET2  ON                Handle interrupt on line 2.
.                                Code which will
.                                initiate an action
.                                resulting in an interrupt.
DIAG:INT:RESP?                   Read the acknowledge response.
```

:INTerrupt:SETup[*n*]

DIAGnostic:INTerrupt:SETup[*n*] <*mode*> specifies that an interrupt on VXI backplane interrupt line [*n*] will be serviced by the System instrument service routine (DIAGnostic:INTerrupt commands) rather than the operating system service routine.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
[*n*]	numeric	1 through 7	1
<*mode*>	boolean	0 | 1 | OFF | ON	none

Comments

- SETup1 through SETup7 specifies the VXI interrupt lines 1 through 7.
- Sending SETup without an [*n*] value specifies VXI interrupt line 1.
- ON or 1 specifies that interrupt handling is to be set up for the specified interrupt line. OFF or 0 indicates that interrupt handling of the specified line is to be done by the operating system.
- In order for an interrupt to be serviced using the DIAG:INT commands, the interrupt line [*n*] must be assigned to an interrupt handler using the interrupt line allocation table covered in Chapter 2 (see page 54).
- **Related Commands:** DIAG:INTerrupt:ACTivate, DIAG:INTerrupt:PRIority[*n*], DIAG:INTerrupt:RESPonse?
- ***RST Condition:** DIAG:INTerrupt:SETup OFF (for all lines).

Example

Setup and Wait for VXI Interrupt Response on Line 2

```
DIAG:INT:PRI2 5          Set priority to 5 on line 2.
DIAG:INT:SET2 ON         Handle interrupt on line 2 code which will initiate an action resulting in an interrupt.
DIAG:INT:RESP?           Read the acknowledge response.
```

:INTerrupt:SETup[*n*]?

DIAGnostic:INTerrupt:SETup[*n*]? returns the current state set by DIAG:INTerrupt:SETup[*n*] <*mode*>, for the VXI interrupt line specified by [*n*] in …SETup[*n*]?.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
[*n*]	numeric	1 through 7	1

Comments

- SETup1? through SETup7? specifies the VXI interrupt lines 1 through 7.
- Sending SETup? without an [*n*] value specifies VXI interrupt line 1.
- If 1 is returned, interrupt handling is set up for the specified interrupt line using the System instrument (DIAG:INT … commands). If 0 is returned, interrupt handling is done by the operating system.

- **Related Commands:** DIAG:INTerrupt:SETup[*n*], DIAG:INTerrupt:PRIority[*n*], DIAG:INTerrupt:ACTivate, DIAG:INTerrupt:RESPonse?

Example **Determine Interrupt Setup for Line 4**

```
DIAG:INT:SET4?
enter statement                    Statement returns 0 or 1.
```

:NRAM:ADDRess?

DIAGnostic:NRAM:ADDRess? returns the starting address of the Non-volatile user RAM segment allocated using DIAG:NRAM:CREate.

Comments

- DIAG:NRAM:CREate does not allocate the RAM segment until after a subsequent reboot. To get accurate results, execute DIAG:NRAM:ADDRess? after the reboot.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:CREate?, DIAG:DOWNload, DIAG:UPLoad?

Example **Determine Address of the Most Recently Created User RAM Segment**

```
DIAG:NRAM:ADDR?
enter statement                    Statement returns decimal numeric address.
```

:NRAM:CREate

DIAGnostic:NRAM:CREate ***<size>*** allocates a segment of Non-volatile user RAM for a user-defined table.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*size*>	numeric	0 to available RAM or MIN | MAX	none

Comments

- The RAM segment will be created only after the System instrument has been rebooted (cycle power or execute DIAG:BOOT).
- Based on the *size* specified, DIAG:NRAM:CREate rounds the *size* up to an even value.
- NRAM will de-allocate a previously allocated RDISk segment.
- Using all of the available RAM (MAX) for the NRAM segment will limit some functions such as IBASIC program space, instrument reading storage space, and full functionality of the display terminal interface.
- Use DIAG:NRAM:ADDRess? to determine the starting address of the RAM segment.
- Use DIAG:DOWNload, DIAG:UPLoad?, DIAG:PEEK, or DIAG:POKE to store and retrieve information in the Non-volatile RAM segment.
- Use DIAG:NRAM:CREate? MAX to find maximum available segment size.
- **Related Commands:** DIAG:NRAM:CREate?, DIAG:NRAM:ADDRess?, DIAG:DOWNload, DIAG:UPLoad?

Example Allocate a 15 Kbyte User Non-Volatile RAM Segment

```
DIAG:NRAM:CREate 15360          Allocate 15 Kbyte segment of user RAM.
```

:NRAM:CREate?

DIAGnostic:NRAM:CREate? [MIN | MAX] returns the current or allowable (MIN | MAX) size of the user Non-volatile RAM segment.

Comments

- DIAG:NRAM:CREate does not allocate driver RAM until a subsequent reboot. To get accurate results, execute DIAG:NRAM:CREate? after the reboot.
- **Related Commands:** DIAG:NRAM:ADDRess?, DIAG:NRAM:CREate

Example Check the Size of the User RAM Segment

```
DIAG:NRAM:CREate?
enter statement                 Statement enters size in bytes.
```

:PEEK?

DIAGnostic:PEEK? *<address>,<width>* reads the data (number of bits given by *width*) starting at *address*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	0 to 16,777,215 (#HFFFFFF)	none
<*width*>	numeric	8 | 16 | 32	none

Comments

- <*address*> specifies a location within the range of the control processor's addressing capability.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- **Related Commands:** DIAG:POKE

Example Read Byte from User Non-Volatile RAM

```
DIAG:PEEK? 16252928,8           Ask for byte.
enter statement                 Return value of byte.
```

:POKE

DIAGnostic:POKE ***<address>,<width>,<data>*** writes data (number of bits given by width) starting at address.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<address>	numeric	0 to 16,777,215 (#HFFFFFF)	none
<width>	numeric	8 | 16 | 32	none
<data>	numeric	8 to 32-bit integer	none

Comments

- *<address>* specifies a location within the range of the control processor's addressing capability.
- *<address>* and *<data>* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- **CAUTION:** DIAG:POKE can change the contents of any address in RAM. Changing the contents of RAM used by the command module's control processor can cause unpredictable results.
- **Related Commands:** DIAG:PEEK?

Example

Store Byte in User Non-Volatile RAM

```
DIAG:POKE 16252928,8,255
```

:RDISk:ADDress?

DIAGnostic:RDISk:ADDRess? returns the starting address of the RAM disc volume previously defined with the DIAG:RDISk:CREate command. The RAM disc volume is defined for use only by the IBASIC option.

Comments

- DIAG:RDISk:CREate does not allocate the RAM volume segment until after a subsequent reboot. To get accurate results, execute DIAG:RDISk:ADDRress? after the reboot.
- **Related Commands:** DIAG:RDISk:CREate, DIAG:RDISk:CREate?

Example

Return the Starting Address of the IBASIC RAM Volume

```
DIAG:RDIS:ADDR?
enter statement                    Statement returns decimal numeric
                                   address.
```

:RDISk:CREate

DIAGnostic:RDISk:CREate ***<size>*** allocates memory for a RAM disc volume. The RAM disc volume is defined for use only by the IBASIC option.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<size>	numeric	0 to available RAM or MIN | MAX	none

Comments

- The RAM disc segment will only be created after the System instrument has been rebooted (cycle power or execute DIAG:BOOT).
- Using all of the available RAM (MAX) for the disc volume segment will limit some functions such as IBASIC program space, instrument reading storage space, and full functionality of the display terminal interface.
- **Related Commands:** DIAG:RDISk:ADDRess?, DIAG:RDISk:CREate?

Example

Allocate a 64 Kbyte Segment for the IBASIC Option's RAM Volume

```
DIAG:RDIS:CRE 65536
```

:RDISk:CREate?

DIAGnostic:RDISk:CREate? [MIN | MAX] returns the current or allowable (MIN | MAX) size of the RAM disc volume segment.

Comments

- DIAG:RDISk:CREate does not allocate driver RAM until a subsequent reboot. To get accurate results, execute DIAG:RDISk:CREate? after the reboot.
- **Related Commands:** DIAG:RDISk:CREate, DIAG:RDISk:ADDRess?

Example

Return the Size of the Current RAM Disc Volume

```
DIAG:RDIS:CRE?
enter statement                    Returns numeric size.
```

:UPLoad[:MADDress]?

DIAGnostic:UPLoad[:MADDress]? <*address*>,<*byte_count*> returns the number of bytes specified by *byte_count*, starting at *address*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	0 to 16,777,215 (#HFFFFFE)	none
<*byte_count*>	numeric	0 to (999,999,998)	none

Comments

- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- UPLoad is done by word (16-bit) access so *address* and *byte_count* must be even.
- Data is returned in the Definite Block Response Data format:

  #<*non-zero digit*><*digit*(*s*)><*data byte*(*s*)>

  Where the value of <*non-zero digit*> equals the number of <*digit*(*s*)>. The value of <*digit*(*s*)> taken as a decimal integer indicates the number of <*data byte*(*s*)> to expect in the block.
- This command can also be used to retrieve data from a device with registers in A16 address space. See DIAGnostic:UPLoad:SADDress?
- **Related Commands:** DIAG:NRAM:ADDress?, DIAG:NRAM:CREate, DIAG:DOWNload

Example

Upload Data Stored on Non-Volatile User RAM

```
DIM HEADER$[6],DATA(1024)          ! 6 chars for "#41024" header;
                                   ! 1,024 chars for data bytes.
DIAG:NRAM:ADDR?                    ! Get starting address of NRAM.
enter ADD                          ! Address into ADD.
DIAG:UPL? <value of ADD>,1024      ! Request 1 Kbyte from address in
                                   ! ADD.
enter HEADER$                      ! Strip "#41024" from data.
enter DATA                         ! Get 1024 data bytes into the array;
                                   ! use enter format so statement will
                                   ! not terminate on CRs or LFs, and
                                   ! so forth. Line Feed (LF) and EOI
                                   ! follow the last character retrieved.
```

:UPLoad:SADDress?

DIAGnostic:UPLoad:SADDress? ***<address>,<byte_count>*** returns the number of bytes specified by *byte_count* at *address*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<address>	numeric	0 to 16,777,215 (#HFFFFFE)	none
<byte_count>	numeric	0 to (999,999,998)	none

Comments

- *<address>* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- UPLoad is done by word (16-bit) access so *address* and *byte_count* must be even.
- The register address in A16 address space can be determined by:

  $1FC000_{16}$ + (LADDR * 64) + register_number

  Where $1FC000_{16}$ is the base address in the VXIbus A16 address space, LADDR is the device logical address, 64 is the number of address bytes per device, and register_number is the register from which data is retrieved.

  If the device is an A24 device, the address can be determined using the VXI:CONF:DLISt? command to find the base address in A24, and then adding the register_number to that value. A24 memory between address 200000_{16} and address $E00000_{16}$ is directly addressable by the command module.
- Data is returned in the Definite Block Response Data format:

  #*<non-zero digit><digit(s)><data byte(s)>*

  where the value of *<non-zero digit>* equals the number of *<digit(s)>*. The value of *<digit(s)>* taken as a decimal integer indicates the number of *<data byte(s)>* to expect in the block.
- **Related Commands:** DIAG:DOWNload:SADDress

Example

Upload Data Stored in Non-Volatile User RAM

This program reads 1,024 data bytes from register 32 on a device with logical address 40 in command module A16 address space.

```
DIM HEADER$[6],DATA(1024)          ! 6 chars for "#41024" header;
                                   ! 1,024 chars for data bytes.
DIAG:UPL:SADD? #H1FCA20,1024       ! Request 1 Kbyte from device
                                   ! register 32.
enter HEADER$                      ! Strip "#41024" from data.
enter DATA                         ! Get 1,024 data bytes into the
                                   ! array; use enter format so
                                   ! statement will not terminate on
                                   ! CRs or LFs, and so forth. Line
                                   ! Feed (LF) and EOI follow the last
                                   ! character retrieved.
```

OUTPut

The OUTPut subsystem controls the output of pulses and levels to the ECLTrg and TTLTrg* trigger buses as well as the command module's front panel Trig Out connector. Signals connected to the front panel Trig In connector can also operate the ECLTrg and TTLTrg* trigger buses.

Note The HP E1406A Command Module's TTLTrg trigger lines and Trig Out port use "low true" or negative logic. When a trigger level is set (for example, OUTPut:EXTernal:LEVel 1), a low voltage is present.

Subsystem Syntax

```
OUTPut
    :ECLTrg<n> (:ECLTrg0 or :ECLTrg1)
        :IMMediate
        :LEVel
            [:IMMediate] <level>
            [:IMMediate]?
        :SOURce <source>
        :SOURce?
        [:STATe] <mode>
        [:STATe]?
    :EXTernal
        :IMMediate
        :LEVel
            [:IMMediate] <level>
            [:IMMediate]?
        :SOURce <source>
        :SOURce?
        [:STATe] <mode>
        [:STATe]?
    :TTLTrg<n> (:TTLTrg0 through :TTLTrg7)
        :IMMediate
        :LEVel
            [:IMMediate] <level>
            [:IMMediate]?
        :SOURce <source>
        :SOURce?
        [:STATe] <mode>
        [:STATe]?
```

:ECLTrg<*n*>:IMMediate

OUTPut:ECLTrg<*n*>:IMMediate causes a pulse to appear on the specified ECL Trigger line.

Comments

- ECLTrg<*n*> represents either ECLTrg0 or ECLTrg1.
- OUTPut:ECLTrg<*n*>:STATe must be ON and OUTPut:ECLTrg<*n*>:SOURce must be set to INT or NONE in order to issue an immediate pulse. A "settings conflict" error is generated if :STATe is not ON.
- **Related Commands:** OUTPut:ECLTrg<*n*>:SOURce, OUTPut:ECLTrg<*n*>[:STATe]

Example

Send Trigger Pulse to ECLTrg0

```
OUTP:ECLT0:STAT  ON          Set System instrument to send a pulse on ECLT0.
OUTP:ECLT0:SOUR  INT         Set trigger source to internal.
OUTP:ECLT0:IMM               Pulse the ECLTrg0 bus.
```

:ECLTrg<*n*>:LEVel [:IMMediate]

OUTPut:ECLTrg<*n*>:LEVel[:IMMediate] ***<level>*** sets the selected ECLTrg trigger line to logic level 0 or 1.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 or 1	N/A
<*level*>	boolean	0 | 1 | OFF | ON	none

Comments

- OUTP:ECLTrg<*n*>:STATe must be ON.
- OUTP:ECLTrg<*n*>:SOURce must be INTernal.
- OUTP:ECLTrg<*n*>:STATe must be ON for the source to drive the trigger line. Setting :STATe OFF does not change the source, so the signal driving the line is still present. Setting :STATe back ON sets the source to NONE and de-asserts the line.
- **Related Commands:** OUTP:ECLTrg<*n*>:LEVel[:IMMediate]?, OUTP:ECLTrg<*n*>:SOURce, OUTP:ECLTrg<*n*>[:STATe]
- ***RST Condition:** OUTP:ECLTrg<*n*>:LEVel 0

Example

ECLTrg0 Set to Logic Level 1

```
OUTP:ECLT0  ON               Enable ECLT0.
OUTP:ECLT0:SOUR  INT         Set the source to internal.
OUTP:ECLT0:LEV  1            Set trigger level.
```

:ECLTrg<*n*>:LEVel [:IMMediate]?

OUTPut:ECLTrg<*n*>:LEVel[:IMMediate]? returns the current logic level of the selected ECLTrg trigger line.

- ECLTrg<*n*> represents either ECLTrg0 or ECLTrg1.

Example

Determine Current State of ECLTrg1

```
OUTP:ECLT1:LEV?                 Ask for level.
enter statement                 Return state of trigger line.
```

:ECLTrg<*n*>:SOURce

OUTPut:ECLTrg<*n*>:SOURce <*source*> selects which source will drive the selected trigger line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 or 1	N/A
<*source*>	discrete	INT | EXT | NONE	none

Comments

- INT allows the selected trigger line to be driven by OUTP:ECLTrg<*n*>:LEVel commands.
- EXT allows the selected trigger line to be driven by the HP E1406A Command Module's Trig In front panel SMB connector.
- OUTP:ECLTrg<*n*>:STATe must be ON for the source to drive the trigger line. Setting :STATe OFF does not change the source, so the signal driving the line is still present. Setting :STATe back ON sets the source to NONE and de-asserts the line.
- **Related Commands:** OUTP:ECLTrg<*n*>[:STATe], OUTP:ECLTrg<*n*>:LEVel[:IMMediate]
- ***RST Condition:** OUTP:ECLTrg<*n*>:SOURce NONE

Example

Select the Trig In Connector to Drive ECLTrg0

```
OUTP:ECLT0:SOUR EXT
```

:ECLTrg<*n*> :SOURce?

OUTPut:ECLTrg<*n*>:SOURce? queries the source currently driving the selected trigger line.

Comments

- ECLTrg<*n*> represents either ECLTrg0 or ECLTrg1.
- Querying the source with :STATe OFF returns NONE, regardless of the actual source setting.

Example

Determine the Source Driving ECLTrg1

```
OUTP:ECLT1:SOUR?
enter statement                 Return trigger source.
```

:ECLTrg<*n*>[:STATe]

OUTPut:ECLTrg<*n*>[:STATe] <*mode*> enables configuration (for example, source and level) of the specified trigger line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 or 1	N/A
<*mode*>	boolean	0 | 1 | OFF | ON	none

Comments

- When a trigger line is asserted (OUTP:ECLTrg<*n*>:LEVel 1), it remains asserted when :STATe OFF is set. Setting :STATe ON again de-asserts the line by setting the source to NONE.
- **Related Commands:** OUTP:ECLTrg<*n*>:LEVel[:IMMediate], OUTP:ECLTrg<*n*>:SOURce
- ***RST Condition:** OUTP:ECLTrg<*n*>[:STATe] 0

Example

Enable the ECLTrg1 Trigger Bus

```
OUTP:ECLT1:STAT ON
```

:ECLTrg<*n*>[:STATe]?

OUTPut:ECLTrg<*n*>[:STATe]? returns the current state (`ON` or `OFF`) of the selected trigger line.

Comments

- ECLTrg<*n*> represents either ECLTrg0 or ECLTrg1.

Example

Query the State of ECLTrg1

```
OUTP:ECLT1:STAT?
enter statement                    Return the current state.
```

:EXTernal:IMMediate

OUTPut:EXTernal:IMMediate causes a pulse to appear on the HP E1406A Command Module's front panel Trig Out SMB port.

Comments

- OUTP:EXTernal:STATe must be ON and OUTP:EXTernal:SOURce must be INT or NONE.
- **Related Commands:** OUTP:EXTernal[:STATe], OUTP:EXTernal:SOURce

Example

Send Trigger Pulse to Trig Out Port

```
OUTP:EXT:STAT ON                   Enable Trig Out port.
OUTP:EXT:SOUR INT                  Set trigger source.
OUTP:EXT:IMM                       Pulse Trig Out.
```

:EXTernal:LEVel [:IMMediate]

OUTPut:EXTernal:LEVel[:IMMediate] ***<level>*** sets the Trig Out port to a logic level of 0 or 1.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<level>	boolean	0 | 1 | OFF | ON	none

Comments

- OUTP:EXTernal:STATe must be ON.
- OUTP:EXTernal:SOURce must be INTernal.
- Once the level of the Trig Out port is set to logic level 1, it remains set if OUTP:EXTernal:STATe OFF is set. Setting OUTP:EXTernal:STATe back to ON sets the output back to logic level 0, and sets OUTP:EXTernal:SOURce to NONE.
- **Related Commands:** OUTP:EXTernal:LEVel[:IMMediate]?, OUTP:EXTernal:SOURce, OUTP:EXTernal[:STATe]
- ***RST Condition:** OUTP:EXTernal:LEVel 0

Example

Set Trig Out Port to Logic Level 1

```
OUTP:EXT:STAT  ON          Enable output.
OUTP:EXT:SOUR  INT         Set trigger source internal.
OUTP:EXT:LEV  1            Set output level.
```

:EXTernal:LEVel [:IMMediate]?

OUTPut:EXTernal:LEVel[:IMMediate]? returns the current logic level of the Trig Out port.

Example

Determine the Current State of Trig Out Port

```
OUTP:EXT:LEV?              Ask for level.
enter statement            Return state of trigger bus.
```

:EXTernal:SOURce

OUTPut:EXTernal:SOURce ***<source>*** selects which source will drive the Trig Out port.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<source>	discrete	INT | TTLTrg | ECLTrg | NONE	none

Comments

- INT allows the Trig Out port to be driven by OUTP:EXTernal:LEVel.
- TTLTrg or ECLTrg allows the Trig Out port to be driven by the selected VXIbus trigger line.
- OUTP:EXTernal:STATe must be ON for the source to operate the Trig Out port. Setting :STATe OFF does not change the source, so the signal driving the port is still present. Setting :STATe back ON sets the source to NONE.

- **Related Commands:** OUTP:EXTernal[:STATe], OUTP:EXTernal:LEVel[:IMMediate]
- ***RST Condition:** OUTP:EXTernal:SOURce NONE

Example **Select TTLTrg0* to Drive the Trig Out Port**

```
OUTP:EXT:SOUR TTLT0
```

:EXTernal:SOURce?

OUTPut:EXTernal:SOURce? queries for the source currently driving the Trig Out port.

Comments

- Querying the source with :STATe OFF returns NONE, regardless of the actual source setting.

Example **Determine the Source Driving Trig Out**

```
OUTP:EXT:SOUR?
enter statement                    Return Trig Out source.
```

:EXTernal[:STATe]

OUTPut:EXTernal[:STATe] ***<mode>*** enables configuration (for example, source and level) of the command module's Trig Out port.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<mode>	boolean	0 | 1 | OFF | ON	none

Comments

- When the Trig Out port is set to logic level 1, it remains set if OUTP:EXTernal:STATe is set to OFF. Setting OUTP:EXTernal:STATe back to ON sets the Trig Out port back to logic level 0. OUTP:EXTernal:SOURce is set to NONE.
- **Related Commands:** OUTP:EXTernal:SOURce, OUTP:EXTernal:LEVel[:IMMediate]
- ***RST Condition:** OUTP:EXTernal[:STATe] 0

Example **Enable the Trig Out Port**

```
OUTP:EXT:STAT ON
```

:EXTernal[:STATe]?

OUTPut:EXTernal[:STATe]? returns the current state (ON or OFF) of the Trig Out port.

Example **Query the State of Trig Out Port**

```
OUTP:EXT:STAT?
enter statement                    Return the current state.
```

:TTLTrg<*n*>:IMMediate

OUTPut:TTLTrg<*n*>:IMMediate causes a pulse to appear on the specified TTL trigger line.

Comments

- TTLTrg<*n*> represents TTLTrg0 through TTLTrg7.
- OUTP:TTLTrg<*n*>:STATe must be ON and OUTP:TTLTrg<*n*>:SOURce must be set to INT or NONE in order to issue an immediate pulse. An error message is generated if :STATe is not ON.
- **Related Commands:** OUTP:TTLTrg<*n*>:SOURce, OUTP:TTLTrg<*n*>[:STATe]

Example

Send Trigger Pulse to TTLTrg0* and TTLTrg4*

```
OUTP:TTLT0:STAT  ON          Enable the System instrument.
OUTP:TTLT4:STAT  ON          Send a pulse on TTLT0 and TTLT4.
OUTP:TTLT0:SOUR  INT
OUTP:TTLT4:SOUR  INT         Set trigger sources.
OUTP:TTLT0:IMM               Pulse the TTLTrg0 bus.
OUTP:TTLT4:IMM               Pulse the TTLTrg4 bus.
```

:TTLTrg<*n*>:LEVel[:IMMediate]

OUTPut:TTLTrg<*n*>:LEVel[:IMMediate] ***<level>*** sets the selected TTLTrg* trigger line to logic level 0 or 1.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 through 7	N/A
<*level*>	boolean	0 | 1 | OFF | ON	none

Comments

- OUTP:TTLTrg<*n*>:STATe must be ON for the source to drive the trigger line. Setting :STATe OFF does not change the source, so the signal driving the line is still present. Setting :STATe back ON sets the source to NONE and de-asserts the line.
- OUTPut:TTLTrg<*n*>:SOURce must be INTernal.
- **Related Commands:** OUTP:TTLTrg<*n*>:LEVel[:IMMediate]?, OUTP:TTLTrg<*n*>:SOURce, OUTP:TTLTrg<*n*>[:STATe]
- ***RST Condition:** OUTP:TTLTrg<*n*>:LEVel 0

Example

TTLTrg0* Set to Logic Level 1

```
OUTP:TTLT0:STAT  ON          Enable TTLT0.
OUTP:TTLT0:SOUR  INT         Set source to internal.
OUTP:TTLT0:LEV  1            Set trigger level.
```

:TTLTrg<*n*>:LEVel [:IMMediate]?

OUTPut:TTLTrg<*n*>:LEVel[:IMMediate]? returns the current logic level of the selected TTLTrg* trigger line specified by *n* 0 through 7.

Comments

- TTLTrg<*n*> represents TTLTrg0 through TTLTrg7.

Example

Determine Current State of TTLTrg1*

```
OUTP:TTLT1:LEV?                 Ask for level.
enter statement                 Return state of trigger line.
```

:TTLTrg<*n*>:SOURce

OUTPut:TTLTrg<*n*>:SOURce <*source*> selects which source will drive the selected trigger line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 through 7	N/A
<*source*>	discrete	INT | EXT | NONE	none

Comments

- INT allows the selected trigger line to be driven by OUTP:TTLTrg*n*:LEVel commands.
- EXT allows the selected trigger line to be driven by the Trig In front panel SMB connector.
- OUTP:TTLTrg<*n*>:STATe must be ON for the source to drive the trigger line. Setting :STATe OFF does not change the source, so the signal driving the line is still present. Setting :STATe back ON sets the source to NONE and de-asserts the line.
- **Related Commands:** OUTP:TTLTrg<*n*>[:STATe], OUTP:TTLTrg<*n*>:LEVel[:IMMediate]
- ***RST Condition:** OUTP:TTLTrg<*n*>:SOURce NONE

Example

Select the Trig In Connector to Drive TTLTrg0*

```
OUTP:TTLT0:SOUR EXT
```

:TTLTrg<*n*>:SOURce?

OUTPut:TTLTrg<*n*>:SOURce? queries the source currently driving the selected trigger line.

Comments

- TTLTrg<*n*> represents TTLTrg0 through TTLTrg7.
- Querying the source with :STATe OFF returns NONE, regardless of the actual source setting.

Example

Determine the Source Driving TTLTrg1*

```
OUTP:TTLT1:SOUR?
enter statement                 Return trigger source.
```

:TTLTrg<*n*>[:STATe]

OUTPut:TTLTrg<*n*>[:STATe] <*mode*> controls whether the System instrument may drive the specified trigger line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 through 7	N/A
<*mode*>	boolean	0 | 1 | OFF | ON	none

Comments

- OUTP:TTLTrg<*n*>:STATe must be ON in order to specify a trigger source, issue a pulse, or set a trigger level.
- OUTP:TTLTrg<*n*>:STATe must be ON for the source to drive the trigger line. Setting :STATe OFF does not change the source, so the signal driving the line is still present. Setting :STATe back ON sets the source to NONE and de-asserts the line.
- **Related Commands:** OUTP:TTLTrg<*n*>:SOURce, OUTP:TTLTrg<*n*>:LEVel[:IMMediate]
- ***RST Condition:** OUTP:TTLTrg<*n*>:STATe 0

Example

Enable the TTLTrg1* Trigger Line

```
OUTP:TTLT1:STAT ON
```

:TTLTrg<*n*>[:STATe]?

OUTPut:TTLTrg<*n*>[:STATe]? returns the current state (`ON` or `OFF`) of the selected trigger line.

Comments

- TTLTrg<*n*> represents TTLTrg0 through TTLTrg7.

Example

Query the State of TTLTrg1*

```
OUTP:TTLT1:STAT?
enter statement                    Return the current state.
```

PROGram

The PROGram subsystem allows you to write an operating system into the command module Flash ROM, to read data from the Flash ROM, or to delete the contents of the Flash ROM. PROG:DEFine? and PROG:DEFine:CHECked? are valid in SYSTEM *or* LOAD mode. The other PROGram commands listed are active ONLY in LOAD mode.

Subsystem Syntax

```
PROGram
    [:SELected]
        :DEFine
            :CHECked <op_sys>
            :CHECked?
        :DEFine?
        :DELete
```

[:SELected]:DEFine

PROGram[:SELected]:DEFine *<op_sys>* writes the operating system into Flash ROM.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*op_sys*>	arbitrary block program data	See comments below.	none

Comments

- This command returns an error if executed from the System instrument (switch set to the "Run" position).
- **Arbitrary Block Program Data parameters** are used to transfer blocks of data in the form of bytes. The block of data bytes is preceded by a preamble which indicates either 1) the number of data bytes which follow, or 2) that the following data block will be terminated upon receipt of a New Line message with the EOI signal true. The syntax is:

  Definite Length Block

  #<*non-zero digit*><*digit*(*s*)><*data byte*(*s*)>

  Where the value of <*non-zero digit*> equals the number of <*digit*(*s*)>. The value of <*digit*(*s*)> taken as a decimal integer indicates the number of <*data byte*(*s*)> in the block.

  Indefinite Length Block

  #0<*data byte*(*s*)><NL^END>

  Examples of sending 4 data bytes:

  #14<*byte*><*byte*><*byte*><*byte*>
  #3004<*byte*><*byte*><*byte*><*byte*>
  #0<*byte*><*byte*><*byte*><*byte*><NL^END>

- **Related Commands:** PROG[:SELected]:DELete

[:SELected]:DEFine :CHECked

PROGram[:SELected]:DEFine:CHECked ***<op_sys>*** writes the operating system into Flash ROM over an RS-232 line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<op_sys>	arbitrary block program data	See comments below.	none

Comments

- This command returns an error if executed from the System instrument.
- **Arbitrary Block Program Data parameters** are used to transfer blocks of data in the form of bytes. The block of data bytes is preceded by a preamble which indicates either 1) the number of data bytes which follow, or 2) that the following data block will be terminated upon receipt of a New Line message with the EOI signal true. The syntax is:

  #### Definite Length Block

  #<*non-zero digit*><*digit*(*s*)><*data byte*(*s*)>

  Where the value of <*non-zero digit*> equals the number of <*digit*(*s*)>. The value of <*digit*(*s*)> taken as a decimal integer indicates the number of <*data byte*(*s*)> in the block.

  #### Indefinite Length Block

  #0<*data byte*(*s*)><NL^END>

  Examples of sending 4 data bytes:

  #14<*byte*><*byte*><*byte*><*byte*>
  #3004<*byte*><*byte*><*byte*><*byte*>
  #0<*byte*><*byte*><*byte*><*byte*><NL^END>

- **Related Commands:** PROG[:SELected]:DELete

Byte Format Each byte sent with this command is expected to be in the following format:

Bit #	7	6	5	4	3	2	1	0
	Control Bit	*Check Bits*			*Data Bits*			

- ***Control Bit*** is used to indicate the serial driver information such as clear, reset, or end of transmission. This bit is ignored by the regular 488.2 driver. The control bit should be one for regular data.
- ***Check Bits*** are used to detect and correct a single bit error. The control bit is not included in the check. The check bits are a Hamming single bit error correction code, as specified by the following table: over an RS-232 line.

Data Value	Check Bits
0	0
1	7
2	6
3	1
4	5
5	2
6	3
7	4
8	3
9	4
10	5
11	2
12	6
13	1
14	0
15	7

- ***Data Bits*** are the actual data being transferred (four bits at a time). Each word to be written requires four data bytes for transmission. The significance of the data is dependent on the order received. The first data byte received contains the most significant nibble of the 16-bit word to be written (bits 15-12).

  The next data byte received contains the least significant nibble of the most significant byte of the word (bits 11-8). The third data byte received contains the most significant nibble of the least significant byte of the word (bits 7-4). The fourth data byte received contains the least significant nibble of the least significant byte of the word to be written (bits 3-0). Once all four bytes have been received the word will be written.

[:SELected]:DEFine :CHECked?

PROGram[:SELected]:DEFine:CHECked? reads *data* from Flash ROM over an RS-232 line.

Comments

- This command returns a definite length arbitrary block of *data* in the same format used to send data over RS-232.

[:SELected]:DEFine?

PROGram[:SELected]:DEFine? reads *data* from the Flash ROM.

Comments

- This command returns the operating system program loaded in Flash ROM as a definite length arbitrary block.

[:SELected]:DELete

PROGram[:SELected]:DELete erases the entire contents of the Flash ROM.

Comments

- This command returns an error if executed from the System instrument.

CAUTION **This command will remove the HP E1406A operating system and should NEVER be used unless you are updating the operating system. Do not use this command when addressing instruments other than the command module, as the results may be undetermined and may cause the instrument to fail.**

STATus

The STATus subsystem commands access the Condition, Event, and Enable Registers in the Operation Status Group and the Questionable Data Group.

Subsystem Syntax

```
STATus
    :OPERation
        :CONDition?
        :ENABle <event>
        :ENABle?
        [:EVENt]?
        :NTRansition <unmask>
        :PTRansition <unmask>
    :PRESet
    :QUEStionable
        :CONDition?
        :ENABle <event>
        :ENABle?
        [:EVENt]?
        :NTRansition <unmask>
        :PTRansition <unmask>
```

:OPERation :CONDition?

STATus:OPERation:CONDition? returns the state of the Condition Register in the Operation Status Group. The state represents conditions which are part of an instrument's operation.

Comments

- Bit 8 in the register is used by the System instrument (command module) to indicate when an interrupt set up by the DIAG:INTerrupt commands has been acknowledged.
- Reading the Condition Register does not change the setting of bit 8. Bit 8 is cleared by the DIAG:INTerrupt:RESPonse? command.
- **Related Commands:** STAT:OPER:ENABle, STAT:OPER[:EVENt]?

Example

Read the Contents of the Condition Register

```
STAT:OPER:COND?                    Query register.
enter statement
```

:OPERation:ENABle

STATus:OPERation:ENABle ***<event>*** sets an enable mask to allow events monitored by the Condition Register and recorded in the Event Register, to send a Summary bit to the Status Byte Register (bit 7).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<event>	numeric	256	none

Comments

- Bit 8 in the Condition Register is used by the System instrument (command module) to indicate when an interrupt set up by the DIAG:INTerrupt commands has been acknowledged.
- Bit 8 is the only bit used in the Condition Register (by the System instrument), therefore, it is the only bit which needs to be unmasked in the Event Register. Specifying the "bit weight" for the *event* unmasks the bit. The bit weight is 256 and can be specified in decimal, hexadecimal (#H), Octal (#Q) or binary (#B).
- When the Summary bit is sent, it sets bit 7 in the Status Byte Register.
- **Related Commands:** STAT:OPER:ENABle?

Example

Unmask Bit 8 in the Event Register

```
STAT:OPER:ENAB  256          Unmask bit 8.
```

:OPERation:ENABle?

STATus:OPERation:ENABle? returns which bits in the Event Register (Operation Status Group) are unmasked.

Comments

- Bit 8 in the Condition Register is used by the System instrument (command module) to indicate when an interrupt set up by the DIAG:INTerrupt commands has been acknowledged.
- Bit 8 in the Event Register generally is the only bit which will be unmasked. If this bit is unmasked when STAT:OPER:ENABle? is sent, 256 is returned.
- Reading the Event Register mask does not change the mask setting (STAT:OPER:ENABle *<event>*).
- **Related Commands:** STAT:OPER:ENABle

Example

Read the Event Register Mask

```
STAT:OPER:ENAB?              Query register mask.
enter statement
```

:OPERation[:EVENt]?

STATus:OPERation[:EVENt]? returns which bits in the Event Register (Operation Status Group) are set. The Event Register indicates when there has been a positive transition in the Condition Register.

Comments

- Bit 8 in the Condition Register is used by the System instrument (command module) to indicate when an interrupt set up by the DIAG:INTerrupt commands has been acknowledged.
- Bit 8 in the Event Register generally is the only bit which is used. If this bit is set when STAT:OPER:EVENt? is sent, 256 is returned.
- Reading the Event Register clears the contents of the register. If the Event Register is to be used to generate a service request (SRQ), you should clear the register before enabling the SRQ (*SRE). This prevents an SRQ from occurring due to a previous event.
- **Related Commands:** STAT:OPER:ENABle, STAT:OPER:ENABle?

Example

Read the Event Register

```
STAT:OPER:EVEN?                    Query if bit(s) is set.
enter statement
```

:OPERation :NTRansition

STATus:OPERation:NTRansition ***<unmask>*** sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the associated Condition Register will set the same bit in the associated Event Register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<unmask>	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- Executable when initiated.
- No coupled commands.
- ***RST Condition**: No change.
- **Related Commands:** STATus subsystem commands, *SRE, *STB?

Example

Set the Operation Register Negative Transition Mask

```
STAT:OPER:NTR  64                  Set event bit when wait-for-arm
                                   state is entered.
```

:OPERation :PTRansition

STATus:OPERation:PTRansition ***<unmask>*** sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the associated Condition Register will set the same bit in the associated Event Register.

Comments

- See STATus:OPERation:NTRansition <*unmask*> for parameters and comments.

Example **Set the Operation Register Positive Transition Mask**

```
STAT:OPER:PTR  64          Set event bit when wait-for-arm state is entered.
```

:PRESet

STATus:PRESet sets each bit in the Enable Register (Standard Operation Status Group) to '0'.

Example **Preset the Enable Register**

```
STAT:PRES                  Preset Enable Register.
```

:QUEStionable :CONDition?

STATus:QUEStionable:CONDition? returns the state of the Condition Register in the Questionable Status Group. The state represents conditions which are part of an instrument's operation.

Comments

- **Related Commands:** STAT:QUES:ENABle, STAT:QUES[:EVENt]?

Example **Read the Contents of the Condition Register**

```
STAT:QUES:COND?            Query register.
```

Note **STATus:QUEStionable** commands are supported by the System instrument, however, they are not used by the System instrument. Queries of the Questionable Data Condition and Event Registers will always return +0.

:QUEStionable :ENABle

STATus:QUEStionable:ENABle ***<event>*** sets an enable mask to allow events monitored by the Condition Register and recorded in the Event Register, to send a summary bit to the Status Byte Register (bit 7).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<event>	numeric	256	none

Comments

- When the summary bit is sent, it sets bit 7 in the Status Byte Register.
- **Related Commands:** STAT:QUES:ENABle?

Example

Unmask Bit 8 in the Event Register

```
STAT:QUES:ENAB  256          Unmask bit 8.
```

:QUEStionable :ENABle?

STATus:QUEStionable:ENABle? returns which bits in the Event Register (Questionable Status Group) are unmasked.

Comments

- Reading the Event Register mask does not change the mask setting (STAT:QUES:ENABle <*event*>).
- **Related Commands:** STAT:QUES:ENABle

Example

Read the Event Register Mask

```
STAT:QUES:ENAB?          Query register mask.
```

:QUEStionable [:EVENt]?

STATus:QUEStionable[:EVENt]? returns which bits in the Event Register (Questionable Status Group) are set. The Event Register indicates when there has been a positive transition in the Condition Register.

Comments

- Reading the Event Register clears the contents of the register. If the Event Register is to be used to generate a service request (SRQ), you should clear the register before enabling the SRQ (*SRE). This prevents an SRQ from occurring due to a previous event.
- **Related Commands:** STAT:QUES:ENABle, STAT:QUES:ENABle?

Example

Read the Event Register

```
STAT:QUES:EVEN?          Query returns bit(s) set.
```

:QUEStionable :NTRansition

STATus:QUEStionable:NTRansition ***<unmask>*** sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the associated Condition Register will set the same bit in the associated Event Register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<unmask>	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- Executable when initiated.
- No coupled commands.
- ***RST Condition**: No change.
- **Related Commands:** STATus subsystem commands, *SRE, *STB?

Example

Set the Questionable Signal Register Negative Transition Mask

```
STAT:QUES:NTR  64          Set event bit when wait-for-arm state is entered.
```

:QUEStionable :PTRansition

STATus:QUEStionable:PTRansition ***<unmask>*** sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the associated Condition Register will set the same bit in the associated Event Register.

Comments

- See STATus:QUEStionable:NTRansition *<unmask>* for parameters and comments.

Example

Set the Questionable Signal Register Positive Transition Mask

```
STAT:QUES:PTR  64          Set event bit when wait-for-arm state is entered.
```

SYSTem

The SYSTem command subsystem for the System instrument provides for:

- Control and access of the System instrument's real time clock/calendar (SYST:TIME, SYST:TIME?, SYST:DATE, SYST:DATE?).
- Access to the System instrument's error queue (SYST:ERRor?).
- Configuring the communication ports (HP-IB and serial).

Subsystem Syntax

```
SYSTem
    :COMMunicate
        :GPIB
            :ADDRess?
        :SERial[n]
            :CONTrol
                :DTR <dtr_cntrl>| ON | OFF | STANdard | IBFull
                :DTR?
                :RTS <rts_cntrl>| ON | OFF | STANdard | IBFull
                :RTS?
            [:RECeive]
                :BAUD <baud_rate>| MIN | MAX
                :BAUD? [MIN | MAX]
                :BITS <bits>| 7 | 8 | MIN | MAX
                :BITS? [MIN | MAX]
                :PACE
                    [:PROTocol] <protocol> XON | NONE
                    [:PROTocol]?
                    :THReshold
                        :STARt <char_count>
                        :STARt? [MIN | MAX]
                        :STOP <char_count>
                        :STOP? [MIN | MAX]
                :PARity
                    <type>| EVEN | ODD | ZERO | ONE | NONE
                    <type>?
                    :CHECk <check_cntrl>| 1 | 0 | ON | OFF
                    :CHECk?
                :SBITs <sbits>| 1 | 2 | MIN | MAX
                :SBITs? [MIN | MAX]
            :TRANsmit
                :AUTO <auto_cntrl>| 1 | 0 | ON | OFF
                :AUTO?
                :PACE
                    [:PROTocol] <protocol> XON | NONE
                    [:PROTocol]?
    :DATE <year>,<month>,<day>
    :DATE? [MIN | MAX,MIN | MAX,MIN | MAX]
    :ERRor?
    :TIME <hour>,<minute>,<second>
    :TIME? [MIN | MAX,MIN | MAX,MIN | MAX]
    :VERSion?
```

:COMMunicate:GPIB :ADDRess?

SYSTem:COMMunicate:GPIB:ADDRess? returns the HP E1406A Command Module's primary HP-IB address.

Comments

- The HP E1406A Command Module (primary) HP-IB address is set using switches on the module.

Example

Read the Primary HP-IB Address

```
SYST:COMM:GPIB:ADDR?          Read the HP-IB address.
enter statement               Enter the HP-IB address.
```

:COMMunicate :SERial[*n*]:...

The **SYSTem:COMMunicate:SERial[*n*]:...** commands set and/or modify the configuration of the serial interface(s) that are under control of the System instrument (command module). The interface to be affected by the command is specified by a number (zero through seven) which replaces the [*n*] in the SERial[*n*] command. The number is the interface's **card number**. Card number zero specifies the command module's built-in interface while one through seven specify one of up to seven HP E1324 B-size plug-in serial interface modules. The serial interface installed at (System instrument's logical address) +1 becomes card number 1, the serial interface installed at the next sequential logical address becomes card number 2, and so on. The logical addresses used by plug-in serial interfaces must start at (System instrument's logical address) +1 and be contiguous (no unused logical addresses). The factory set logical address of the HP E1406A Command Module is 0.

Comments

- Serial communication commands take effect *after* the end of the program message containing the command.
- Serial communication settings for the built-in RS-232 interface can be stored in its non-volatile RAM ***only*** after the DIAG:COMM:SERial[*n*]:STORe command is executed. These settings are used at power-up and DIAG:BOOT[:WARM].
- Serial communication settings for the HP E1324A RS-232/422 Terminal Interface can be stored in its on-board non-volatile EEROM ***only*** after the DIAG:COMM:SER[*n*]:STOR command is executed. These settings are used at power-up and DIAG:BOOT[:WARM].
- DIAG:BOOT:COLD will set the serial communication parameters to the following defaults:
  - BAUD 9600
  - BITS 8
  - PARity NONE
  - SBITs 1
  - DTR ON
  - RTS ON
  - PACE XON

Example

Set Baud Rate for Plug-in Card 2

```
SYST:COMM:SER2:BAUD  9600     (must be a card number 1 also)
```

:COMMunicate :SERial[*n*]:CONTrol :DTR

SYSTem:COMMunicate:SERial[*n*]:CONTrol:DTR ***<dtr_cntrl>*** controls the behavior of the Data Terminal Ready output line. DTR can be set to a static state (ON | OFF), can operate as a modem control line (STANdard), or can be used as a hardware handshake line (IBFull).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*dtr_cntrl*>	discrete	ON | OFF | STAN | IBF	none

Comments

- The following table defines each value of *dtr_cntrl*:

Value	Definition
ON	DTR Line is asserted.
OFF	DTR Line is unasserted.
STANdard	DTR will be asserted when the serial interface is ready to send *output* data. Data will be sent if the connected device asserts DSR and CTS.
IBFull	While the input buffer is not yet at the :STOP threshold, DTR is asserted. When the input buffer reaches the :STOP threshold, DTR will be unasserted.

- DIAG:BOOT:COLD will set DTR to ON.
- **Related Commands:** SYST:COMM:SER[*n*]:CONT:RTS, SYST:COMM:SER[*n*][:REC]:PACE:THR:STARt, SYST:COMM:SER[*n*][:REC]:PACE:THR:STOP
- ***RST Condition:** No change.

Example

Assert the DTR Line

```
SYST:COMM:SER0:CONT:DTR  ON
```

:COMMunicate :SERial[*n*]:CONTrol :DTR?

SYSTem:COMMunicate:SERial[*n*]:CONTrol:DTR? returns the current setting for DTR line control.

Example

Check the Setting of DTR Control

```
SYST:COMM:SER0:CONT:DTR?
enter statement                    Statement enters the string "ON",
                                   "OFF", "STAN", or "IBF".
```

:COMMunicate :SERial[*n*]:CONTrol :RTS

SYSTem:COMMunicate:SERial[*n*]:CONTrol:RTS ***<rts_cntrl>*** controls the behavior of the Request To Send output line. RTS can be set to a static state (ON | OFF), can operate as a modem control line (STANdard), or can be used as a hardware handshake line (IBFull).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<rts_cntrl>	discrete	ON | OFF | STAN | IBF	none

Comments

- The following table defines each value of *rts_cntrl*:

Value	Definition
ON	RTS Line is asserted.
OFF	RTS Line is unasserted.
STANdard	RTS will be asserted when the serial interface is ready to send *output* data. Data will be sent if the connected device asserts CTS and DSR.
IBFull	While the input buffer is not yet at the :STOP threshold, RTS is asserted. When the input buffer reaches the :STOP threshold, RTS will be unasserted.

- DIAG:BOOT:COLD will set RTS to ON.
- **Related Commands:** SYST:COMM:SER[*n*]:CONT:DTR, SYST:COMM:SER[*n*][:REC]:PACE:THR:STARt, SYST:COMM:SER[*n*][:REC]:PACE:THR:STOP
- ***RST Condition:** No change.

Example

Unassert the RTS Line

```
SYST:COMM:SER0:CONT:RTS  OFF
```

:COMMunicate :SERial[*n*]:CONTrol :RTS?

SYSTem:COMMunicate:SERial[*n*]:CONTrol:RTS? returns the current setting for RTS line control.

Example

Check the Setting of RTS Control

```
SYST:COMM:SER0:CONT:RTS?
enter statement                    Statement enters the string "ON",
                                   "OFF", "STAN", or "IBF".
```

:COMMunicate :SERial[*n*][:RECeive] :BAUD

SYSTem:COMMunicate:SERial[*n*][:RECeive]:BAUD <*baud_rate*> sets the baud rate for the serial port.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*baud_rate*>	numeric	300 | 1200 | 2400 | 4800 | 9600 | 19200 | MIN | MAX	none

Comments

- Attempting to set *baud_rate* to other than those values shown will result in an `Error -222, "Data out of range"`.
- DIAG:BOOT:COLD will set BAUD to 9600.
- ***RST condition:** No change.

Example

Set the Baud Rate to 1200

```
SYST:COMM:SER0:BAUD 1200
```

:COMMunicate :SERial[*n*][:RECeive] :BAUD?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:BAUD? [MIN | MAX] returns:

- The current baud rate setting if no parameter is sent.
- The maximum allowable setting if MAX is sent.
- The minimum allowable setting if MIN is sent.

Example

Query the Current Baud Rate

```
SYST:COMM:SER0:BAUD?
enter statement                    Statement enters a numeric value.
```

:COMMunicate :SERial[*n*][:RECeive] :BITS

SYSTem:COMMunicate:SERial[*n*][:RECeive]:BITS <*bits*> sets the number of bits to be used to transmit and receive data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*bits*>	numeric	7 | 8 | MIN | MAX	none

Comments

- Attempting to set *bits* to other than those values shown will result in an `Error -222, "Data out of range"`.
- While this command operates independently of either the …PARity <*type*> or …SBITs commands, there are two combinations which are disallowed because of their data frame bit width. The following table shows the possible combinations:

…BITS	…PARity <*type*>	…SBITs	Frame Bits
7	NONE	1	9 - disallowed
7	NONE	2	10
7	Yes	1	10
7	Yes	2	11
8	NONE	1	10
8	NONE	2	11
8	Yes	1	11
8	Yes	2	12 - disallowed

- DIAG:BOOT:COLD will set …BITS to 8.
- **Related Commands:** SYST:COMM:SER[*n*][:REC]:PAR
- ***RST Condition:** No change.

Example

Configure Data Width to 7 Bits

```
SYST:COMM:SER0:BITS 7
```

:COMMunicate :SERial[*n*][:RECeive] :BITS?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:BITS? [MIN | MAX] returns:

- The current data width if no parameter is sent.
- The maximum allowable setting if MAX is sent.
- The minimum allowable setting if MIN is sent.

Example

Query the Current Data Width

```
SYST:COMM:SER0:BITS?
enter statement                    Statement enters 7 or 8.
```

:COMMunicate :SERial[*n*][:RECeive] :PACE[:PROTocol]

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PACE[:PROTocol] ***<protocol>*** enables or disables receive pacing (XON/XOFF) protocol.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*protocol*>	discrete	XON | NONE	none

Comments

- While …PROT is XON, the serial interface will send XOFF when the buffer reaches the …STOP threshold, and XON when the buffer reaches the …STARt threshold.
- For an HP E1324A, AUTO is always ON. In this case …[:RECeive]:PACE will also set …TRAN:PACE
- The XON character is Control Q (ASCII 17_{10}, 11_{16}), The XOFF character is Control S (ASCII 19_{10}, 13_{16}).
- DIAG:BOOT:COLD will set …PACE to XON.
- **Related Commands:** SYST:COMM:SER[*n*][:REC]:PACE:THR:STARt, SYST:COMM:SER[*n*][:REC]:PACE:THR:STOP, SYST:COMM:SER[*n*]TRAN:AUTO
- ***RST Condition:** No change.

Example

Enable XON/XOFF Handshaking

```
SYST:COMM:SER0:PACE:PROT  XON
```

:COMMunicate :SERial[*n*][:RECeive] :PACE[:PROTocol]?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PACE[:PROTocol]? returns the current receive pacing protocol.

Example

See if XON/XOFF Protocol is Enabled

```
SYST:COMM:SER0:PACE:PROT?
enter statement                    Statement enters the string "XON"
                                   or "NONE".
```

:COMMunicate :SERial[*n*][:RECeive] :PACE:THReshold :STARt

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PACE:THReshold:STARt <*char_count*> configures the input buffer level at which the specified interface may send the XON character (ASCII 11_{16}), assert the DTR line, and/or assert the RTS line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*char_count*>	numeric	1 through 99 for built-in, 1 through 8191 for HP E1324A	none

Comments

- To determine the size of the input buffer of the serial interface you are using, send SYST:COMM:SER[*n*]:PACE:THR:STARt? MAX. The returned value will be the buffer size less one.
- …STARt must be set to less than …STOP.
- The …THR:STAR command has no effect unless …PACE:PROT XON, …CONT:DTR IBF, or …CONT:RTS IBF has been sent.
- **Related Commands:** SYST:COMM:SER[*n*][:REC]:PACE[:PROT] XON | NONE, SYST:COMM:SER[*n*]:CONT:DTR, SYST:COMM:SER[*n*]:CONT:RTS
- ***RST Condition:** No change.

Example

Set Interface to Send XON When Input Buffer Contains 10 Characters

```
SYST:COMM:SER0:PACE:PROT XON
SYST:COMM:SER0:PACE:THR:STAR 10
```

:COMMunicate :SERial[*n*][:RECeive] :PACE:THReshold :STARt?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PACE:THReshold:STARt [MIN | MAX] returns:

- The current start threshold if no parameter is sent.
- The maximum allowable setting if MAX is sent.
- The minimum allowable setting if MIN is sent.

Comments

- To determine the size of the input buffer of the serial interface you are using, send SYST:COMM:SER[*n*]:PACE:THR:STARt? MAX. The returned value will be the buffer size less one.

Example

Return Current Start Threshold

```
SYST:COMM:SER0:PACE:THR:STAR?   Query for threshold value.
enter statement                 Statement enters a numeric value.
```

:COMMunicate :SERial[*n*][:RECeive] :PACE:THReshold :STOP

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PACE:THReshold:STOP <*char_count*> configures the input buffer level at which the specified interface may send the XOFF character (ASCII 13_{16}), de-assert the DTR line, and/or de-assert the RTS line.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*char_count*>	numeric	1 through 99 for built-in, 1 through 8191 for HP E1324A	none

Comments

- To determine the size of the input buffer of the serial interface you are using, send SYST:COMM:SER[*n*]:PACE:THR:STOP MAX. The returned value will be the buffer size less one.
- …STOP must be set to greater than …STARt.
- The …THR:STOP command has no effect unless …PACE:PROT XON, …CONT:DTR IBF, or …CONT:RTS IBF has been sent.
- **Related Commands:** SYST:COMM:SER[*n*][:REC]:PACE[:PROT] XON | NONE, SYST:COMM:SER[*n*]:CONT:DTR, SYST:COMM:SER[*n*]:CONT:RTS
- ***RST Condition:** No change.

Example

Set Interface to Send XOFF When Input Buffer Contains 80 Characters

```
SYST:COMM:SER0:PACE:THR:STOP 80
```

:COMMunicate :SERial[*n*][:RECeive] :PACE:THReshold :STOP?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PACE:THReshold:STOP? [MIN | MAX] returns:

- The current stop threshold if no parameter is sent.
- The maximum allowable setting if MAX is sent.
- The minimum allowable setting if MIN is sent.

Comments

- To determine the size of the input buffer of the serial interface you are using, send SYST:COMM:SER[*n*]:PACE:THR:STOP? MAX. The returned value will be the buffer size less one.

Example

Return Current Stop Threshold

```
SYST:COMM:SER0:PACE:THR:STOP?   Query for threshold.
enter statement                 Statement enters a numeric value.
```

:COMMunicate :SERial[*n*][:RECeive] :PARity

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PARity ***<type>*** configures the type of parity to be checked for received data, and generated for transmitted data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*type*>	discrete	EVEN | ODD | ZERO | ONE | NONE	none

Comments

- Attempting to set *type* to other than the values shown results in `Error -222, "Data out of range"`.
- The following table defines each value of <*type*>:

Value	Definition
EVEN	If …PARity:CHECk is ON, the received parity bit must maintain even parity. The transmitted parity bit will maintain even parity.
ODD	If …PARity:CHECk is ON, the received parity bit must maintain odd parity. The transmitted parity bit will maintain odd parity.
ZERO	If …PARity:CHECk is ON, the received parity bit must be a zero. The transmitted parity bit will be a zero.
ONE	If …PARity:CHECk is ON, the received parity bit must be a logic one. The transmitted parity bit will be a logic one.
NONE	A parity bit must not be received in the serial data frame. No parity bit will be transmitted.

- While this command operates independently of either the …BITS or …SBITs commands, there are two combinations which are disallowed because of their data frame bit width. The following table shows the possible combinations:

…BITS	…PARity <*type*>	…SBITs	Frame Bits
7	NONE	1	9 - disallowed
7	NONE	2	10
7	Yes	1	10
7	Yes	2	11
8	NONE	1	10
8	NONE	2	11
8	Yes	1	11
8	Yes	2	12 - disallowed

- Received parity will not be checked unless …PAR:CHEC ON is has been sent. Transmitted data will include the specified parity whether …PAR:CHEC is ON or OFF.
- DIAG:BOOT:COLD will set …PARity to NONE.

- **Related Commands:**
  SYST:COMM:SER[*n*][:REC]:PAR:CHEC 1 | 0 | ON | OFF,
  SYST:COMM:SER[*n*][:REC]:BITS 7 | 8,
  SYST:COMM:SER[*n*][:REC]:SBIT 1 | 2,

- ***RST Condition:** No change.

Example **Set Parity Check/Generation to ODD**

```
SYST:COMM:SER0:PAR  ODD          Set parity type.
SYST:COMM:SER0:PAR:CHEC  ON      Enable parity check/generation.
```

:COMMunicate :SERial[*n*][:RECeive] :PARity?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PARity? ***<type>*** returns the *type* of parity checked and generated.

Example **What Type of Parity Checking is Set?**

```
SYST:COMM:SER0:PAR?              Ask for parity type.
enter statement                  Returns the string EVEN, ODD,
                                 ZERO, ONE, or NONE.
```

:COMMunicate :SERial[*n*][:RECeive] :PARity:CHECk

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PARity:CHECk ***<check_cntrl>*** controls whether or not the parity bit in received serial data frames will be considered significant.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*check_cntrl*>	boolean	0 | 1 | OFF | ON	none

Comments

- When *check_cntrl* is set to 0 or OFF, received data is not checked for correct parity. Transmitted data still includes the type of parity configured with … PARity <*type*>.

- DIAG:BOOT:COLD will set … CHECk to OFF.

- **Related Commands:**
  SYST:COMM:SER[*n*][REC]:PAR <*type*>

- ***RST Condition:** No change.

Example **Set Parity Check to ON**

```
SYST:COMM:SER0:PAR:CHEC  ON
```

:COMMunicate :SERial[*n*][:RECeive] :PARity:CHECk?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:PARity:CHECk? returns the state of parity checking.

Example

Query Parity Checking

```
SYST:COMM:SER0:PAR:CHEC?
enter statement                    Statement enters 0 or 1.
```

:COMMunicate :SERial[*n*][:RECeive] :SBITs

SYSTem:COMMunicate:SERial[*n*][:RECeive]:SBITs <*sbits*> sets the number of stop bits to be used to transmit and receive data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*sbits*>	numeric	1 | 2 | MIN | MAX	none

Comments

- Attempting to set *sbits* to other than those values shown will result in an `Error -222, "Data out of range"`.
- While this command operates independently of either the …BITS or …PARity <*type*> commands, there are two combinations which are disallowed because of their data frame bit width. The following table shows the possible combinations:

…BITS	…PARity <*type*>	…SBITs	Frame Bits
7	NONE	1	9 - disallowed
7	NONE	2	10
7	Yes	1	10
7	Yes	2	11
8	NONE	1	10
8	NONE	2	11
8	Yes	1	11
8	Yes	2	12 - disallowed

- DIAG:BOOT:COLD will set …SBITs to 1.
- **Related Commands:** SYST:COMM:SER[*n*][:REC]:BAUD
- ***RST Condition:** No change.

Example

Configure for 2 Stop Bits

```
SYST:COMM:SER0:SBITS 2
```

:COMMunicate :SERial[*n*][:RECeive] :SBITs?

SYSTem:COMMunicate:SERial[*n*][:RECeive]:SBITs? [MIN | MAX] returns:

- The current stop bit setting if no parameter is sent.
- The maximum allowable setting if MAX is sent.
- The minimum allowable setting if MIN is sent.

Example

Query the Current Stop Bit Configuration

```
SYST:COMM:SER0:SBITs?          :REC is implied.
enter statement                Statement enters 1 or 2.
```

:COMMunicate :SERial[*n*]:TRANsmit :AUTO

SYSTem:COMMunicate:SERial[*n*]:TRANsmit:AUTO *<auto_cntrl>* when ON, sets the transmit pacing mode to be the same as that set for receive pacing. When OFF, the transmit pacing mode may be set independently of the receive pacing mode.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<auto_cntrl>	boolean	0 | 1 | ON | OFF	none

Comments

- For an HP E1324A, AUTO is always ON. Trying to set OFF or 0 will generate an error.
- DIAG:BOOT:COLD will set … AUTO to ON.
- **Related Commands:** SYST:COMM:SER[*n*][:REC]:PACE[:PROT], SYST:COMM:SER[*n*]:TRAN:PACE[:PROT]
- ***RST Condition:** …TRAN:AUTO ON

Example

Link Transmit Pacing with Receive Pacing

```
SYST:COMM:SER0:TRAN:AUTO ON
```

:COMMunicate :SERial[*n*]:TRANsmit :AUTO?

SYSTem:COMMunicate:SERial[*n*]:TRANsmit:AUTO? returns the current state of receive to transmit pacing linkage.

Comments

- For an HP E1324A, AUTO is always ON. In this case …AUTO? will always return a 1.

Example

Query if AUTO is ON or OFF

```
SYST:COMM:SER0:TRAN:AUTO?
enter statement                Statement enters the number 1 or 0.
```

:COMMunicate :SERial[*n*]:TRANsmit :PACE[:PROTocol]

SYSTem:COMMunicate:SERial[*n*]:TRANsmit:PACE[:PROTocol] <*protocol*> enables or disables the transmit pacing (XON/XOFF) protocol.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*protocol*>	discrete	XON | NONE	none

Comments

- For an HP E1324A, AUTO is always ON. In this case, …TRAN:PACE will also set …[RECeive]:PACE.
- Receipt of an XOFF character (ASCII 19_{10}, 13_{16}) will hold off transmission of data until an XON character (ASCII 17_{10}, 11_{16}) is received.
- DIAG:BOOT:COLD will set …PACE to XON.
- **Related Commands:** SYST:COMM:SER[*n*]:TRAN:AUTO
- ***RST Condition:** No change.

Example

Set XON/XOFF Transmit Pacing

```
SYST:COMM:SER0:TRAN:PACE:PROT  XON
```

:COMMunicate :SERial[*n*]:TRANsmit :PACE[:PROTocol]?

SYSTem:COMMunicate:SERial[*n*]:TRANsmit:PACE[:PROTocol]? returns the current transmit pacing protocol.

Example

Check Transmit Pacing Protocol

```
SYST:COMM:SER0:TRAN:PACE:PROT?
enter statement                    Statement enters the string "XON"
                                   or "NONE"
```

:DATE

SYSTem:DATE **<*year*>,<*month*>,<*day*>** sets the command module's internal calendar.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*year*>	numeric	Must round to 1980 to 2079.	none
<*month*>	numeric	Must round to 1 to 12.	none
<*day*>	numeric	Must round to 1 through last day of month.	none

Comments

- The upper limit on the day parameter is dependent on the month parameter and may be dependent on the year parameter in the case of a leap year.
- **Related Commands:** SYST:TIME, SYST:TIME?, SYST:DATE?
- ***RST Condition:** *RST does not change the setting of the calendar.

Example

Set the System Date

```
SYST:DATE 1996,06,08          Set June 8, 1996.
```

:DATE?

SYSTem:DATE? [MAX | MIN,MAX | MIN,MAX | MIN] returns:

When no parameter is sent: the current system date in the form +YYYY,+MM,+DD, where YYYY can be the year 1980 through 2079, MM can be the month 1 through 12, and DD can be the day 1 through 31.

When parameters are sent: the minimum or maximum allowable values for each of the three parameters. The parameter count must be three.

Example

Query the System Date

```
SYST:DATE?                    Ask for current date.
input values of year,month,day  Read back date.
```

:ERRor?

SYSTem:ERRor? queries the system's error queue. The response format is: **<error number>,"<error description string>"**.

Comments

- As system errors are detected, they are placed in the System instrument error queue. The error queue is first in, first out. This means that if several error messages are waiting in the queue, each SYST:ERR? query will return the oldest error message, and that message will be deleted from the queue.
- If the error queue fills to 30 entries, the last error in the queue is replaced with `Error -350, "Too many errors"`. No further errors are accepted by the queue until space becomes available using SYST:ERR?, or the queue is cleared using *CLS.
- The SYST:ERR? command can be used to determine if any configuration errors occurred during the power-on sequence.
- When SYST:ERR? is sent while the error queue is empty, the System instrument responds with `+0,"No error"`.
- **Related Commands:** *ESE, *ESR?, *SRE
- ***RST Condition:** Error queue is cleared.

Example

Read All Error Messages From, and Empty the Error Queue

loop statement	*Loop to read all errors.*
SYST:ERR?	*Ask for error message.*
enter statement	*Input the error (a number), and error message (a string)*
until statement	*until error number is 0.*

:TIME

SYSTem:TIME ***<hour>***,***<minute>***,***<second>*** sets the command module's internal clock.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<hour>	numeric	Must round to 0 to 23.	none
<minute>	numeric	Must round to 0 to 59.	none
<second>	numeric	Must round to 0 to 60.	none

Comments

- **Related Commands:** SYST:DATE, SYST:DATE?, SYST:TIME?
- ***RST Condition:** *RST does not change the command module's real time clock.

Example

Set the System Time

SYST:TIME 14,30,20	*Set 2:30:20 PM.*

:TIME?

SYSTem:TIME? [MAX | MIN,MAX | MIN,MAX | MIN] returns:

When no parameter is sent: the current system time is in the form +HH,+MM,+SS, where HH can be 0 through 23 hours, MM can be 0 through 59 minutes, and SS can be 0 through 60 seconds.

When parameters are sent: the minimum or maximum allowable values for each of the three parameters are returned. The parameter count must be three.

Example

Query the System Time

```
SYST:TIME?                    Ask for current time.
input values of hour,min,sec  Read back time.
```

:VERSion?

SYSTem:VERSion? returns the SCPI version for which this instrument complies.

Comments

- The returned information is in the format: `YYYY.R`; where `YYYY` is the year, and `R` is the revision number within that year.
- **Related Commands:** *IDN?

Example

Determine Compliance Version for this Instrument

```
SYST:VERS?
enter statement               Statement enters 1990.0
```

VXI

The VXI command subsystem provides for:

- Determining the number, type, and logical address of the devices (instruments) installed in the C-size mainframe.
- Direct access to VXIbus A16 registers within devices installed in the mainframe.
- Sending commands using the word serial protocol.
- Access to message-based devices from an RS-232 terminal.

Subsystem Syntax

```
VXI
    :CONFigure
        :CTABle <address>
        :CTABle?
        :DCTable <address>
        :DCTable?
        :DLADdress?
        :DLISt?[<logical_addr>]
        :DNUMber?
        :ETABle <address>
        :ETABle?
        :HIERarchy?
            :ALL?
        :INFormation?
            :ALL?
        :ITABle <address>
        :ITABle?
        :LADDress?
            :MEXTender?
        :MEXTender
            :ECLTrg<n>  <direction>
            :INTerrupt<n>  <direction>
            :TTLTrg<n>  <direction>
        :MTABle <address>
        :MTABle?
        :NUMber?
            :MEXTender?
    :QUERy? <logical_addr>
    :READ? <logical_addr>,<register_addr>
    :RECeive
        [:MESSage]? <logical_addr>[,<end_of_msg>]
    :REGister
        :READ? <register>
        :WRITe <register>,<data>
    :RESet <logical_addr>
    :RESet?
    :ROUTe
        :ECLTrg<n>
        :INTerrupt<n>
        :TTLTrg<n>
    :SELect <logical_addr>
    :SELect?
    :SEND
```

:COMMand <*logical_addr*>,<*command*>[,<*data*>]
:COMMand? <*logical_addr*>,<*command*>[,<*data1*>[,<*data2*>]]
[:MESSage] <*logical_addr*>,"<*msg_string*>"[,<*end_of_flag*>]
:WRITe <*logical_addr*>,<*register_addr*>,<*data*>
:WSProtocol
:COMMand
:AHLine <*hand_id*>,<*line_number*>
:AILine <*int_id*>,<*line_number*>
:AMControl <*response_mask*>
:ANO
:ANY <*cmd_word*>
:BAVailable <*end_bit*>
:BNO <*top_level*>
:BREQuest
:CEVent <*enable*>,<*event_number*>
:CLR
:CLOCk
:CRESponse <*response_mask*>
:ENO
:GDEVice <*cmdr_laddr*>
:ICOMmander
:RDEVice <*logical_addr*>
:RHANdlers
:RHLine <*hand_id*>
:RILine <*int_id*>
:RINTerrupter
:RMODid
:RPERror
:RPRotocol
:RSAREa
:RSTB
:SLModid <*enable*>,<*modid*> (0-127)
:SLOCk
:SUModid <*enable*>,<*modid*> (0-63)
:TRIGger
:MESSage
:RECeive? <*count* | *terminator*>
:SEND <*msg_string*>[,(END | NEN)]
:QUERy
:AHLine? <*hand_id*>,<*line_number*>
:AILine? <*int_id*>,<*line_number*>
:AMControl? <*response_mask*>
:ANO?
:ANY? <*cmd_word*>
:BNO? <*top_level*>
:BREQuest?
:CEVent? <*enable*>,<*event_number*>
:CRESponse? <*response_mask*>
:ENO?
:RDEVice? <*logical_addr*>
:RHANdlers?
:RHLine? <*hand_id*>
:RILine? <*int_id*>
:RINTerrupter?
:RMODid?
:RPERror?

:RPRotocol?
:RSAREa?
:RSTB?
:SLModid? <*enable*>,<*modid*> (0-127)
:SUModid? <*enable*>,<*modid*> (0-63)
:RESPonse?

:CONFigure:CTABle

VXI:CONFigure:CTABle *<address>* links a user-defined commander/servant hierarchy table to the command module (resource manager) processor. The command module must be the acting resource manager in order for the table to be implemented.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	(DIAG:NRAM:ADDR?)	none

Comments

- Be certain that *address* specifies the starting address of the area in user RAM (allocated using DIAG:NRAM:CREate) where you stored the commander/servant hierarchy table.
- Tables must start on an even address. Note that DIAG:NRAM:CREate allocates RAM for the table with an even starting address.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- Setting *address* to 0 (zero) prevents the parameters defined by the table from being invoked when the system is rebooted, however, the table remains in user RAM.
- For more information see "User-Defined Commander/Servant Hierarchies" on page 39.
- **Related Commands:** DIAGnostic:NRAM:CREate, DIAGnostic:NRAM:ADDRess?, DIAGnostic:DOWNload, VXI:CONF:CTABle?

Example

Link a Commander/Servant Hierarchy Table to the Processor

DIAG:NRAM:CRE <*size*>	*Allocate space for table in user RAM.*
DIAG:BOOT	*Reboot system to complete allocation.*
DIAG:NRAM:ADDR?	*Get starting address of table (RAM segment).*
DIAG:DOWN <*address*>,<*data*>	*Download data into table.*
VXI:CONF:CTABle *<address>*	*Link table to processor.*
DIAG:BOOT	*Reboot system to implement table.*

:CONFigure:CTABle?

VXI:CONFigure:CTABle? returns the starting address of the user's commander/servant hierarchy table.

Example

Query Address of the Commander/Servant Hierarchy Table

VXI:CONF:CTABle?	*Ask for address.*
enter statement	*Return address.*

:CONFigure:DCTable

VXI:CONFigure:DCTable *<address>* links a user-defined dynamic configuration table to the command module (resource manager) processor. The command module must be the acting resource manager in order for the table to be implemented.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<address>	numeric	(DIAG:NRAM:ADDR?)	none

Comments

- Be certain that *address* specifies the starting address of the area in user RAM (allocated using DIAG:NRAM:CREate) where you stored the dynamic configuration table data.
- Tables must start on an even address. Note that DIAG:NRAM:CREate allocates RAM for the table with an even starting address.
- *<address>* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- Setting *address* to 0 (zero) prevents the parameters defined by the table from being invoked when the system is rebooted, however, the table remains in user RAM.
- For more information see "User-Defined Dynamic Configuration" on page 23.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:ADDRess?, DIAG:DOWNload, VXI:CONF:DCTable?

Example

Link a Dynamic Configuration Table to the Processor

DIAG:NRAM:CRE *<size>*	*Allocate space for table in user RAM.*
DIAG:BOOT	*Reboot system to complete the allocation.*
DIAG:NRAM:ADDR?	*Get starting address of table (RAM segment).*
DIAG:DOWN *<address>,<data>*	*Download data into table.*
VXI:CONF:DCTable *<address>*	*Link table to processor.*
DIAG:BOOT	*Reboot system to implement table.*

:CONFigure:DCTable?

VXI:CONFigure:DCTable? returns the starting address of the user's dynamic configuration table.

Example

Query Address of Dynamic Configuration Table

```
VXI:CONF:DCTable?          Ask for address.
enter statement            Return address.
```

:CONFigure :DLADdress?

VXI:CONFigure:DLADdress? returns a comma (,) separated decimal numeric list of device logical addresses currently installed in the mainframe. If the command module is not the resource manager, it only returns the logical addresses of the devices in its servant area.

Comments

- Use the VXI:CONF:DNUM? command to determine the number of values which will be returned by VXI:CONF:DLAD?.
- Use each of the logical addresses returned by VXI:CONF:DLAD? with VXI:CONF:DLIS? to determine the types of devices installed.
- VXI:CONF:DEVICELAD? is also accepted.
- This command has been retained for compatibility with existing programs. For new programs you should use the VXI:CONF:LADDress? command.
- **Related Commands:** VXI:CONF:DLISt?, VXI:CONF:DNUMber?, VXI:CONF:LADDress?

Example

Determine the Device Addresses within the System

```
VXI:CONF:DLAD?             Query for list of addresses.
enter statement            List of addresses.
```

:CONFigure:DLISt?

VXI:CONFigure:DLISt? [<*logical_addr*>] returns information about the device specified by *logical_addr*. Response data is in the form:

n1, n2, n3, n4, n5, n6, c1, c2, c3, c4, c5, s1, s2, s3, s4

Where the fields above are defined as:

n fields Indicate numeric data response fields.
c fields Indicate character data response fields.
s fields Indicate string data response fields.

n1 Device's Logical Address. A number from 0 to 255.

n2 Commander's Logical Address. A number from -1 to 255; -1 means this device has no commander.

n3 Manufacturer's ID. A number from 0 to 4095.

n4 Model Code. A number from 0 to 65535, chosen by the manufacturer to signify the model of this device.

n5 Slot Number. A number between -1 and the number of slots in this mainframe; -1 indicates that the slot associated with this device is unknown. This is always -1 for B size mainframes.

n6 Slot 0 Logical Address. A number from 0 to 255.

c1 Device Class. 3 data characters; EXT|HYB|MEM|MSG|REG|VME.
EXT = Extended device, HYB = Hybrid device (e.g., IBASIC
MEM = Memory device, MSG = Message-based device
REG = Register-based device, VME = VME device

c2 Memory Space. Up to 4 data characters; A16|A24|A32|NONE|RES.
A16 = A16 addressing mode, A24 = A24 addressing mode,
A32 = A32 addressing mode, NONE = no addressing mode,
RES = reserved.

c3 Memory Offset. 10 data characters which define the base address of the A24 or A32 address space on the device. This value is expressed in hex format (first two characters are #H).

c4 Memory Size. 10 data characters which define the size of the A24 or A32 address space in bytes. This value is expressed in hex format (first two characters are #H).

c5 Pass/Failed. Up to 5 data characters which define the status of the device; FAIL | IFAIL | PASS | READY. FAIL = failed self-test,
IFAIL = configuration register initialization fails,
PASS = self-test passed,
READY = ready to receive commands

s1 Extended Field 1. Not currently used; returns ""

s2 Extended Field 2. Not currently used; returns ""

s3 Extended Field 3. Not currently used; returns ""

s4 Manufacturer's Specific Comments. Up to 80 character string contains manufacturer specific data in string response data format. This field is sent with a 488.2 string response data format, and will contain the instrument name and its IEEE 488.1 secondary address unless a start-up error is detected. In that case, this field will contain one or more error codes in the form "CNFG ERROR: n, m, ...,z". Table B-3 in Appendix B for a complete list of these codes.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	numeric	0 to 255 (or nothing)	none

Comments

- When *logical_addr* is not specified, VXI:CONF:DLIS? returns information for each of the devices installed, separated by semicolons (;). If the command module is not the resource manager, it returns information on only the devices in its servant area.
- Cards which are part of a combined instrument such as a switchbox or scanning voltmeter always return the same manufacturer's comments as the first card in the instrument. Information in the other fields correspond to the card for which the logical address was specified.
- This command has been retained for compatibility with existing programs. For new programs you should use the VXI:CONF:INF? and VXI:CONF:HIER? commands.
- **Related Commands:** VXI:CONF:DLADdress?, VXI:CONF:DNUMber?, VXI:CONF:INFormation?, VXI:CONF:HIERarchy?

Example

Query the device list for the System Instrument

```
dimension string[1000]          ! String size large in case of multiple device list.
VXI:CONF:DLIS? 0                ! Ask for the device list for the System instrument.
enter string                    ! Enter return data into string.
```

Example response data (no error):
```
+0, -1, +4095, +1301, +0, +0, HYB, NONE, #H00000000, #H00000000, READY, "", "", "", "SYSTEM INSTALLED AT SECONDARY ADDR 0"
```

Example response data (with error):
```
+255, +0, +4095, +65380, -1, +0, REG, A16, #H00000000, #H00000000, READY, "", "", "", "CNFG ERROR: 11"
```

:CONFigure :DNUMber?

VXI:CONFigure:DNUMber? returns the number of devices installed in the mainframe (including the System instrument itself). If the command module is not the resource manager, it returns the number of devices in its servant area.

Comments

- Use the VXI:CONF:DNUM? command to determine the number of values which will be returned by VXI:CONF:DLAD?.
- This command has been retained for compatibility with existing programs. For new programs you should use the VXI:CONF:NUMBer? command.
- **Related Commands:** VXI:CONF:DLADdress?, VXI:CONF:DLISt?

Example

Determine the Number of Devices Within the System

```
VXI:CONF:DNUM?                  ! Query the number of devices.
enter statement                 ! Input number of devices.
```

:CONFigure:ETABle

VXI:CONFigure:ETABle ***<address>*** links a user-defined extender table to the command module (resource manager) processor. The command module must be the acting resource manager in order for the table to be implemented.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<address>	numeric	(DIAG:NRAM:ADDR?)	none

Comments

- Be certain that *address* specifies the starting address of the area in user RAM (allocated using DIAG:NRAM:CREate) where you stored the extender table.
- Tables must start on an even address. Note that DIAG:NRAM:CREate allocates RAM for the table with an even starting address.
- *<address>* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- Setting *address* to 0 (zero) prevents the parameters defined by the table from being invoked when the system is rebooted, however, the table remains in user RAM.
- For more information see the "User-Defined Logical Address and Memory Windows" on page 31.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:ADDRess?, DIAG:DOWNload, VXI:CONF:ETABle?

Example

Link an Extender Table to the Processor

```
DIAG:NRAM:CRE <size>          Allocate space for table in user RAM.
DIAG:BOOT                     Reboot system to complete allocation.
DIAG:NRAM:ADDR?               Get starting address of table (RAM segment).
DIAG:DOWN <address>,<data>    Download data into table.
VXI:CONF:ETAB <address>       Link table to processor.
DIAG:BOOT                     Reboot system to implement table.
```

:CONFigure:ETABle?

VXI:CONFigure:ETABle? returns the starting address of the user's extender table.

Example

Query Address of the Extender Table

```
VXI:CONF:ETABle?              Ask for address.
enter statement               Return address.
```

:CONFigure :HIERarchy?

VXI:CONFigure:HIERarchy? returns current hierarchy configuration information about the selected logical address. The individual fields of the response are comma separated. If the information about the selected logical address is not available from the destination device (that is, the requested device is not in the mainframe or the command module's servant area) then `Error -224, "Illegal parameter value"` will be set and no response data will be sent.

Comments

- This command returns the following values:

  Logical address: An integer between -1 and 255 inclusive. -1 indicates that the device has no logical address.

  Commander's logical address: An integer between -1 and 255 inclusive. -1 indicates that the device has no commander or that the commander is unknown.

  Interrupt handlers: A comma (,) separated list of seven integers between 0 and 7 inclusive. Interrupt lines 1–7 are mapped to the individual return values. 0 (zero) is used to indicate that the particular interrupt handler is not configured. A set of return values of `0,0,0,5,2,0,6` would indicate that:

  - handler 4 is configured to handle interrupts on line 5
  - handler 5 is configured to handle interrupts on line 2
  - handler 7 is configured to handle interrupts on line 6
  - handlers 1, 2, 3, and 6 are not configured

  Interrupters: A comma (,) separated list of seven integers between 0 and 7 inclusive. Interrupt lines 1–7 are mapped to the individual return values. 0 (zero) indicates that the particular interrupter is not configured. A set of return values of `0,0,0,5,2,0,6` would indicate that:

  - interrupter 4 is configured to handle interrupts on line 5
  - interrupter 5 is configured to handle interrupts on line 2
  - interrupter 7 is configured to handle interrupts on line 6
  - interrupters 1, 2, 3, and 6 are not configured

  Pass/Failed: An integer which contains the pass/fail status of the specified device encoded as follows:

  0 = FAIL, **1 =** IFAIL, **2 =** PASS, **3 =** READY

  Manufacturer specific comment: Up to an 80 character quoted string that contains manufacturer specific data. It is sent with a 488.2 string response data format, and will contain the instrument name and its IEEE 488.1 secondary address unless a start-up error is detected. In that case, this field will contain one or more error codes in the form "CNFG ERROR: n, m, ...,z". See Table B-3 in Appendix B for a complete list of these codes.

- Cards which are part of a combined instrument such as a switchbox or scanning voltmeter always return the same manufacturer's comments as the first card in the instrument. Information in the other fields correspond to the card for which the logical address was specified.

- **Related Commands:** VXI:SELect, VXI:CONF:HIERarchy:ALL?, VXI:CONF:LADDress?

:CONFigure :HIERarchy:ALL?

VXI:CONFigure:HIERarchy:ALL? returns the configuration information about all logical addresses in the mainframe, or the devices in the command module's servant area if the command module is not the resource manager. The information is returned in the order specified in the response to VXI:CONF:LADDress?. The information about multiple logical addresses will be semicolon (;) separated and follow the IEEE 488.2 response message format. Individual fields of the output are comma (,) separated.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:HIERarchy?, VXI:CONF:LADDress?

:CONFigure :INFormation?

VXI:CONFigure:INFormation? returns the static information about the selected logical address (see VXI:SELect). The individual fields of the response are comma (,) separated. If the information about the selected logical address is not available from the destination device (that is, the requested device is not in the mainframe or the command module's servant area) then `Error -224, "Illegal parameter value"` will be set and no response data will be sent. The command returns the following values:

Logical address: An integer between -1 and 255 inclusive. -1 indicates that the device has no logical address.

Manufacturer ID: An integer between -1 and 4095 inclusive. -1 indicates that the device has no Manufacturer ID.

Model code: An integer between -1 and 65535 inclusive. -1 indicates that the device has no model code.

Device class: An integer between 0 and 5 inclusive.
0 = VXIbus memory device, 1 = VXIbus extended device,
2 = VXIbus message based device, 3 = VXIbus register-based device,
4 = Hybrid device, 5 = Non-VXIbus device.

Address space: An integer between 0 and 15 inclusive, which is the sum of the binary weighted codes of the address space(s) occupied by the device.
1 = The device has A16 registers, 2 = The device has A24 registers,
4 = The device has A32 registers, 8 = The device has A64 registers.

A16 memory offset: An integer between -1 and 65535 inclusive. Indicates the base address for any A16 registers (other than the VXIbus defined registers) which are present on the device. -1 indicates that the device has no A16 memory.

A24 memory offset: An integer between -1 and 16777215 inclusive. Indicates the base address for any A24 registers which are present on the device. -1 indicates that the device has no A24 memory.

A32 memory offset: An integer between -1 and 4294967295 inclusive. Indicates the base address for any A32 registers which are present on the device. -1 indicates that the device has no A32 memory.

A16 memory size: An integer between -1 and 65535 inclusive. Indicates the number of bytes reserved for any A16 registers (other than the VXIbus defined registers) which are present on the device. -1 indicates that the device has no A16 memory.

A24 memory size: An integer between -1 and 16777215 inclusive. Indicates the number of bytes reserved for any A24 registers which are present on the device. -1 indicates that the device has no A24 memory.

A32 memory size: An integer between -1 and 4294967295 inclusive. Indicates the number of bytes reserved for any A32 registers which are present on the device. -1 indicates that the device has no A32 memory.

Slot number: An integer between -1 and the number of slots which exist in the cage. -1 indicates that the slot which contains this device is unknown.

Slot 0 logical address: An integer between -1 and 255 inclusive. -1 indicates that the Slot 0 device associated with this device is unknown.

Subclass: An integer representing the contents of the subclass register. -1 indicates that the subclass register is not defined for this device.

Attribute: An integer representing the contents of the attribute register. -1 indicates that the attribute register is not defined for this device.

Manufacturer specific comment: Up to an 80 character quoted string that contains manufacturer specific data. It is sent with a 488.2 string response data format, and will contain the instrument name and its IEEE 488.1 secondary address unless a start-up error is detected. In that case, this field will contain one or more error codes in the form "CNFG ERROR: n, m, ...,z". See Table B-3 in Appendix B for a complete list of these codes.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:INFormation:ALL?, VXI:CONF:LADDress?

Example

Get Static Information on the Currently Selected Logical Address

```
VXI:SEL 0                    Select the logical address.
VXI:CONF:INF?                Ask for data.
enter statement              Return data.
```

:CONFigure :INFormation:ALL?

VXI:CONFigure:INFormation:ALL? returns the static information about all logical addresses. The information is returned in the order specified in the response to VXI:CONF:LADDress?. The information about multiple logical addresses will be semicolon (;) separated and follow the IEEE 488.2 response message format. Individual fields of the output are comma (,) separated.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:INFormation?, VXI:CONF:LADDress?

:CONFigure:ITABle

VXI:CONFigure:ITABle *<address>* links a user-defined interrupt line allocation table to the command module (resource manager) processor. The command module must be the acting resource manager in order for the table to be implemented.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<address>	numeric	(DIAG:NRAM:ADDR?)	none

Comments

- Be certain that *address* specifies the starting address of the area in User RAM (allocated using DIAG:NRAM:CREate) where you stored the interrupt line allocation table data.
- Tables must start on an even address. Note that DIAG:NRAM:CREate allocates RAM for the table with an even starting address.
- *<address>* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- Setting *address* to 0 (zero) prevents the parameters defined by the table from being invoked when the system is re-booted, however, the table remains in user RAM.
- For more information see the section "User-Defined Interrupt Line Allocation Table" on page 54.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:ADDRess?, DIAG:DOWNload, VXI:CONF:ITABle?

Example

Link an Interrupt Line Allocation Table to the Processor

DIAG:NRAM:CRE *<size>*	*Allocate space for table in user RAM.*
DIAG:BOOT	*Reboot system to complete the allocation.*
DIAG:NRAM:ADDR?	*Get starting address of table (RAM segment).*
DIAG:DOWN *<address>,<data>*	*Download data into table.*
VXI:CONF:ITAB *<address>*	*Link table to processor.*
DIAG:BOOT	*Reboot system to implement.*

:CONFigure:ITABle?

VXI:CONFigure:ITABle? returns the starting address of the user's interrupt line allocation table.

Example

Query Address of Interrupt Line Allocation Table

```
VXI:CONF:ITABle?          Ask for address.
enter statement           Return address.
```

:CONFigure :LADDress?

VXI:CONFigure:LADDress? returns a comma (,) separated list of logical addresses of devices in the mainframe, or a list of devices in the command module's servant area if the command module is not the resource manager. This is an integer between 1 and 256 inclusive. The logical address of the device responding to the command will be the first entry in the list. If the command is received by a device other than the resource manager, the response will contain the logical address of the destination device followed by a list of devices which are immediate servants to the destination device.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:NUMBer?

:CONFigure :LADDress :MEXTender?

VXI:CONFigure:LADDress:MEXTender? returns a comma (,) separated list of logical addresses of mainframe extender devices in the system. This is an integer between 1 and 256 inclusive. If there are no extender devices in the system a `-1` will be returned. An error is reported if the command is received by a device other than the resource manager.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:NUMBer:MEXTender?

:CONFigure :MEXTender :ECLTrg<*n*>

VXI:CONFigure:MEXTender:ECLTrg<*n*> <*direction*> is used to configure the selected mainframe extender to direct the ECL trigger specified by <*n*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 or 1	none
<*direction*>	discrete	IN | OUT | NONE	none

Comments

- Select the logical address of the extender to access with the VXI:SELect command.
- The trigger line affected is specified in the ECLTrg<*n*> node of the command by an integer of 0 or 1. Integers greater than 1 will generate `Error -113, "Undefined header"`.
- A mainframe extender can direct a trigger line into or out of the VXIbus card cage (mainframe) that it is plugged into.
- If you specify NONE the trigger line will be disabled and will not be directed in or out.
- Some mainframe extender devices do not support some trigger lines. These commands will determine whether the specified trigger line is supported before it attempts to execute the command. If the trigger line is not supported a "trigger not supported" error will be returned.
- This command can only be executed by the System instrument in a command module that is serving as resource manager for the entire VXIbus system.
- **Related Commands:** VXI:CONF:MEXTender:INTerrupt, VXI:CONF:MEXTender:TTLTrg<*n*>, VXI:ROUTe:ECLTrg<*n*>

Example

Direct ECL trigger line 1 from a card cage with "child side" extender at logical address 5 to an extended card cage with a "parent side extender" of logical address 6.

```
VXI:SEL  5                      Select logical address 5.
VXI:CONF:MEXT:ECLT1  OUT        Configure the logical address 5
                                extender as OUT.
VXI:SEL 6                       Select logical address 6.
VXI:CONF:MEXT:ECLT1 IN          Configure the logical address 6
                                extender as IN.
```

:CONFigure :MEXTender :INTerrupt<*n*>

VXI:CONFigure:MEXTender:INTerrupt<*n*> <*direction*> is used to configure the selected mainframe extender to direct the interrupt line specified by <*n*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 or 1	none
<*direction*>	discrete	IN | OUT | NONE	none

Comments

- Select the logical address of the extender to access with VXI:SELect.
- The interrupt line affected is specified in the INTerrupt<*n*> node of the command by a number ranging from 1 to 7. Numbers less than 1 and greater than 7 will generate `Error -113, "Undefined header"`.
- A mainframe extender can direct an interrupt line into the VXIbus card cage (mainframe) that it is plugged into or it can direct the interrupt line out of the card cage.
- If you specify NONE the interrupt line will be disabled and will not be directed in or out.
- Some mainframe extender devices do not support directing interrupt lines. These commands will determine whether the specified interrupt line is supported before it attempts to execute the command. If the interrupt line is not supported, a trigger not supported error will be returned.
- This command can only be executed by the System instrument in a command module that is serving as resource manage for the entire VXIbus system.
- **Related Commands:** VXI:CONF:MEXTender:ECLTrg<*n*>, VXI:CONF:MEXTender:TTLTrg<*n*>, VXI:ROUTe:INTerrupt<*n*>

Example

Direct interrupt line 1 from a card cage with "child side" extender at logical address 5 to an extended card cage with a "parent side extender" of logical address 6.

```
VXI:SEL 5                    Select logical address 5.
VXI:CONF:MEXT:INT1 OUT       Configure the logical address 5
                             extender as OUT.
VXI:SEL 6                    Select logical address 6.
VXI:CONF:MEXT:INT1 IN        Configure the logical address 6
                             extender as IN.
```

:CONFigure :MEXTender :TTLTrg<*n*>

VXI:CONFigure:MEXTender:TTLTrg<*n*> <*direction*> is used configure the selected mainframe extender to direct the TTL trigger specified by <*n*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 through 1	none
<*direction*>	discrete	IN | OUT | NONE	none

Comments

- Select the logical address of the extender to access with VXI:SELect.
- The trigger line affected is specified in the TTLTrg<*n*> node of the command by a number ranging from 0 to 7. Numbers greater than 7 will generate `Error -113, "Undefined header"`.
- A mainframe extender can direct a trigger line into the VXIbus card cage (mainframe) that it is plugged into or it can direct the trigger line out of the card cage.
- If you specify NONE the trigger line will be disabled and will not be directed in or out.
- Some mainframe extender devices do not support some trigger lines. These commands will determine whether the specified trigger line is supported before it attempts to execute the command. If the trigger line is not supported, a "trigger not supported" error will be returned.
- This command can only be executed by the System instrument in a command module that is serving as resource manager for the entire VXIbus system.
- **Related Commands:** VXI:CONF:MEXTender:INTerrupt<*n*>, VXI:CONF:MEXTender:ECLTrg<*n*>, VXI:ROUTe:TTLTrg<*n*>

Example

Direct TTL trigger line 1 from a card cage with "child side" extender at logical address 5 to an extended card cage with a "parent side extender" of logical address 6.

```
VXI:SEL  5                      Select logical address 5.
VXI:CONF:MEXT:TTLT1  OUT        Configure the logical address 5
                                extender as OUT.
VXI:SEL 6                       Select logical address 6.
VXI:CONF:MEXT:TTLT1 IN          Configure the logical address 6
                                extender as IN.
```

:CONFigure:MTABle

VXI:CONFigure:MTABle <*address*> links a user-defined A24/A32 address allocation table to the command module (resource manager) processor. The command module must be the acting resource manager in order for the table to be implemented.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*address*>	numeric	(DIAG:NRAM:ADDR?)	none

Comments

- Be certain that *address* specifies the starting address of the area in user RAM (allocated using DIAG:NRAM:CREate) where you stored the A24/A32 address allocation table data.
- Tables must start on an even address. Note that DIAG:NRAM:CREate allocates RAM for the table with an even starting address.
- <*address*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- Setting *address* to 0 prevents the parameters defined by the table from being invoked when the system is rebooted, however, the table remains in user RAM.
- For more information see "Reserving A24/A32 Address Space" on page 48.
- **Related Commands:** DIAG:NRAM:CREate, DIAG:NRAM:ADDRress?, DIAG:DOWNload, VXI:CONF:MTABle?

Example

Link an A24/A32 Address Allocation Table to the Processor

DIAG:NRAM:CRE <*size*>	*Allocate space for table in user RAM.*
DIAG:BOOT	*Reboot system to complete the allocation.*
DIAG:NRAM:ADDR?	*Get starting address of table (RAM segment).*
DIAG:DOWN <*address*>,<*data*>	*Download data into table.*
VXI:CONF:MTAB <*address*>	*Link table to processor.*
DIAG:BOOT	*Reboot system to implement table.*

:CONFigure:MTABle?

VXI:CONFigure:MTABle? returns the starting address of the user's A24/A32 address allocation table.

Example

Query Address of A24/A32 Address Allocation Table

```
VXI:CONF:MTABle?          Ask for address.
enter statement           Return address.
```

:CONFigure:NUMBer?

VXI:CONFigure:NUMBer? returns the number of devices in the system when it is issued to a resource manager. This is an integer between 1 and 256 inclusive. If the command is received by a device that is not the resource manager, it returns the number of devices which are immediate servants to the destination device, including the destination device. For example, a commander with 3 servants would return a value of 4, or a resource manager for a system of 4 devices would return a value of 5.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:LADDress?

:CONFigure:NUMBer :MEXTender?

VXI:CONFigure:NUMBer:MEXTender? returns the number of devices in the system when it is issued to a resource manager. This is an integer between 1 and 256 inclusive, which indicates the number of mainframe extender devices in the system. If the command is received by a device other than the resource manager an error is reported.

Comments

- **Related Commands:** VXI:SELect, VXI:CONF:LADDress?, VXI:CONF:NUMBer?

:QUERy?

VXI:QUERy? *<logical_addr>* returns one 16-bit data word from the Data Low Register of the message-based device at *logical_addr*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<logical_addr>	numeric	Must round to 0 through 255.	none

Comments

- Send a Device Clear to "unlock" the System instrument in case the device at *logical_addr* does not respond.
- VXI:QUERy? can be used to read the response in the Data Low Register when the VXI:SEND:COMM command is ANY, and the command sent is a query.
- This command has been retained for compatibility with existing programs. For new programs you should use VXI:WSP:RESP?
- **Related Commands:** VXI:SEND:COMMand, VXI:WSProtocol:RESP?

Example Read the Data Low Register of Device at Logical Address 72

```
VXI:QUERY? 72                    Query value of Data Low Register.
enter statement                  Input 16-bit value.
```

:READ?

VXI:READ? *<logical_addr>*, *<register_addr>* allows access to the entire 64-byte A16 register address space for the device specified by *logical_addr*. Since the VXIbus system is byte-addressed, while the registers are 16-bits wide, registers are specified by even addresses only. This method of identifying registers follows the VXIbus standard format.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	numeric	Must round to 0 through 255.	none
<*register_addr*>	numeric	Must round to an even value from 0 through 62 ($3E_h$).	none

Comments

- Specifying an odd *register address* will cause `Error +2003, "Invalid word address"`.
- Specifying a *logical address* not currently in the system will cause `Error +2005, "No card at logical address"`.
- If the command module is the resource manager it can read from any device within the mainframe. If the command module is not the resource manager it can only read from devices within its servant area.
- <*logical_addr*> **must** be specified in decimal. <*register_addr*> **may** be specified in decimal, hex (#H), octal (#Q), or binary (#B).
- Accesses are 16-bit non-privileged data accesses.
- This command has been retained for compatibility with existing programs. For new programs you should use VXI:REG:READ?.
- **Related Commands:** VXI:WRITe, VXI:REGister:READ?

Example Read from One of a Device's Configuration Registers

```
VXI:READ? 8,0                    Read ID Register on device at
                                 logical address 8.
enter statement                  Enter value from Device Register.
```

:RECeive[:MESSage]?

VXI:RECeive[:MESSage]? <*logical_addr*>[,<*end_of_msg*>] receives a message from the message-based device at *logical_addr*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	numeric	Must round to 0 through 255.	none
<*end_of_msg*>	discrete/numeric	END | LF | CRLF | <*count*>	none

Comments

- A message ends when the condition specified by the *end_of _msg* parameter is met. When *end_of_msg* specifies a *count*, it can range from 1 through 2,147,483,647.
- The default *end_of_msg* parameter is END.
- VXI:REC? together with VXI:SEND can be used to communicate with message-based devices from an RS-232 monitor via the command module. If the command module is the resource manager, the message-based devices can be inside or outside its servant area. If the command module is not the resource manager, the message-based devices must be in the command module's servant area.
- VXI:REC? uses the Byte Transfer Protocol which uses the DIR and DOR bits in the Response Register. This protocol and DIR/DOR are described in the *VXIbus System Specifications*.
- Send a Device Clear to "unlock" the System instrument in case the device at *logical_addr* does not satisfy the *end_of_msg* condition (insufficient data for count, or no END | LF | CRLF).
- This command has been retained for compatibility with existing programs. For new programs you should use the VXI:WSP:MESS:REC? command
- **Related Commands:** VXI:SEND[:MESSage], VXI:WSProtocol:MESSage:RECeive?, VXI:WSProtocol:MESSage:SEND

Example

Query for Message from Module at Logical Address 16

```
VXI:SEND  16,"*IDN?"          Send command to device at logical address 16.
VXI:REC?  16                  Enter message.
```

:REGister:READ?

VXI:REGister:READ? *<register>* returns the contents of the specified 16-bit register at the selected logical address as an integer (see VXI:SELect).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<register>	numeric	Even numbers from 0 to 62 or register name (see below).	none

Comments

- The *register* parameter can be all even numbers from 0 to 62 inclusive (as a numeric value) or the following (optional) words:

  A16 Window: A16 Window Map Register (12)
  A24Low: A24 Pointer Low Register (18)
  A24High: A24 Pointer High Register (16)
  A24 Window: A24 Window Map Register (14)
  A32Low: A32 Pointer Low Register (22)
  A32High: A32 Pointer High Register (20)
  A32 Window: A32 Window Map Register (16)
  ATTRibute: Attribute Register (8)
  DHIGh: Data High Register (12)
  DLOW: Data Low Register (14)
  DTYPe: Device Type Register (2)
  ETConfigure: ECL Trigger Configuration Register (22)
  ICNF: Interrupt Configuration Register (18)
  ICONtrol: Interrupt Control Register (28)
  ID: ID Register (0)
  ISTatus: Interrupt Status Register (26)
  LAWindow: Logical Address Configuration Register (10)
  TTConfigure: TTL Trigger Configuration Register (20)
  MODid: MODID Register (8)
  OFFSet: Offset Register (6)
  PROTocol: Protocol Register (8)
  RESPonse: Response Register (10)
  SNHigh: Serial Number High Register (10)
  SNLow: Serial Number Low Register (12)
  STATus: Status Register (4)
  SUBClass: Subclass Register (30)
  UCONfigure: Utility Configuration Register (24)
  VNUMber: Version Number Register (14)

Note

The optional register names are decoded into the equivalent register address. You will get correct results if you use any one of the words for a given register address, even if the word itself does not make sense for the device you are using.

- **Related Commands:** VXI:SELect, VXI:REGister:WRITe

Example **Read from a Register on the Currently Selected Device**

```
VXI:READ? ICON
```

Read from the Interrupt Control Register of the currently selected device.

:REGister:WRITe

VXI:REGister:WRITe ***<register>, <data>*** writes *data* to the specified 16-bit register at the selected logical address (see VXI:SELect).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<register>	numeric	Even numbers from 0 to 62 or register name (see below).	none
<data>	numeric	-32768 to 32767	none

Comments

- The *register* parameter can be all even numbers from 0 to 62 inclusive (as a numeric value) or the following (optional) words:

  A16 Window: A16 Window Map Register (12)
  A24 Window: A24 Window Map Register (14)
  A32 Window: A32 Window Map Register (16)
  CONTrol: Control Register (4)
  DEXTended: Data Extended Register (10)
  DHIGh: Data High Register (12)
  DLOW: Data Low Register (14)
  ETConfigure: ECL Trigger Configuration Register (22)
  ICNF: Interrupt Configuration Register (18)
  ICONtrol: Interrupt Control Register (28)
  LAWindow: Logical Address Configuration Register (10)
  MODid: MODID Register (8)
  LADDress: Logical Address Register (0)
  OFFSet: Offset Register (6)
  SIGNal: Signal Register (8)
  TTConfigure: TTL Trigger Configuration Register (20)
  UCONfigure: Utility Configuration Register (24)

Note The optional register names are decoded into the equivalent register address. You will get correct results if you use any one of the words for a given register address, even if the word itself does not make sense for the device you are using.

- **Related Commands:** VXI:SELect, VXI:REGister:READ?

Example **Write to a Register on the Currently Selected Device**

```
VXI:REG:WRIT DHIG,64
```

Writes "64" to Data High Register.

:RESet

VXI:RESet <*logical_addr*> performs a soft reset of the device at *logical_addr*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	numeric	Must round to 0 through 255.	none

Comments

- VXI:RESet sets the Sysfail Inhibit bit in the device's Control Register, then sets the Reset bit, waits 100µs, then clears Reset. When the device has passed its self-test, Sysfail Inhibit is cleared. If the device fails during the reset (does not assert "Passes" within 4.9 sec), Sysfail Inhibit remains asserted.
- If the command module is the resource manager, it can reset any device within the mainframe. If the command module is not the resource manager, it can only reset devices within its servant area. You cannot use VXI:RESet to reset the command module (use DIAG:BOOT).
- When a device is reset, the command module (system instrument) will write 1's to the device dependent bits in the device's Control Register.
- This command has been retained for compatibility with existing programs. For new programs you should use VXI:RESet?.

Example

Reset a VXIbus Device

```
VXI:RES 64          Reset device at logical addr 64.
```

:RESet?

VXI:RESet? resets the selected logical address. SYSFAIL generation is inhibited while the device is in the self-test state. The command waits for 5 seconds or until the selected device has indicated passed (whichever occurs first). If the device passes its self-test, the SYSFAIL generation is re-enabled. If the device fails the self-test, then SYSFAIL generation will remain inhibited.

Comments

- The return value from this command is the state of the selected device after it has been reset. The command returns a <NR1> encoded as follows:
  0 = FAIL, **2** = PASS, **3** = READY
- The state of the A24/A32 enable bit is not altered by this command.
- If the command module is the resource manager, it can reset any device within the mainframe. If the command module is not the resource manager, it can only reset devices within its servant area. You cannot use VXI:RESet? to reset the command module (use DIAG:BOOT).
- **Related Commands:** VXI:SELect

:ROUTe:ECLTrg<*n*>

VXI:ROUTe:ECLTrg<*n*> configures the routing of the ECL trigger line specified by <*n*> for all mainframe extenders in the system.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 or 1	none

Comments

- The routing is set so the device selected by the VXI:SELect command can source the trigger line and all other devices in the system may monitor that trigger line.
- Some mainframe extender devices do not support some trigger lines. This command will determine whether the specified trigger line is supported while it attempts to execute the command and return a trigger not supported error if it encounters any extenders that do not support the specified trigger. It will attempt to direct all extenders that do support the specified trigger, even if it encounters some extenders that do not.
- This command can only be executed by the System instrument in a command module that is serving as resource manager for the entire VXIbus system.
- **Related Commands:** VXI:SELect, VXI:ROUTe:TTLTrg<*n*>, VXI:ROUTe:INTerrupt<*n*>, VXI:CONFigure:MEXTender…

:ROUTe:INTerrupt<*n*>

VXI:ROUTe:INTerrupt<*n*> configures the routing of the interrupt line specified by <*n*> for all mainframe extenders in the system.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 through 7	none

Comments

- The routing is set so the device selected by the VXI:SELect command can handle the interrupt line and all other devices in the system may assert that interrupt line.
- Some mainframe extender devices do not support directing interrupt lines. This command will determine whether the specified interrupt line is supported while it attempts to execute the command and return a trigger not supported error if it encounters any extenders that do not support the specified line. It will attempt to direct all extenders that do support the specified line, even if it encounters some extenders that do not.
- This command can only be executed by the System instrument in a command module that is serving as resource manager for the entire VXIbus system.
- **Related Commands:** VXI:SELect, VXI:ROUTe:TTLTrg<*n*>, VXI:ROUTe:ECLTrg<*n*>, VXI:CONFigure:MEXTender

:ROUTe:TTLTrg<*n*>

VXI:ROUTe:TTLTrg<*n*> configures the routing of the TTL trigger line specified by <*n*> for all mainframe extenders in the system.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*n*>	numeric	0 through 7	none

Comments

- The routing is set so the device selected by the VXI:SELect command can source the trigger line and all other devices in the system may monitor that trigger line.
- Some mainframe extender devices do not support some trigger lines. This command will determine whether the specified trigger line is supported while it attempts to execute the command and return a trigger not supported error if it encounters any extenders that do not support the specified trigger. It will attempt to direct all extenders that do support the specified trigger, even if it encounters some extenders that do not.
- This command can only be executed by the System instrument in a command module that is serving as resource manager for the entire VXIbus system.
- **Related Commands:** VXI:SELect, VXI:ROUTe:INTerrupt<*n*>, VXI:ROUTe:ECLTrg<*n*>, VXI:CONFigure:MEXTernal…

:SELect

VXI:SELect <*logical_addr*> specifies the *logical address* to be used by many subsequent commands in the VXI subsystem.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	numeric	Must round to 0 through 255.	none

Comments

- The *RST default value for *logical_addr* is that no logical address is selected (i.e., -1). All other commands which require a logical address to be selected will respond with `Error -221, "Settings conflict"`, if no instruments *logical address* is selected.
- When a command encounters an `Error -240, "Hardware error"`, the equivalent of a *RST is executed. This will cause the selected *logical address* to be set to -1.
- **Related Commands:** VXI:CONFigure:LADDress?

Example

Select a Logical Address

```
VXI:SEL 64
```

Sets the logical address to be used by subsequent VXI subsystem commands to 64.

:SELect?

VXI:SELect? returns the logical address which will be used by many subsequent commands in the VXI subsystem. If no logical address has been selected, this query will return −1.

:SEND:COMMand

VXI:SEND:COMMand <*logical_addr*>,<*command*>[,<*data*>] sends the specified word serial *command* (and optional *data*) to *logical_addr*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	numeric	Must round to 0 through 255.	none

The *command* field and any required *data* fields are specified in the table below.

<*command*>	<*data*>	Description
BAVailable	<*byte*>(0 - 511)	Byte Available (bit 8 = 1 = END, bits 7-0 = data byte)
CLEar		Clear
CLOCk		Clear Lock
GDEVice	<*device_laddr*>(0 - 255)	Grant Device
ICOMmander	<*cmdr_laddr*>(0 - 255)	Identify Commander
SLOCk		Set Lock
TRIGger		Trigger
ANY	<*cmd_word*>	Specify any word serial command as a 16-bit value in *cmd_word*. Read response from the Data Low Register using VXI:QUERy?.

Comments

- <*data*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- VXI:SEND:COMMand uses the Word Serial Transfer Protocol. This protocol is described in the *VXIbus System Specifications*.
- VXI:SEND:COMMand is recommended for use with devices conforming to *VXIbus System Specifications*, revision 1.3 or later.
- This command has been retained for compatibility with existing programs. For new programs you should use VXI:WSP:COMM.
- **Related Commands:** VXI:SEND:COMMands?, VXI:WSProtocol:COMMand, VXI:WSProtocol:QUERy?

Example

Send 1 Data Byte to Logical Address 241

```
VXI:SEND:COMM  241,BAV,452          End bit = 1 and data byte is 196.
```

:SEND:COMMand?

VXI:SEND:COMMand ***<logical_addr>,<command>*[,*<data1>*[,*<data2>*]]** sends the specified word serial *command* (and optional *dataN* values) using the word-serial protocol, to the module at *logical_addr*. It then waits for and returns a 16-bit response value.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<logical_addr>	numeric	Must round to 0 through 255.	none

The *command* field and any required *data* fields are specified in the following table.

<command>	*<data1>*	*<data2>*	Description
AHLine	*<hand_id>* (1 - 7)	*<line_#>* (0 - 7)	Assign Handler Line. A line number of 0 means the handler is to be disconnected.
AILine	*<int_id>* (1 - 7)	*<line_#>* (0 - 7)	Assign Interrupter Line. A line number of 0 means the handler is to be disconnected.
AMControl	*<rspns_mask>* (0 - 15)		Asynchronous Mode Control
ANO			Abort Normal Operation
ANY	*<cmd_word>* (-32768 - 32767)		Specify any VXIbus command
BNO	*<top_level>* (0 | non-zero)		Begin Normal Operation
BREQuest			Byte Request
CEVent	*<enable>* (0 | 1 | OFF | ON)	*<event_#>* (0 - 127)	Control Event
CRESponse	*<rspns_mask>* (0 - 127)		Control Response
ENO			End Normal Operation
RDEVice	*<logical_addr>* (0 - 255)		Release Device
RHANdlers			Read Handlers
RHLine	*<hand_id>* (1 - 7)		Read Handler Line
RILine	*<int_id>* (1 - 7)		Read Interrupter Line
RINTerrupter			Read Interrupters
RMODid			Read MODID
RPERror			Read Protocol Error
RPRotocol			Read Protocol
RSAREa			Read Servant Area
RSTB			Read STB
SLModid	*<enable>* (0 | 1 | OFF | ON)	*<modid>* (0 - 127)	Set Lower MODID (lines 0 - 6)
SUModid	*<enable>* (0 | 1 | OFF | ON)	*<modid>* (0 - 63)	Set Upper MODID (lines 7 - 12)

Comments

- <*data1*> and <*data2*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- VXI:SEND:COMMand uses the Word Serial Transfer Protocol. This protocol is described in the *VXIbus System Specification Manual*.
- VXI:SEND:COMMand? is recommended for use with devices conforming to *VXIbus Specifications*, revision 1.3 or later.
- This command has been retained for compatibility with existing programs. For new programs you should use VXI:WSP:QUER?
- **Related Commands:** VXI:SEND:COMMand, VXI:WSProtocol:QUERy?

Example

Read Which IRQ Line is Used by Interrupt Handler in Logical Address 241

```
VXI:SEND:COMM?  241,RHLINE,2      Which line used by second handler
                                  in servant at 241.
enter statement                   Return the number of the interrupt
                                  line.
```

:SEND[:MESSage]

VXI:SEND[:MESSage] <*logical_addr*>,"<*msg_string*>"[<*end_flag*>] sends the specified message string to the message based module at *logical_addr*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*logical_addr*>	decimal numeric	Must round to 0 through 255.	none
<*msg_string*>	string	ASCII characters (no nulls)	none
<*end_flag*>	discrete	END | NOENd	none

Comments

- VXI:REC? together with VXI:SEND can be used to communicate with message-based devices from an RS-232 monitor via the command module. If the command module is the resource manager, the message-based devices can be inside or outside its servant area. If the command module is not the resource manager, the message-based devices must be in the command module's servant area.
- VXI:SEND uses the Byte Transfer Protocol which uses the DIR and DOR bits in the Response register. This protocol and DIR/DOR are described in the *VXIbus System Specifications*.
- The last byte of *msg_string* is sent with the END bit set unless *end_flag* is specified as NOENd.
- If CR or CRLF is to be sent, they must be included in *msg_string*.
- Null characters (ASCII value 0) must not occur in *msg_string*.
- This command has been retained for compatibility with existing programs. For new programs you should use VXI:WSP:MESS:SEND.
- **Related Commands:** VXI[:RECeive]:MESSage?, VXI:WSProtocol:MESSage:SEND, VXI:WSProtocol:MESSage:RECeive?

Example **Send a Message to a Message-Based Device at Logical Address 16**

VXI:SEND 16,"MEAS:VOLT:DC?"	*Send command to message-based multimeter (last by is sent with END bit set).*
VXI:REC? 16	*Retrieve voltage measurement.*

:WRITe

VXI:WRITe *<logical_addr>*, *<register_addr>*,*<data>* allows access to the entire 64-byte A16 register address space for the device specified by *logical_addr*. Since the VXIbus system is byte-addressed, while the registers are 16-bits wide, registers are specified by even addresses only. This method of identifying registers follows the VXIbus standard format.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<logical_addr>	decimal numeric	Must round to 0 through 255.	none
<register_addr>	numeric	Must round to an even value from 0 through 62 (3E$_h$).	none
<data>	numeric	Must round to -32768 to 32767 (0 to FFFF$_h$).	none

Comments

- Specifying an odd register address will cause `Error +2003, "Invalid word address"`.
- Specifying a logical address not currently in use in the system will cause `Error +2005, "No card at logical address"`.
- If the command module is the resource manager, it can write to any device within the mainframe. If the command module is not the resource manager, it can only write to those devices within its servant area.
- *<logical_addr>* **must** be specified in decimal. *<register_addr>* and *<data>* **may** be specified in decimal, hex (#H), octal (#Q), or binary (#B) format.
- This command has been retained for compatibility with existing programs. For new programs you should use the VXI:REG:WRIT command.
- Accesses are 16-bit non-privileged data accesses.
- **Related Commands:** VXI:READ?, VXI:REGister:WRITe

Example **Write a Value into a Device's Device Dependent Register**

VXI:WRIT 8,24,#H4200	*Write hex 4200 (16,896 decimal) to register 24 of device at logical address 8.*

:WSProtocol :COMMand:*command*

VXI:WSProtocol:COMMand:*command* is a series of commands which sends the specified Word Serial Command to the address set using the VXI:SELect command and continues without waiting for a response. The response to this command can be read with the VXI:WSProtocol:RESPonse? command. The following table lists the available commands and their parameters (if any).

:command	*parameter1*	*parameter2*	Description
:AHLine	<*hand_id*> (1-7)	<*line_#*> (0-7)	Assign Handler Line. A line number of 0 means the handler is to be disconnected.
:AILine	<*int_id*> (1 - 7)	<*line_#*> (0 - 7)	Assign Interrupter Line. A line number of 0 means the handler is to be disconnected.
:AMControl	<*rspns_mask*> (0 - 15)		Asynchronous Mode Control
:ANO			Abort Normal Operation
:ANY	<*cmd_word*> (-32768 - 32767)		Specify any word serial command as a 16-bit value in *cmd_word*.
:BAVailable	<*end_bit*> (1 | 0 | OFF | ON)	<*byte*> (0 - 255)	Byte Available (bit 8 = 1 = END, bits 7 - 0 = data byte)
:BNO	<*top_level*> (1 | 0 | OFF | ON)		Begin Normal Operation
:BREQuest			Byte Request
:CEVent	<*enable*> (0 | 1 | OFF | ON)	<*event_#*> (0 - 127)	Control Event
:CLEar			Clear
:CLOCk			Clear Lock
:CRESponse	<*rspns_mask*> (0 - 127)		Control Response
:ENO			End Normal Operation
:GDEVice	<*cmdr_laddr*> (0 - 255)		Grant Device
:ICOMmander			Identify Commander
:RDEVice	<*logical_addr*> (0 - 255)		Release Device
:RHANdlers			Read Handlers
:RHLine	<*hand_id*> (1 - 7)		Read Handler Line
:RILine	<*int_id*> (1 - 7)		Read Interrupter Line
:RINTerrupter			Read Interrupters
:RMODid			Read MODID
:RPERror			Read Protocol Error
:RPRotocol			Read Protocol
:RSAREa			Read Servant Area
:RSTB			Read STB
:SLModid	<*enable*> (0 | 1 | OFF | ON)	<*modid*> (0 - 127)	Set Lower MODID (lines 0 - 6)
:SLOCk			Set Lock
:SUModid	<*enable*> (0 | 1 | OFF | ON)	<*modid*> (0 - 63)	Set Upper MODID (lines 7 - 12)
:TRIGger			Trigger

Comments

- *byte*, *cmd_word*, *event_number*, *hand_id*, *int_id*, *line_number*, *logical_address*, *modid,* and *response_mask* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- *end_bit* selects whether the END bit is set in the command.
- *top_level* selects whether the Top_level bit is set in the command.
- *enable* selects whether the Enable bit is set in the command.
- **Related Commands:** VXI:SELect, VXI:WSProtocol:RESPonse?, VXI:WSProtocol:QUERy?

:WSProtocol :MESSage:RECeive?

VXI:WSProtocol:MESSage:RECeive? <*count* | *terminator*> receives a message from the selected logical address using both the word serial protocol and the byte transfer protocol.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*terminator*>	numeric	count | LF | CRLF | END	END

Comments

- The command will always terminate on the End bit being set. Additional termination options are on a specified number of bytes (*count*), or on a match to a particular *terminator* (That is, LF, CRLF, END).
- The response is returned as a string.
- **Related Commands:** VXI:SELect, VXI:WSProtocol:MESSage:SEND

:WSProtocol :MESSage:SEND

VXI:WSP:MESS:SEND <*message_string*>[,(END | NEN)] sends the specified *message_string* to the selected logical address. The string is sent using the word serial protocol with the byte transfer protocol.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<*message_string*>	text string	Any valid test string.	none
<*end_bit*>	discrete	END | NEN	END

Comments

- The last byte of the string is sent with the *end_bit* set unless you specify NEN (NoENd).
- **Related Commands:** VXI:SELect, VXI:WSProtocol:MESSage:RECeive?

:WSProtocol:QUERy :*command*?

VXI:WSProtocol:QUERy*:command*? is a series of commands which sends the specified Word Serial Command to the address set using the VXI:SELect command and waits for a response. The returned value is the response to the command and is an integer. The following table lists the available commands and their parameters (if any).

:command	*parameter1*	*parameter2*	Description
:AHLine?	<*hand_id*> (1-7)	<*line_#*> (0-7)	Assign Handler Line. A line number of 0 means the handler is to be disconnected.
:AILine?	<*int_id*> (1 - 7)	<*line_#*> (0 - 7)	Assign Interrupter Line. A line number of 0 means the handler is to be disconnected.
:AMControl?	<*rspns_mask*> (0 - 15)		Asynchronous Mode Control
:ANO?			Abort Normal Operation
:ANY?	<*cmd_word*> (-32768 - 32767)		Specify any VXIbus command
:BNO?	<*top_level*> (0 | non-zero)		Begin Normal Operation
:BREQuest?			Byte Request
:CEVent?	<*enable*> (0 | 1 | OFF | ON)	<*event_#*> (0 - 127)	Control Event
:CRESponse?	<*rspns_mask*> (0 - 127)		Control Response
:ENO?			End Normal Operation
:RDEVice?	<*logical_addr*> (0 - 255)		Release Device
:RHANdlers?			Read Handlers
:RHLine?	<*hand_id*> (1 - 7)		Read Handler Line
:RILine?	<*int_id*> (1 - 7)		Read Interrupter Line
:RINTerrupter?			Read Interrupters
:RMODid?			Read MODID
:RPERror?			Read Protocol Error
:RPRotocol?			Read Protocol
:RSARea?			Read Servant Area
:RSTB?			Read STB
:SLModid?	<*enable*> (0 | 1 | OFF | ON)	<*modid*> (0 - 127)	Set Lower MODID (lines 0 - 6)
:SUModid?	<*enable*> (0 | 1 | OFF | ON)	<*modid*> (0 - 63)	Set Upper MODID (lines 7 - 12)

Comments

- *event_number*, *hand_id*, *int_id*, *line_number*, *modid*, and *response_mask* may be specified in decimal, hex (#H), octal (#Q), or binary (#B) formats.
- *top_level* selects whether the END bit is set in the command.
- *enable* selects whether the Enable bit is set in the command.
- **Related Commands:** VXI:SELect, VXI:WSProtocol:COMMand

:WSProtocol :RESPonse?

VXI:WSProtocol:RESPonse? returns one word of data from the data low register on the selected logical address. This command obeys the byte transfer protocol. The data is returned as an integer.

Common Command Reference

This section describes the IEEE-488.2 common commands that can be used to program instruments in the mainframe. Commands are listed alphabetically (the following table shows the common commands listed by functional group). Examples are shown when the command has parameters or returns a response; otherwise the command string is as shown in the headings in this section. For additional information on any common commands, refer to the *IEEE Standard 488.2-1987*.

IEEE 488.2 Common Command Functional Groupings

Category	Command	Title
General	*IDN?	Identification Query
	*RST	Reset Command
	*TST?	Self-test Query
Instrument Status	*CLS	Clear Status Command
	*ESE <*mask*>	Standard Event Status Enable Command
	*ESE?	Standard Event Status Enable Query
	*ESR?	Standard Event Status Register Query
	*PSC <*flag*>	Power-on Status Clear Command
	*PSC?	Power-on Status Clear Query
	*SRE <*mask*>	Service Request Enable Command
	*SRE?	Service Request Enable Query
	*STB?	Status Byte Query
Macros	*DMC <*name*>,<*cmds*>	Define Macro Command
	*EMC <*state*>	Enable Macros Command
	*EMC?	Enable Macro Query
	*GMC? <*name*>	Get Macro Query
	*LMC?	Learn Macro Query
	*PMC	Purge all Macros Command
	*RMC <*name*>	Remove individual Macro Command
Synchronization	*OPC	Operation Complete Command
	*OPC?	Operation Complete Query
	*WAI	Wait-to-Continue Command

*CLS

Clear Status Command clears all status registers (Standard Event Status Register, Standard Operation Event Status Register, Questionable Data Event Register) and the error queue for an instrument. This clears the corresponding summary bits (bits 3, 5, and 7) and the instrument-specific bits (bits 0, 1, and 2) in the Status Byte Register. *CLS does not affect the enabling of bits in any of the status registers (Status Byte Register, Standard Event Status Register, Standard Operation Event Status Register, or Questionable Data Event Status Register). (The SCPI command STATus:PRESet *does* clear the Standard Operation Status Enable and Questionable Status Enable registers.) *CLS disables the Operation Complete function (*OPC command) and the Operation Complete Query function (*OPC? command).

*DMC <*name_string*>, <*command_block*>

Define Macro Command assigns one, or a sequence of commands to a macro name.

The command sequence may be composed of SCPI and/or Common commands.

The name given to the macro may be the same as a SCPI command, but may not be the same as a common command. When a SCPI named macro is executed, the macro rather than the SCPI command is executed. To regain the function of the SCPI command, execute the *EMC 0 command.

Example **Create a Macro to Return the System Instrument's Device List**

```
OUTPUT 70900;"*DMC 'LIST',#0VXI:CONF:DLIS?"
```

Note that the name LIST is in quotes. The second parameter type is *arbitrary block program data.* The characters that define a command message are prefixed by the characters #0 (pound zero). For a more information on this parameter type, see page 121.

*EMC <*state*>

Enable Macros Command when *enable* is non-zero, macros are enabled. When *enable* is zero, macros are disabled.

*EMC?

Enable Macros Query returns either "1" (macros are enabled), or "0" (macros are disabled) for the selected instrument.

*ESE <*mask*>

Standard Event Status Enable Register Command enables one or more events in the Standard Event Status Register to be reported in bit 5 (the Standard Event Status Summary Bit) of the Status Byte Register. You enable an event by specifying its decimal weight for <*mask*>. To enable more than one event, specify the sum of the decimal weights. Refer to Chapter 4 in this manual for more information on the Standard Event Status Register.

Example

```
OUTPUT 70900;"*ESE 60"
```

Enable Bits 2, 3, 4, and 5. Respective weights are 4 + 8 + 16 + 32 = 60.

*ESE?

Standard Event Status Enable Query returns the weighted sum of all enabled (unmasked) bits in the Standard Event Status Register.

Example

```
10  OUTPUT  70900;"*ESE?"      Send status enable query.
20  ENTER  70900;A             Place response in variable.
30  PRINT  A                   Print response.
40  END
```

*ESR?

Standard Event Status Register Query returns the weighted sum of all set bits in the Standard Event Status Register. After reading the register, *ESR? clears the register. The events recorded in the Standard Event Status Register are independent of whether or not those events are enabled with the *ESE command.

Example

```
10  OUTPUT  70900;"*ESR?"      Send Standard Event Status
                               Register query.
20  ENTER  70900;A             Place response in variable.
30  PRINT  A                   Print response.
40  END
```

*GMC? <*name_string*>

Get Macro Query returns *arbitrary block response data* which contains the command or command sequence defined by *name_string*. The command sequence will be prefixed with characters which indicate the number of characters that follow the prefix.

Example

```
10  OUTPUT  70900;"*GMC?  'LIST'"   Ask for definition of macro from
                                    *DMC example.
20  ENTER  70900;Cmds$              Enter into Cmds$ the definition of
                                    the macro "LIST".
30  PRINT  Cmds$                    Cmds$=#214VXI:CONF:DLIS?
40  END
```

In this case, the prefix consists of "#214". The 2 says to expect two character-counting digits. The 14 says that 14 characters of data follow. Had the returned macro been shorter, such as #15*EMC?, we would read this as 1 counting digit indicating 5 data characters.

*IDN?

Identity returns the device identity. The response consists of the following four fields (fields are separated by commas):

- Manufacturer
- Model Number
- Serial Number (returns 0 if not available)
- Firmware Revision (returns 0 if not available)

The *IDN? command returns the following command string for the HP E1406A System instrument (Flash ROMS Run/Load switch is in the "Run" position):

```
HEWLETT-PACKARD,E1406A,0,A,01.00
```

This command will return the following string for the HP E1406A Loader instrument (Flash ROMS Run/Load switch is in the "Load" position):

```
HEWLETT-PACKARD,LOADER,0,A,01.00
```

Note The revision will vary with the revision of the downloaded operating system installed in the system. This is the only indication of which version of operating system is in the box. The major number (01 in the examples) indicates whether there have been functional changes made in this downloaded operating system. The minor number (00 in the examples) indicates whether only bug fixes and minor changes were made.

Example **Get and Print the ID Fields from the System**

```
10  DIM  A$[50]                  Dimension array for ID fields.
20  OUTPUT  70900;"*IDN?"        Query identity.
30  ENTER  70900;A$              Place ID fields in array.
40  PRINT  A$                    Print ID fields.
50  END
```

*LMC?

Learn Macros Query returns a quoted string *name* for each currently defined macro. If more than one macro is defined, the quoted strings are separated by commas (,). If no macro is defined, then a quoted null string ("") is returned.

*LRN?

Learn Query Command causes the instrument to respond with a string of SCPI commands which define the instrument's current state. Your application program can enter the *LRN? response data into a string variable, later to be sent back to the instrument to restore that configuration.

Example response from an HP E1326B multimeter in the power-on state:

```
*RST;:CAL:ZERO:AUTO 1; :CAL:LFR +60; VAL
+0.00000000E+000; :DISP:MON:STAT 0; CHAN (@0); :FORM
ASC,+7; :FUNC "VOLT"; :MEM:VME:ADDR +2097152; SIZE
+0; STAT 0; :RES:APER +1.666667E-002; OCOM 0; RANG
+1.638400E+004; RANG:AUTO 1;:VOLT:APER
+1.666667E-002; RANG +8.000000E+000; RANG:AUTO 1;
:TRIG:COUN +1; DEL +0.00000000E+000; DEL:AUTO 1;
:TRIG:SOUR IMM; :SAMP:COUN  +1; SOUR IMM;TIM
+5.000000E-002 S
```

Note The System instrument no longer implements the *LRN? command. Attempting to have the System instrument execute this command will generate `Error -113, "Undefined header"`.

*OPC

Operation Complete causes an instrument to set bit 0 (Operation Complete Message) in the Standard Event Status Register when all pending operations have been completed. By enabling this bit to be reflected in the Status Byte Register (*ESE 1 command), you can ensure synchronization between the instrument and an external computer or between multiple instruments.

*OPC?

Operation Complete Query causes an instrument to place an ASCII 1 into the instrument's output queue when all pending instrument operations are finished. By requiring the computer to read this response before continuing program execution, you can ensure synchronization between one or more instruments and the computer.

*PMC

Purge Macros Command purges all currently defined macros in the selected instrument.

*PSC <*flag*>

Power-on Status Clear Command controls the automatic power-on clearing of the Service Request Enable Register and Standard Event Status Enable Register. Executing *PSC 1 disables any previously enabled bits at power-on, preventing the System instrument from requesting service when power is cycled. Executing *PSC 0 causes any previously enabled bits to remain enabled at power-on which allows the System instrument to request service (if it has been enabled - *SRE) when power is cycled. The value of *flag* is stored in non-volatile memory.

Example **This example configures the System instrument to request service from the external computer whenever power is cycled.**

Status Byte Register and Standard Event Status Register bits remain enabled (unmasked) after cycling power.

```
10  OUTPUT 70900;"*PSC 0"
```

Enable bit 5 (Standard Event Status Register Summary bit) in the Status Byte Register.

```
20  OUTPUT 70900;"*SRE 32"
```

Enable bit 7 (Power-on bit) in the Standard Event Status Register to be reflected as bit 5 in the Status Byte Register.

```
30  OUTPUT 70900;"*ESE 128"
```

*PSC?

Power-on Status Clear Query returns a response indicating whether an instrument's Status Byte Register and Standard Event Status Register bits remain enabled or become disabled at power-on. A "1" means the bits are disabled at power-on; a "0" means the bits remain enabled at power-on.

*RMC <*name_string*>

Remove Individual Macro Command purges an individual macro identified by the *name_string* parameter.

Example

```
OUTPUT 70900;"*RMC 'LIST'"        Remove macro command from *DMC example.
```

*RST

Reset Resets an instrument as follows:

- Sets the instrument to a known state (usually the power-on state).
- Aborts all pending operations.
- Disables the *OPC and *OPC? modes.

*RST does not affect:

- The state of the HP-IB interface.
- The HP-IB address.
- The output queue.
- The Service Request Enable Register.
- The Standard Event Status Enable Register.
- The power-on flag.
- Calibration data.
- Protected user data.

*SRE <*mask*>

Service Request Enable When a service request event occurs, it sets a corresponding bit in the Status Byte Register (this happens whether or not the event has been enabled (unmasked) by *SRE). The *SRE command allows you to identify which of these events will assert a service request (SRQ). When an event is enabled by *SRE and that event occurs, it sets a bit in the Status Byte Register and issues an SRQ to the computer (sets the HP-IB SRQ line true). You enable an event by specifying its decimal weight for <*mask*>. To enable more than one event, specify the sum of the decimal weights. Refer to Chapter 4 in this manual for more information on the Status Byte Register.

Example

```
OUTPUT 70900;"*SRE 160"           Enables bits 5 and 7. Respective
                                  weights are 32 + 128 = 160.
```

*SRE?

Status Register Enable Query returns the weighted sum of all enabled (unmasked) events (those enabled to assert SRQ) in the Status Byte Register.

Example

```
10  OUTPUT 70900;"*SRE?"          Send Status Register Enable query.
20  ENTER 70900;A                 Place response in variable.
30  PRINT A                       Print response.
40  END
```

*STB?

Status Byte Register Query returns the weighted sum of all set bits in the Status Byte Register. Refer to Chapter 4 in this manual for more information on the Status Byte Register.

Comments

You can read the Status Byte Register using either the *STB? command or an HP-IB serial poll (IEEE 488.1 message). Both methods return the weighted sum of all set bits in the register. The difference between the two methods is that *STB? does not clear bit 6 (Service Request); serial poll does clear bit 6. No other Status Byte Register bits are cleared by either method with the exception of the Message Available bit (bit 4) which may be cleared as a result of reading the response to *STB?.

Example

```
10  OUTPUT 70900;"*STB?"      Send Status Byte Register query.
20  ENTER 70900;A             Place response in variable.
30  PRINT A                   Print response.
40  END
```

*TST?

Self-Test causes an instrument to execute an internal self-test and returns a response showing the results of the self-test. A 0 (zero) response indicates that self-test passed. A value other than zero indicates a self-test failure or error.

Example

```
10  OUTPUT 70900;"*TST?"      Execute self-test, return response.
20  ENTER 70900;A             Place self-test response in variable.
30  PRINT A                   Print response.
40 END
```

*WAI

Wait-to-continue prevents an instrument from executing another command until the operation caused by the previous command is finished (sequential operation). Since all instruments normally perform sequential operations, executing the *WAI command causes no change to the instrument's operation.

HP-IB Message Reference

This section describes IEEE-488.1 defined messages and their affect on instruments installed in the mainframe. The examples shown are specifically for HP 9000 Series 200/300 computers using BASIC language. Although any IEEE-488 controller can send these messages, the syntax may be different from that shown here.

Device Clear (DCL) or Selected Device Clear (SDC)

DCL clears all instruments in the command module servant area. SDC clears a specific instrument. The purpose of DCL or SDC is to prepare one or more instruments to receive and execute commands (usually *RST).

DCL or SDC do the following to each instrument:

- Clear the input buffer and output queue.
- Reset the command parser.
- Disable any operation that would prevent *RST from being executed.
- Disable the Operation Complete and Operation Complete Query modes.
- DCL or SDC does not affect:
- Any settings or stored data in the instrument (except the Operation Complete and Operation Complete Query modes).
- Front panel operation.
- Any instrument operation in progress (except as stated above).
- The status byte (except for clearing the Message Available bit as a result of clearing the output queue).

Example

```
CLEAR 7          Clear all instruments.
CLEAR 70900      Clear the System instrument.
```

Go To Local (GTL)

Places an instrument in local state.

Comments

- Refer to the Local Lockout message later in this chapter for information on how GTL affects front panel lockout.

Example

Set HP-IB remote enable line false (all instruments go to local). (You must now execute REMOTE 7 to return to remote mode).

```
LOCAL 7
```

Issue HP-IB GTL to System instrument. (The instrument will return to remote mode when it is listen addressed.)

```
LOCAL 70900
```

Group Execute Trigger (GET)

Executing a group execute trigger will trigger an instrument assuming the following conditions are true:

- The instrument's trigger source is set to Bus (TRIG:SOUR BUS command),
- The instrument is in the Wait-for-Trigger state, and;
- The instrument is addressed to listen (can be done by sending any command, the REMOTE 709ss (ss = secondary address) command, or with the LISTEN command).

Comments

- For instruments in the servant area of an HP E1406A Command Module, only one instrument at a time can be programmed to respond to GET. This is because only one instrument can be addressed to listen at any one time. GET has no affect on the System instrument.

Interface Clear (IFC)

Unaddresses all instruments in the servant area of the specified command module and breaks any bus handshaking in progress.

Example

```
ABORT 7
```

Local Lockout (LLO)

When an instrument is in remote mode, Local Lockout prevents an instrument from being operated from the mainframe's front panel.

Comments

- Certain front panel operations such as menu control and display scrolling are still active in Local Lockout mode.
- If the instrument is in the local state when you send LOCAL LOCKOUT, it remains in local. If the instrument is in the remote state when you send LOCAL LOCKOUT, front panel control is disabled immediately for that instrument.
- After executing LOCAL LOCKOUT, you can enable the keyboard by sending the LOCAL 7 command or by cycling power. The LOCAL 709ss (ss = secondary address) command enables the front panel for that instrument but a subsequent remote command disables it. Sending the LOCAL 7 command removes lockout for all instruments and places them in the local state.

Example

```
10  REMOTE 70900          Set the System instrument remote state.
20  LOCAL LOCKOUT 7       Disable front panel control for the System instrument and all other instruments that were in the remote state.
30  END
```

Remote

Sets the HP-IB remote enable line (REN) true which places an instrument in the remote state.

Comments

- The REMOTE 709ss (ss = secondary address) command places the instrument in the remote state. The REMOTE 7 command, does not, by itself, place the instrument in the remote state. After sending the REMOTE 7 command, the instrument will only go into the remote state when it receives its listen address.
- In most cases, you will only need the REMOTE command after using the LOCAL command. REMOTE is independent of any other HP-IB activity and toggles a single bus line called REN. Most controllers set the REN line true when power is applied or when reset.

Example

```
REMOTE 7                    Sets HP-IB REN line true.
REMOTE 70900                Sets REN line true and addresses
                            System instrument.
```

Serial Poll (SPOLL)

The SPOLL command, like the *STB? Common Command, returns the weighted sum of all set bits in an instrument's Status Byte Register (status byte). Refer to Chapter 4 in this manual for more information on the Status Byte Register.

Comments

- The SPOLL command differs from the *STB? command in that SPOLL clears bit 6 (SRQ). Executing *STB? does not clear bit 6.

Example

```
10  P=SPOLL (70900)         Send Serial Poll and place
                            response into P.
20  DISP P                  Display response.
30  END
```

SCPI Commands Quick Reference

The following table summarizes SCPI commands for the HP E1406A Command Module System Instrument and Loader Instrument. The "Mode" column shows the active mode(s) for the command.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
DIAGnostic		
:BOOT		
:COLD	R/L	Restarts System processor, clears stored configurations.
[:WARM]	R/L	Same as cycling power.
:COMMunicate		
:SERial[0]		
[:OWNer]<*owner*>[SYSTem|IBASic|NONE]	R/L	Allocates the built-in serial interface.
[:OWNer]?	R/L	Returns SYST, IBAS, or NONE.
:SERial[*n*]		
:STORe	R/L	Stores serial communication parameters into non-volatile storage.
:DOWNload		
:CHECked		
[:MADDress] <*address*>,<*data*>	R/L	Write data to non-volatile user RAM starting at the specified address using error correction.
:SADDress <*address*>,<*data*>	R/L	Write data to non-volatile user RAM at the specified address using error correction.
[:MADDress] <*address*>,<*data*>	R/L	Write data to non-volatile user RAM starting at the specified address.
:SADDress <*address*>,<*data*>	R/L	Write data to non-volatile user RAM at the specified address.
:DRAM		
:AVAilable?	R/L	Returns the amount of RAM remaining in the DRAM (Driver RAM) segment.
:CREate <*size*>,<*num_drivers*>	R/L	Creates a non-volatile RAM area for loading instrument drivers.
:CREate? [<MIN|MAX>, <MIN|MAX|DEF>]	R/L	Returns the current or allowable size and maximum number of drivers for Driver RAM.
:DRIVer		
:INSTall	L	Makes the drivers downloaded into Flash ROM available (installs them) by creating the driver index table.
:LIST		
[:ALL]?	R/L	Lists all drivers from all driver tables (RAM and ROM) found on the system.
:FROM?	R	Lists all drivers found in the Flash ROM driver table.
:RAM?	R	Lists all drivers found in the RAM driver table.
:ROM?	R/L	Lists all drivers found in the ROM driver table.
:LOAD <*driver_block*>	R/L	Loads the instrument driver contained in the specified driver_block into a previously created DRAM segment or Flash ROM area.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:CHECked <*driver_block*>	R/L	Loads the instrument driver contained in the specified driver_block into a previously created DRAM segment or Flash ROM area using error correction.
:FROM		
:AVAilable?	R/L	Returns the amount of Flash ROM remaining to hold new device drivers.
:CREate <*num_drivers*>	L	Creates a driver area in Flash ROM for the specified number of drivers.
:CREate?	L	Returns the maximum number of drivers a driver segment in Flash ROM was created with.
:SIZE?	R/L	Returns the amount of Flash ROM available for downloading device drivers.
:INTerrupt		
:ACTivate <*mode*>|0|1|OFF|ON	R/L	Enable VXIbus interrupt acknowledgment.
:PRIority[*n*] <*level*>|MIN|MAX|DEF	R/L	Specifies the priority level of VXI interrupt line [*n*].
:PRIority[*n*]?	R/L	Returns priority level of VXI interrupt line [*n*].
:RESPonse?	R/L	Returns response from the highest priority interrupt line.
:SETup[*n*] <*mode*>|0|1|OFF|ON	R/L	Enables or disables System Instrument control of VXI interrupt line [*n*].
:SETup[*n*]?	R/L	Returns current state of SETup[*n*].
:NRAM		
:ADDRess?	R/L	Returns starting address of the user non-volatile RAM.
:CREate <*size*>|MIN|MAX	R/L	Creates a user non-volatile RAM segment.
:CREate? [MIN|MAX]	R/L	Returns the current or allowable size of user non-volatile RAM.
:PEEK? <*address*>,<*width*>	R/L	Returns an 8, 16, or 32 bit value from memory.
:POKE <*address*>,<*width*>,<*data*>	R/L	Stores an 8, 16, or 32 bit value to RAM.
:RDISk		
:ADDRess?	R/L	Returns the starting address of an IBASIC RAM volume.
:CREate <*size*>|MIN|MAX	R/L	Allocates RAM for an IBASIC RAM volume.
:CREate? [MIN|MAX]	R/L	Returns the current or allowable size of the RAM volume.
:UPLoad		
[:MADDress]? <*address*>,<*byte_count*>	R/L	Returns data from non-volatile user RAM starting at address.
:SADDress? <*address*>,<*byte_count*>	R/L	Returns data from non-volatile user RAM at address.
OUTPut		
:ECLTrg<*line*> (:ECLTrg0 or :ECLTrg1)		
:IMMediate	R	Generate pulse on specified ECL trigger line.
:LEVel		
[:IMMediate] <*level*>|0|1|OFF|ON	R	Sets the output level of the specified ECL trigger line.
[:IMMediate]?	R	Returns the output level of the specified ECL trigger line.
:SOURce <*source*>|INT|EXT|NONE	R	Set the source which drives the selected ECL trigger line.
:SOURce?	R	Returns the source driving the selected ECL trigger line.
[:STATe] <*mode*>|0|1|OFF|ON	R	Enables configuration of the specified ECL trigger line.
[:STATe]?	R	Returns the current state of the selected trigger line.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:EXTernal		
:IMMediate	R	Generate pulse on command module "Trig Out" port.
:LEVel		
[:IMMediate] <*level*>|0|1|OFF|ON	R	Sets the output level of the "Trig Out" port.
[:IMMediate]?	R	Returns the output level of the "Trig Out" port.
:SOURce <*source*>|INT|TTLT<*n*>|ECLT<*n*>|NONE	R	Sets the source which drives the "Trig Out" port.
:SOURce?	R	Returns the source driving the "Trig Out" port.
[:STATe] <*mode*>|0|1|OFF|ON	R	Enables configuration of the "Trig Out" port.
[:STATe]?	R	Returns the state of the "Trig Out" port.
:TTLTrg<*line*> (:TTLTrg0 through :TTLTrg7)		
:IMMediate	R	Generate pulse on the selected TTLT trigger line.
:LEVel		
[:IMMediate] <*level*>|0|1|OFF|ON	R	Sets the output level of the selected TTLT trigger line.
[:IMMediate]?	R	Returns the output level of the selected TTLT trigger line.
:SOURce <*source*>|INT|EXT|NONE	R	Sets the source driving the selected TTLT trigger line.
:SOURce?	R	Returns the source driving the selected TTLT trigger line.
[:STATe] <*mode*>|0|1|OFF|ON	R	Enables configuration of the selected TTLT trigger line.
[:STATe]?	R	Returns the state of the selected TTLT trigger line.
PROGram		
[:SELected]		
:DEFine <*op_sys*>	L	Writes an operating system into Flash ROM.
:CHECked <*op_sys*>	L	Writes an operating system into Flash ROM over an RS-232 line.
:CHECked?	R/L	Returns the operating system in Flash ROM as a definite length arbitrary block formatted for sending over RS-232.
:DEFine?	R/L	Returns the operating system in Flash ROM as a definite length arbitrary block.
:DELete	L	Erases the entire contents of the Flash ROMS.
STATus		
:OPERation		
:CONDition?	R/L	Returns the state of the Condition Register.
:ENABle <*event*>	R/L	Set Standard Operation Enable Register mask.
:ENABle?	R/L	Returns value of enable mask.
[:EVENt]?	R/L	Returns value of the bit set in the Event Register (Standard Operation Status Group).
:NTRansition <*unmask*>	R/L	Sets the negative transition mask.
:PTRansition <*unmask*>	R/L	Sets the positive transition mask.
:PRESet	R/L	Presets Status Registers.
:QUEStionable	R/L	Always returns +0.
:CONDition?	R/L	Returns the state of the Condition Register in the Questionable Status Group.
:ENABle <*event*>	R/L	Set enable mask in Questionable Status Group.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:ENABle?	R/L	Returns value of enable mask in Questionable Status Group.
[:EVENt]?	R/L	Returns value of the bit set in the Event Register (Questionable Status Group).
:NTRansition <*unmask*>	R/L	Sets the negative transition mask.
:PTRansition <*unmask*>	R/L	Sets the positive transition mask.
SYSTem		
:COMMunicate		
:GPIB		
:ADDRess?	R/L	Returns command module primary HP-IB address.
:SERial[*n*]		
:CONTrol		
:DTR <*dtr_cntrl*>|ON|OFF|STAN|IBF	R/L	Sets mode for Data Terminal Ready control line.
:DTR?	R/L	Returns current mode of DTR line.
:RTS <*rts_cntrl*>|ON|OFF|STAN|IBF	R/L	Sets mode for Request To Send control line.
:RTS?	R/L	Returns current mode of RTS line.
[:RECeive]		
:BAUD <*baud_rate*>|MIN|MAX	R/L	Sets transmit and receive baud rate of serial interface.
:BAUD? [MIN|MAX]	R/L	Returns the current or allowable baud rate setting.
:BITS <*bits*>|7|8|MIN|MAX	R/L	Sets the number of data bits in the serial data frame.
:BITS? [MIN|MAX]	R/L	Returns the current or allowable BITS setting.
:PACE		
[:PROTocol] <*protocol*>|XON|NONE	R/L	Sets the receive pacing protocol to XON/XOFF or NONE.
[:PROTocol]?	R/L	Returns the state of receive pacing protocol.
:THReshold		
:STARt <*char_count*>	R/L	Sets the input buffer start threshold for input pacing.
:STARt? [MIN|MAX]	R/L	Returns current or allowable STARt threshold level.
:STOP <*char_count*>	R/L	Sets the input buffer stop threshold for input pacing.
:STOP? [MIN|MAX]	R/L	Returns the current or allowable STOP threshold level.
:PARity		
<*type*>|EVEN|ODD|ZERO|ONE|NONE	R/L	Sets the type of receive and transmit parity.
<*type*>?	R/L	Returns the current parity type setting.
:CHECk <*check_cntrl*>|0|1|OFF|ON	R/L	Enables/disables receive parity checking.
:CHECk?	R/L	Returns the current state of receive parity checking.
:SBITs <*sbits*>|1|2|MIN|MAX	R/L	Sets the number of stop bits for receive and transmit.
:SBITs? [MIN|MAX]	R/L	Returns the number of stop bits set.
:TRANsmit		
:AUTO <*auto_cntrl*>|0|1|OFF|ON	R/L	Links/unlinks the transmit and receive pacing protocol. Note: HP E1324A is always …TRAN:AUTO ON
:AUTO?	R/L	Returns the current transmit/receive pacing linkage.
:PACE		
[:PROTocol] <*protocol*>|XON|NONE	R/L	Sets the transmit pacing protocol to XON/XOFF or NONE.
[:PROTocol]?	R/L	Returns the state of transmit pacing protocol.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:DATE <*year*>,<*month*>,<*day*>	R/L	Sets system calendar.
:DATE? [MIN|MAX,MIN|MAX,MIN|MAX]	R/L	Returns current date or MIN|MAX allowable values.
:ERRor?	R/L	Returns oldest error message in Error Queue.
:TIME <*hour*>,<*minute*>,<*second*>	R/L	Sets the system clock.
:TIME? [MIN|MAX,MIN|MAX,MIN|MAX]	R/L	Returns current time or MIN|MAX allowable values.
:VERSion?	R/L	Returns SCPI version for which this instrument complies.
VXI		
:CONFigure		
:CTABle <*address*>	R	Links the commander/servant hierarchy table to the command module (resource manager) processor.
:CTABle?	R	Gets the commander/servant hierarchy table starting address.
:DCTable <*address*>	R	Links the dynamic configuration table to the command module (resource manager) processor.
:DCTable?	R	Gets the dynamic configuration table starting address.
:DLADdress?	R	Returns a list of the logical addresses in the system.
:DLISt?[<*logical_addr*>]	R	Returns information about one or all installed devices.
:DNUMber?	R	Returns the number of installed devices.
:ETABle <*address*>	R	Links the extender device table to the command module (resource manager) processor.
:ETABle?	R	Gets the extender device table starting address.
:HIERarchy?	R	Gets the current hierarchy configuration data for the selected logical address (see VXI:SELect).
:ALL?	R	Gets the current hierarchy configuration data for all logical addresses.
:INFormation?	R	Gets the static information about the selected logical address (see VXI:SELect).
:ALL?	R	Gets the static information about all logical addresses.
:ITABle <*address*>	R	Links the interrupt line allocation table to the command module (resource manager) processor.
:ITABle?	R	Gets the interrupt line allocation table starting address.
:LADDress?	R	Gets a list of all logical addresses of devices in the system when issued to a resource manager. Generates an error if received by a device other than the resource manager.
:MEXTender?	R	Gets list of all logical addresses for mainframe extenders in the system when issued to a resource manager. Generates an error if received by a device other than the resource manager.
:MEXTender		
:ECLTrg<*line*> <*direction*>|IN|OUT|NONE	R	Configures the selected mainframe extender to direct the ECL trigger specified by <*line*>.
:INTerrupt<*line*> <*direction*>|IN|OUT|NONE	R	Configures the selected mainframe extender to direct the interrupt line specified by <*line*>.
:TTLTrg<*line*> <*direction*>|IN|OUT|NONE	R	Configures the selected mainframe extender to direct the TTL trigger specified by <*line*>.
:MTABle <*address*>	R	Link A24/A32 Address Allocation table to command module (resource manager) processor.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:MTABle?	R	Query A24/A32 address allocation table starting address.
:NUMber?	R	Gets the number of devices in the system when issued to a resource manager. Generates an error if received by a device other than the resource manager.
:MEXTender?	R	Gets the number of mainframe extenders in the system when issued to a resource manager. Generates an error if received by a device other than the resource manager.
:QUERy? <*logical_addr*>	R	Read Data Low Register of device at logical_addr.
:READ? <*logical_addr*>,<*register_addr*>	R	Read the contents of the device register at register_addr.
:RECeive		
[:MESSage]? <*logical_addr*>[,<*end_of_msg*>]	R	Receive message from message-based device at logical_addr.
:REGister		
:READ? <*register*>	R	Returns the contents of the specified 16-bit register at the selected logical address (see VXI:SELect).
:WRITe <*register*>,<*data*>	R	Writes data to the specified 16-bit register at the selected logical address (see VXI:SELect).
:RESet <*logical_addr*>	R	Resets the device at the specified logical address.
:RESet?	R	Resets the device at the selected logical address (see VXI:SELect).
:ROUTe		
:ECLTrg<*line*>	R	Sets the routing of the specified trigger line in all mainframe extenders so that the device selected by VXI:SEL can source the trigger and all other devices in the system can monitor it.
:INTerrupt<*line*>	R	Sets the routing of the specified interrupt line in all mainframe extenders so that the device selected by VXI:SEL can handle the interrupts and all other devices in the system can monitor it.
:TTLTrg<*line*>	R	Sets the routing of the specified trigger line in all mainframe extenders so that the device selected by VXI:SEL can source the trigger and all other devices in the system can monitor it.
:SELect <*logical_addr*>	R	Specifies the logical address to be used by subsequent commands in the VXI subsystem.
:SELect?	R	Returns the logical address to be used by subsequent commands in the VXI subsystem.
:SEND		
:COMMand <*logical_addr*>,<*command*>[,<*data*>]	R	Send word serial command (and optional data) to device at logical_addr.
:COMMand? <*logical_addr*>,<*command*>[,<*data1*>[,<*data2*>]]	R	Send word serial command to device at logical_addr and then wait for response from Data Low Register.
[:MESSage] <*logical_addr*>,<"*msg_string*">[,<*end_flag*>]	R	Send specified message string to the message-based device at logical_addr.
:WRITe <*logical_addr*>,<*register_addr*>,<*data*>	R	Write data to the device register at logical_addr.
:WSProtocol		
:COMMand		
:AHLine <*hand_id*>, <*line_number*>	R	Assigns a handler to the logical address set using VXI:SEL. A line number of 0 means the handler is to be disconnected.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:AILine <*int_id*>, <*line_number*>	R	Assigns an interrupter line to the logical address set using VXI:SEL. A line number of 0 means the handler is to be disconnected.
:AMControl <*response_mask*>	R	Sends an Asynchronous Mode Control command to the logical address set using VXI:SEL.
:ANO	R	Sends an Abort Normal Operation command to the logical address set using VXI:SEL.
:ANY <*cmd_word*>	R	Sends cmd_word as a word serial command to the logical address set using VXI:SEL..
:BAVailable <*end_bit*>	R	Sends a Byte Available command to the logical address set using VXI:SEL.
:BNO <*top_level*>	R	Sends a Begin Normal Operation command to the logical address set using VXI:SEL.
:BREQuest	R	Sends a Byte Request command to the logical address set using VXI:SEL.
:CEVent <*enable*>, <*event_number*>	R	Sends a Control Event command to the logical address set using VXI:SEL.
:CLEar	R	Sends a Clear command to the logical address set using VXI:SEL.
:CLOCk	R	Sends a Clear Lock command to the logical address set using VXI:SEL.
:CRESponse <*response_mask*>	R	Sends a Control Response command to the logical address set using VXI:SEL.
:ENO	R	Sends an End Normal Operation command to the logical address set using VXI:SEL.
:GDEVice <*cmdr_laddr*>	R	Sends a Grant Device command to the logical address set using VXI:SEL.
:ICOMmander	R	Sends an Identify Commander command to the logical address set using VXI:SEL.
:RDEVice <*logical_addr*>	R	Sends a Release Device command to the logical address set using VXI:SEL.
:RHANdlers	R	Sends a Read Handlers command to the logical address set using VXI:SEL.
:RHLine <*hand_id*>	R	Sends a Read Handler Line command to the logical address set using VXI:SEL.
:RILine <*int_id*>	R	Sends a Read Interrupter Line command to the logical address set using VXI:SEL.
:RINTerrupter	R	Sends a Read Interrupter command to the logical address set using VXI:SEL.
:RMODid	R	Sends a Read MODID command to the logical address set using VXI:SEL.
:RPERror	R	Sends a Read Protocol Error command to the logical address set using VXI:SEL.
:RPRotocol	R	Sends a Read Protocol command to the logical address set using VXI:SEL.
:RSAREa	R	Sends a Read Servant Area command to the logical address set using VXI:SEL.
:RSTB	R	Sends a Read Status Byte command to the logical address set using VXI:SEL.
:SLModid <*enable*>, <*modid*> (0-127)	R	Sends a Set Lower MODID command to the logical address set using VXI:SEL.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:SLOCk	R	Sends the Set Lock command to the logical address set using VXI:SEL.
:SUModid <*enable*>, <*modid*> (0-63)	R	Sends a Set Upper MODID command to the logical address set using VXI:SEL.
:TRIGger	R	Sends a Trigger command to the logical address set using VXI:SEL.
:MESSage		
:RECeive? <*count* | *terminator*>	R	Receives a message from the logical address set using VXI:SEL using both the word serial protocol and the byte transfer protocol.
:SEND <*message_string*>[,(END|NEN)]	R	Sends a message to the logical address set using VXI:SEL. The message is sent using both the word serial protocol and the byte transfer protocol.
:QUERy		
:AHLine? <*hand_id*>, <*line_number*>	R	Assigns a handler to the logical address set using VXI:SEL and waits for a response. A line number of 0 means the handler is to be disconnected.
:AILine? <*int_id*>, <*line_number*>	R	Assigns an interrupter line to the logical address set using VXI:SEL and waits for a response. A line number of 0 means the handler is to be disconnected.
:AMControl? <*response_mask*>	R	Sends an Asynchronous Mode Control command to the logical address set using VXI:SEL and waits for a response.
:ANO?	R	Sends an Abort Normal Operation command to the logical address set using VXI:SEL and waits for a response.
:ANY? <*cmd_word*>	R	Sends cmd_word as a word serial command to the logical address set using VXI:SEL and waits for return value.
:BNO? <*top_level*>	R	Sends a Begin Normal Operation command to the logical address set using VXI:SEL and waits for a response.
:BREQuest?	R	Sends a Byte Request command to the logical address set using VXI:SEL and waits for a response.
:CEVent? <*enable*>, <*event_number*>	R	Sends a Control Event command to the logical address set using VXI:SEL and waits for a response.
:CRESponse? <*response_mask*>	R	Sends a Control Response command to the logical address set using VXI:SEL and waits for a response.
:ENO?	R	Sends an End Normal Operation command to the logical address set using VXI:SEL and waits for a response.
:RDEVice? <*logical_addr*>	R	Sends a Release Device command to the logical address set using VXI:SEL and waits for a response.
:RHANdlers?	R	Sends a Read Handlers command to the logical address set using VXI:SEL and waits for a response.
:RHLine? <*hand_id*>	R	Sends a Read Handler Line command to the logical address set using VXI:SEL and waits for a response.
:RILine? <*int_id*>	R	Sends a Read Interrupter Line command to the logical address set using VXI:SEL and waits for a response.
:RINTerrupter?	R	Sends a Read Interrupter command to the logical address set using VXI:SEL and waits for a response.
:RMODid?	R	Sends a Read MODID command to the logical address set using VXI:SEL and waits for a response.

SCPI Commands Quick Reference		
Command	**Mode**	**Description**
Mode: R = active in RUN mode L = active in LOAD mode		
:RPERror?	R	Sends a Read Protocol Error command to the logical address set using VXI:SEL and waits for a response.
:RPRotocol?	R	Sends a Read Protocol command to the logical address set using VXI:SEL and waits for a response.
:RSAArea?	R	Sends a Read Servant Area command to the logical address set using VXI:SEL and waits for a response.
:RSTB?	R	Sends a Read Status Byte command to the logical address set using VXI:SEL and waits for a response.
:SLModid? <*enable*>, <*modid*> (0-127)	R	Sends a Set Lower MODID command to the logical address set using VXI:SEL and waits for a response.
:SUModid? <*enable*>, <*modid*> (0-63)	R	Sends a Set Upper MODID command to the logical address set using VXI:SEL and waits for a response.
:RESPonse?	R	Retrieves the response (one word of integer data) resulting from a WSProtocol:COMMand command.

Common Commands Quick Reference

The following table summarizes IEEE 488.2 common (*) commands for the HP E1406A Command Module. All common commands are available in RUN mode and LOAD mode.

IEEE 488.2 Common Commands Quick Reference		
Category	**Command**	**Title**
All IEEE 488.2 Common Commands are available in RUN mode and LOAD mode.		
General	*IDN?	Identification Query
	*RST	Reset Command
	*TST?	Self Test Query
Instrument Status	*CLS	Clear Status Command
	*ESE <*mask*>	Standard Event Status Enable Register Command
	*ESE?	Standard Event Status Enable Query
	*ESR?	Standard Event Status Register Query
	*PSC <*flag*>	Power-on Status Clear Command
	*PSC?	Power-on Status Clear Query
	*SRE <*mask*>	Service Request Enable Command
	*SRE?	Service Request Enable Query
	*STB?	Status Byte Register Query
Macros	*DMC <*name*>,<*cmds*>	Define Macro Command
	*EMC <*state*>	Enable Macro Command
	*EMC?	Enable Macro Query
	*GMC? <*name*>	Get Macro Query
	*LMC?	Learn Macro Query
	*PMC	Purge all Macros Command
	*RMC <*name*>	Remove individual Macro Command
Synchronization	*OPC	Operation Complete Command
	*OPC?	Operation Complete Query
	*WAI	Wait-to-Continue Command

Appendix A
HP E1406A Specifications and General Information

Device Type

This module returns 014_{16} as the device type in response to a VXI:CONF:DLIS? query if the HP E1406A is set up as a slot zero device and 114_{16} if the HP E1406A is set up as a non-slot zero device.

Real Time Clock

Accuracy: 0.005% of elapsed time since last set.

Temperature coefficient: 0.001% to 0.012% of time since last set (per °C change in temperature).

Resolution: 1.0 sec

Non-volatile lifetime: 10 months minimum for a module with 512 Kbyte memory (following a 15 hour battery charge). 5 months for a module with 1 Mbyte of memory. 2.5 months for a module with 2 Mbyte of memory.

CLK10

Input: TTL or low level AC
Minimum input level: 40 mVp-p
Maximum input level: 42.5 Vp-p
Output: TTL
Jitter: 0.03% (-55 dB)
Initial Accuracy: 50 ppm
Maximum Stability: ±20 ppm/year (0°–55°C)
Typical Stability: ±3 ppm/year at 25°C)

Trigger Input

Levels: TTL
Input load: 5 kΩ, 50 pF
Maximum Rate: 12.5 MHz (TTL), 40 MHz (ECL)
Minimum pulse width: 30 ns (TTL), 12.5 ns (ECL)
Maximum trigger delay: 30 ns

Memory

256 Kbyte user accessible volatile RAM on a module with 512 Kbyte of non-volatile memory. Memory is expandable to 2 Mbyte. NiCad battery backed (10 month minimum lifetime for modules with 512 Kbyte of non-volatile RAM, 5 months for modules with 1 Mbyte of non-volatile RAM, and 2.5 months for modules with 2 Mbyte of non-volatile RAM following a 15 hour battery charge).

Power Requirements

DC Volts	DC Current	Dynamic Current
+5	3.2A	0.32A
+12V	0.01A	0.01A
-12V	0.01A	0.01A
-5.2V	0.4A	0.04A
-2V	0.01A	0.01A
+24V	0.03A	0.003A

Cooling Requirements

For 10 °C rise 1.5 liters/second 0.4mm H_2O

SCPI Conformance Information

The HP E1406A conforms to SCPI-1994.0. The following tables list all the SCPI confirmed and non-SCPI commands that the HP E1406A can execute. Individual commands may not execute without having the proper plug-in module installed in the mainframe. Each plug-in module manual describes the commands that apply to that module.

Switchbox Configuration

The following plug-in modules can be configured as switchbox modules. Refer to the individual plug-in User's Manual for configuration information.

HP E1345A	HP E1353A	HP E1366A
HP E1346A	HP E1357A	HP E1367A
HP E1347A	HP E1358A	HP E1368A
HP E1351A	HP E1361A	HP E1369A
HP E1352A	HP E1364A	HP E1370A

Table A-1. Switchbox SCPI-1994.0 Confirmed Commands

```
ABORt
ARM
    :COUNt
INITiate
    :CONTinuous
    [:IMMediate]
OUTPut
    :ECLTrg<n>
        [:STATe]
    :TTLTrg<n>
        [:STATe]
[ROUTe:]
    CLOSe
    CLOSe?
    OPEN
    OPEN?
    SCAN
STATus
    :OPERation
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
    :PRESet
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
SYSTem
    :CPON
    :CTYPe?
    :ERRor?
    :VERSion?
TRIGger
    [:IMMediate]
    :SOURce
    :SLOPe
```

Table A-2. Switchbox Non-SCPI Commands

```
DISPlay
    :MONitor
        :CARD
        [:STATe]
SYSTem
    :CDEScription?
[ROUTe:]
    SCAN
        [:LIST]
        :MODE
        :PORT
    :SETTling
        [:TIME]
        :TIME?
```

Multimeter Commands

The following tables apply to the HP E1326A/B multimeters.

Table A-3. Multimeter SCPI-1994.0 Confirmed Commands

```
ABORt

CALibration
    :VALue
    :ZERO
        :AUTO
        :AUTO?

CONFigure
    :FRESistance
    :RESistance
    :TEMPerature
    :VOLTage
        :AC
        [:DC]

CONFigure?

FETCh?

FORMat
    [:DATA]

INITiate
    [:IMMediate]

MEASure
    :FRESistance?
    :RESistance?
    :TEMPerature?
    :VOLTage
        :AC?
        [:DC]?

READ?

[SENSe:]
    FUNCtion
    FUNCtion?
    RESistance
        :APERture
        :APERture?
        :NPLCycles
        :NPLCycles?
        :RANGe
            :AUTO
            :AUTO?
        :RANGe?
        :RESolution
        :RESolution?
    VOLTage
        :AC
            :RANGe
            :RANGe?
        [:DC]
            :RANGe
                :AUTO
                :AUTO?
            :RANGe?
            :RESolution
            :RESolution?
        :NPLCycles
        :NPLCycles?

STATus
    :OPERation
        CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
    :PREset
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?

SYSTem
    :CTYPe?
    :ERRor?
    :VERsion?

TRIGger
    :COUNt
    :COUNt?
    :DELay?
        :AUTO
        :AUTO?
    :DELay?
    [:IMMediate]
    :SOURce
    :SOURce?
```

Table A-4. Multimeter Non-SCPI Commands

```
CALibration
    :LFRequency
    :LFRequency?
    :STRain

CONFigure
    :STRain
        :FBENding
        :FBPoisson
        :FPOisson
        :HBENding
        :HPOisson
        :QCOMpression
        :QTENsion
        :QUARter
        :UNSTrained

DISPlay
    :MONitor
        :CHANnel
        :CHANnel?
        [:STATe]
        [:STATe]?

MEASure
    :STRain
        :FBENding?
        :FBPoisson?
        :FPOisson?
        :HBENding?
        :HPOisson?
        :QCOMpression?
        :QTENsion?
        :QUARter?
        :UNSTrained?

MEMory
    :VME
        :ADDRess
        :ADDRess?
        :SIZE
        :SIZE?
        [:STATe]
        [:STATe]?

[ROUTe:]
    FUNCtion

SAMPle
    :COUNt
    :COUNt?
    :SOURce
    :SOURce?
    :TIMer
    :TIMer?

[SENSe:]
    RESistance
        :OCOMpensated
        :OCOmpensated?
    STRain
        :GFACtor
        :POISson
        :UNSTrained

SYSTem
    :CDEScription
```

Counter Commands

The following tables apply to the HP E1332A 4-Channel Counter/Totalizer and the HP E1333A 3-Channel Universal Counter.

Table A-5. HP E1332A SCPI-1994.0 Confirmed Commands

```
ABORt

CONFigure
    :FREQuency
    :PERiod
    :PWIDth
    :NWIDth

CONFigure?

FETCh?

FORMat
    [:DATA]

INITiate
    [:IMMediate]

INPut
    :FILTer
        [:LPASs]
            [:STATe]
            [:STATe]?
            :FREQuency
            :FREQuency?

MEASure
    :FREQuency?
    :PERiod?
    :PWIDth?
    :NWIDth

READ?

[SENSe:]
    FREQuency
        :APERture
        :APERture?
    FUNCtion
        :FREQuency
        :PERiod

STATus
    :OPERation
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
    :PREset
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?

SYSTEM
    :ERRor?
    :VERSion?

TRIGger
    [:IMMediate]
    :SOURCe
    :SOURCe?
```

Table A-6. HP E1332A Non-SCPI Commands

```
CONF[<channel>]
    :TOTalize
    :TINTerval
    :UDCount

DISPlay
    :MONitor
        :CHANnel
        :CHANnel?
        [:STATe]
        [:STATe]?

INPut
    :ISOLate
    :ISOLate?

MEASure[<channel>]
    :TINTerval?

[SENSe[<channel>:]]
    EVENt
        :LEVel
        :LEVel?
        :SLOPe
        :SLOPe?
    PERiod
        :NPERiods
        :NPERiods?
    TOTalize
        :GATE
            :POLarity
            :POLarity?
            [:STATe]
            [:STATe]?
```

Table A-7. HP E1333A SCPI-1994.0 Confirmed Commands

```
ABORt

FETCh?

CONFigure
    :FREQuency
    :NWIDth
    :PERiod
    :PWIDth

CONFigure?

FORMat
    [:DATA]

INITiate
    [:IMMediate]

INPut
    :ATTenuation
    :ATTenuation?
    :COUPling
    :COUPling?
    :FILTer
        [:LPASs]
            [:STATe]
            [:STATe]?
    :IMPedance
    :IMPedance?

MEASure
    :FREQuency?
    :NWIDth?
    :PERiod?
    :PWIDth?

READ?

[SENSe:]
    FUNCtion
        :FREQuency
        :PERiod
    FREQuency
        :APERture
        :APERture?

STATus
    :OPERation
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
    :PREset
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        :[EVENt]?

SYSTem
    :ERRor?
    :VERSion?

TRIGger
    [:IMMediate]
    :SOURCe
    :SOURCe?
```

Table A-8. HP E1333A Non-SCPI Commands

```
CONF[<channel>]
    :RATio
    :TOTalize
    :TINTerval

DISPlay
    :MONitor
        :CHANnel
        :CHANnel?
        [:STATe]
        [:STATe]?

MEASure[<channel>]
    :RATio?
    :TINTerval?

[SENSe[<channel>:]]
    EVENt
        :LEVel
        :LEVel?
        :SLOPe
        :SLOPe?
    PERiod
        :NPERiods
        :NPERiods?
    RATio
        :NPERiods
        :NPERiods?
    TINTerval
        :NPERiods
        :NPERiods?
```

D/A Converter Commands

The following tables apply to the HP E1328A 4-Channel D/A Converter.

Table A-9. HP E1328A SCPI-1994.0 Confirmed Commands

```
CALibration
    [:STATe]
    [:STATe]?

SYSTem
    :ERRor?
    :VERSion?

STATus
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
    :OPERation
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
```

Table A-10. HP E1328A Non-SCPI Commands

```
CALibration
    :CURRent
    :VOLTage

DISPlay
    :MONitor
        :CHANnel
        :CHANnel?
        [:STATe]
        :STRing?

SOURce
    :CURRent <channel>
    :CURRent <channel>?
    :FUNCtion <channel>?
    :VOLTage <channel>
    :VOLTage <channel>?
```

Digital I/O Commands

The following tables apply to the HP E1330A/B Quad 8-bit Digital I/O Module.

Table A-11. HP E1330A/B SCPI-1994.0 Confirmed Commands

```
STATus
    :OPERation
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
    :PREset
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?

SYSTem
    :ERRor?
    :VERSion?
```

Table A-12. HP E1330A/B Non-SCPI Commands

```
DISPlay
    :MONitor
        :PORT
        :PORT?
        [:STATe]
        :STRing?

MEASure
    :DIGital
        :DATA <port>?
            :BIT <number>?
            :BLOCk?
        :FLAG <port>?

MEMory
    :DELete
        MACRo
    :VME
        :ADDRess
        :ADDRess?
        :SIZE
        :SIZE?
        [:STATe]
        [:STATe]?

[SOURce:]
    DIGital
        :CONTrol <port>
            :POLarity
            :POLarity?
            [:VALue]
        :DATA <port>
            :BIT <number>
            :TRACe
            :HANDshake
                :DELay
                [:MODE]
                [:MODE]?
            :POLarity
            :POLarity?
            [:VALue]
        :FLAG <port>
            :POLarity
            :POLarity?
        :HANDshake <port>
            :DELay
            [:MODE]
            [:MODE]?
        :TRACe
            :CATalog
            [:DATA]
            [:DATA]?
            :DEFine
            :DELete
```

Table A-13. System Instrument SCPI-1994.0 Confirmed Commands

```
OUTPut
    :ECLTrg<n>
        :IMMediate
        :LEVel
            [:IMMediate]
            [:IMMediate]?
        :SOURce
        :SOURce?
        [:STATe]
        [:STATe]?
    :TTLTrg<n>
        :IMMediate
        :LEVel
            [:IMMediate]
            [:IMMediate]?
        :SOURce
        :SOURce?
        [:STATe]
        [:STATe]?

PROGram
    [:SELected]
        :DEFine
        :DEFine?
        :DELete
            :ALL
            [:SELected]

STATus
    :OPERation
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
        :NTRansition
        :PTRansition
    :PREset
    :QUEStionable
        :CONDition?
        :ENABle
        :ENABle?
        [:EVENt]?
        :NTRansition
        :PTRansition

SYSTem
    :COMMunicate
        :GPIB
            :ADDRess?
        :SERial
            :CONTrol
                :DTR
                :DTR?
                :RTS
                :RTS?
            [:RECeive]
                :BAUD
                :BAUD?
                :BITS
                :BITS?
                :PACE
                :PARity
                :SBITs
                :SBITs?
            :TRANsmit
                :AUTO
                :AUTO?
                :PACE
    :DATE
    :DATE?
    :ERRor?
    :TIME?
    :VERSion?

VXI
    :CONFigure
        :DNUMber?
        :HIERarchy?
            :ALL?
        :INFormation?
            :ALL?
        :LADDress?
        :NUMBer?
    :REGister
        :READ?
        :WRITe
    :RESet?
    :SELect

VXI
    :WSProtocol
        :COMMand
            :AHLine
            :AILine
            :AMControl
            :ANO
            [:ANY]
            :BAVailable
            :BNO
            :BRQ
            :CEVent
            :CLR
            :CLOCk
            :CRESponse
            :ENO
            :GDEVice
            :ICOMmander
            :RDEVice
            :RHANdlers
            :RHLine
            :RILine
            :RINTerrupter
            :RMODid
            :RPERror
            :RPRotocol
            :RSTB
            :RSAREa
            :SLModid
            :SLOCk
            :SUModid
            :TRIGger
        :MESSage
            :RECeive?
            :SEND
        :QUERy
            :AHLine?
            :AILine?
            :AMControl?
            :ANO?
            [:ANY?]
            :BNO?
            :BRQuest?
            :CEVent?
            :CRESponse?
            :ENO?
            :RDEVice?
            :RHANdlers?
            :RHLine?
            :RILine?
            :RINTerrupter?
            :RMODid?
            :RPERror?
            :RPRotocol?
            :RSAREa?
            :RSTB?
            :SLModid?
            :SUModid?
        :RESPonse?
```

Table A-14. System Instrument Non-SCPI Commands

```
DIAGnostic
    :BOOT
        :COLD
        [:WARM]
    :COMMunicate
        :SERial[0]
            [:OWNer]
            [:OWNer]?
        :SERial[n]
            :STORe
    :DOWNload
        CHECked
            [:MADDress]
            :SADDress
        [:MADDress]
        :SADDress
    :DRAM
        :AVAilable?
        :CREate
        :CREate?
    :DRIVer
        :INSTall
        :LIST
            [:ALL]?
            :FROM?
            :RAM?
            :ROM?
        :LOAD
            :CHECked
    :FROM
        :AVAilable
        :CREate
        :CREate?
        :SIZE?
DIAGnostic
    :INTerrupt
        :ACTivate
        :PRIority[n]
        :PRIority[n]?
        :RESPonse?
        :SETup[n]
        :SETup[n]?
    :NRAM
        :ADDRess?
        :CREate
        :CREate?
    :PEEK?
    :POKE
    :RDISK
        :ADDRess?
        :CREate
        :CREate?
    :UPLoad?
        [:MADDress]
        :SADDress

OUTPut
    :EXTernal
        :IMMediate
        :LEVel
            [:IMMediate]
            [:IMMediate]?
        :SOURce
        :SOURce?
        [:STATe]
        [:STATe]?
PROGram
    [:SELected]
        :CHECked
        :CHECked?
    :DEFine?

VXI
    :CONFigure
        :CTABle
        :DCTable
        :DLADdress?
        :DLIST?
        :ETABle
        :ITABle
    :READ?
    :RECeive[:MESSage]
    :RESet
    :SEND
        :COMMand
        [:MESSage]
    :WRITe
```

Table A-15. IEEE Mandated Common (*) Commands

*CLS	*RST
*ESE	*SRE
*ESE?	*SRE?
*ESR?	*STB?
*IDN?	*TST?
*OPC	*WAI
*OPC?	

Appendix B
HP E1406A Error Messages

Using This Appendix

This appendix shows how to read an instrument's error queue, discusses the types of command language-related error messages, and provides a table of all of the System Instrument's error messages and their probable causes.

Reading an Instrument's Error Queue

Executing the SYST:ERR? command reads the oldest error message from the instruments error queue and erases that error from the error queue. . The response format is: **<error number>,"<error description string>"**.

Example error message; `-113, "Undefined header"`

Positive error numbers are specific to an instrument. Negative error numbers are command language-related and discussed in "Error Types" on page 250. Command language-related errors also set a corresponding bit in the Standard Event Status Register (refer to Chapter 4 for more information).

Example: Reading the Error Queue

This program reads all errors (one error at a time, oldest to newest) from the System instrument's (command module) error queue. After reading each error, that error is automatically erased from the queue. When the error queue is empty, this program returns: `+0, "No error"`.

```
10  OPTION BASE 1
20  DIM Message$[256]                  Create array for error message.
30   REPEAT                            Repeat next 3 lines until error
                                       number = 0.
40    OUTPUT  70900;"SYST:ERR?"        Read error number and message.
50    ENTER  70900;Code,Message$       Enter error number and message.
60    PRINT Code,Message$              Print error number and message.
70   UNTIL Code=0
80  END
```

Error codes read from the error queue are preceded by the number 21. For example, error code 11 displayed on a monitor appears as `2111` if read from the error queue instead.

Error Types

Negative error numbers are language-related and categorized as shown in Table B-1. Positive error numbers are instrument specific and for the System instrument are summarized in Table B-2. For other instruments, refer to their own user's manual for a description of error messages.

Table B-1. Negative Error Numbers

Error Number	Error Type
–199 to –100	Command Errors
–299 to –200	Execution Errors
–399 to –300	Device-Specific Errors
–499 to –400	Query Errors

Command Errors

A command error means the instrument cannot understand or execute the command. When a command error occurs, it sets the Command Error bit (bit 5) in the Standard Event Status Register. Command errors can be caused by:

- A syntax error was detected in a received command or message. Possible errors include a data element which violates the instrument's listening formats or is of the wrong type (binary, numeric, etc.) for the instrument.
- An unrecognizable command header was received. Unrecognizable headers include incorrect SCPI headers and incorrect or unimplemented common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of a common command.

Execution Errors

An execution error indicates the instrument is incapable of doing the action or operation requested by a command. When an execution error occurs, it sets the Execution Error bit (bit 4) in the Standard Event Status Register. Execution errors can be caused by the following:

- A parameter within a command is outside the limits or inconsistent with the capabilities of an instrument.
- A valid command could not be executed because of an instrument failure or other condition.

Device-Specific Errors

A device-specific error indicates an instrument operation did not complete, possibly due to an abnormal hardware or firmware condition (self-test failure, loss of calibration or configuration memory, and so forth). When a device-specific error occurs, it sets the Device-Specific Error bit (bit 3) in the Standard Event Status Register.

Query Errors

A query error indicates a problem has occurred in the instrument's output queue. When a query error occurs, it sets the Query Error bit (bit 2) in the Standard Event Status Register. Query errors can be caused by the following:

- An attempt was made to read the instrument's output queue when no output was present or pending.
- Data in the instrument's output queue has been lost for some reason.

Table B-2. Error Messages and Causes

Error Messages and Causes		
Code	**Message**	**Cause**
–101	Invalid character	Unrecognized character in specified parameter.
–102	Syntax error	Command is missing a space or comma between parameters.
–103	Invalid separator	Command parameter is separated by some character other than a comma.
–104	Data type error	The wrong data type (for example, number, character, string expression) was used when specifying a parameter.
–108	Parameter not allowed	Parameter specified in a command which does not require one.
–109	Missing parameter	No parameter specified in the command in which a parameter is required.
–113	Undefined header	Command header was incorrectly specified.
–123	Numeric overflow	A parameter specifies a value greater than the command allows.
–128	Numeric data not allowed	A number was specified for a parameter when a letter is required.
–131	Invalid suffix	Parameter suffix incorrectly specified (e.g. .5SECOND rather than .5S or .5SEC).
–138	Suffix not allowed	Parameter suffix is specified when one is not allowed.
–141	Invalid character data	The discrete parameter specified is not allowed (e.g. TRIG:SOUR INT - INT is not a choice).
–160	Block data error	The block sent either contained more data then the Flash ROMS could hold or the block count field disagreed with the number of bytes actually sent.
–178	Expression data not allowed	A parameter other than the channel list is enclosed in parentheses.
–211	Trigger ignored	Trigger occurred from a source other than the specified source.
–222	Data out of range	The parameter value specified is too large or too small.
–224	Illegal parameter value	The numeric value specified is not allowed.
–240	Hardware error	Error was encountered while attempting to erase Flash ROMs or Flash ROMs failed to respond correctly to the programming sequence.
–252	Missing media	No programmable ROM was found, or hardware malfunction.
–253	Corrupt media	An incorrect checksum was read from the programmed ROMs. This is indicative of a ROM hardware malfunction or a data transmission error.
–258	Media protected	A command was executed with the "RUN/LOAD" switch in the "RUN" position when it should be in the "LOAD" position.
–310	System error	If caused by *DMC, then macro memory is full.
–350	Too many errors	The error queue is full as more than 30 errors have occurred.

Table B-2. Error Messages and Causes (*continued*)

Error Messages and Causes		
Code	**Message**	**Cause**
–410	Query interrupted	Data is not read from the output buffer before another command is executed.
–420	Query unterminated	Command which generates data not able to finish executing due to a multimeter configuration error.
–430	Query deadlocked	Command execution cannot continue since the mainframe's command input, and data output buffers are full. Clearing the instrument restores control.
+1000	Out of memory	There is not enough available Flash ROM to create a FROM driver area.
+1500	External trigger source already allocated	"Event In" signal already allocated to another instrument such as a Switchbox.
+2002	Invalid logical address	A value less than 0 or greater than 255 was specified for logical address.
+2003	Invalid word address	An odd address was specified for a 16-bit read or write. Always use even addresses for 16-bit (word) accesses.
+2005	No card at logical address	A non-existent logical address was specified with the VXI:READ? or VXI:WRITE command.
+2013	Word serial protocol error	An error has occurred in a word serial protocol command.
+2016	Byte count is not a multiple of two	The program block sent had an improper size.
+2022	Config warning, RAM Disc Volume contents lost	A RAM Disc volume was removed after successful programming of the Flash ROMs.
+2023	Flash driver area not created	An attempt was made to install drivers before the DIAG:DRIV:INST command was executed.
+2024	Flash driver area already installed	An attempt was made to install drivers after the DIAG:DRIV:INST command had already been executed.
+2101	Failed Device	VXI device failed its self test.
+2102	Unable to combine device	Device type can not be combined into an instrument such as a scanning voltmeter or a switchbox.
+2103	Config warning, Device driver not found	ID of device does not match list of drivers available. Warning only.
+2105	Config error 5, A24 memory overflow	More A24 memory installed in the mainframe than can be configured into the available A24 memory space.
+2108	Config error 8, Inaccessible A24 memory	A24 memory device overlaps memory space reserved by the mainframe's operating system.
+2110	Config error 10, Insufficient system memory	Too many instruments installed for the amount of RAM installed in the mainframe. Cannot configure instruments. Only the system instrument is started.
+2111	Config error 11, Invalid instrument address	A device's logical address is not a multiple of 8 and the device is not part of a combined instrument.
+2112	Invalid user-defined commander logical address	The commander assigned to a device by a user-defined Configuration Table does not assign it a secondary address.
+2114	Invalid user-defined secondary address	A secondary address assigned by a user configuration table is illegal.
+2115	Duplicate secondary address	A secondary address specified by a user configuration table is used more than once.

Table B-2. Error Messages and Causes (*continued*)

Error Messages and Causes		
Code	**Message**	**Cause**
+2116	Invalid servant area	The logical address plus servant area of a commander is greater than 255 or greater than that of a superior commander within this tree.
+2117	Slot 0 functions disabled	A command module is in slot 0 but slot 0 switches are in the disabled position.
+2118	Invalid commander logical address	A device does not have a valid commander.
+2119	BNO failed	Sending a BEGIN Normal Operation command to a device failed.
+2120	Write ready timeout	A message based device failed to become write ready.
+2121	Read ready timeout	A message based device failed to become read ready.
+2122	ERR* asserted	The ERR* bit is asserted in a device's response register.
+2123	ENO failed	Sending an End Normal Operation command to a device failed.
+2124	Interrupt line unavailable	No line is available for a programmable interrupt handler. All lines are used or duplicate.
+2125	Invalid user defined handler	The user defined interrupt table specifies a device that is not a programmable interrupt handler, or does not exist.
+2126	Invalid user defined interrupter	The user defined interrupt table specifies a device that is not a programmable interrupter, or does not exist.
+2127	Diagnostic mode on	HP-IB address switch bit 6 is set wrong (warning only).
+2128	Resource Manager not in Slot 0	A command module is configured for slot 0 and resource manager but is installed in another slot (warning only).
+2129	Warning, Sysfail detected	A device was asserting SYSFAIL on the backplane during startup.
+2130	Pseudo instrument logical address unavailable	A physical device has the same logical address as IBASIC (240).
+2131	File system start up failed	Insufficient system resources to allow the IBASIC file system to start.
+2133	Invalid UDEF memory block	Invalid memory block in user defined memory table.
+2134	UDEF memory block unavailable	The same base address or memory are specified more than once in the memory table, or the addresses in the specified block are already in use.
+2135	Invalid UDEF address space	The address specified in the memory table is A24 but the device is A32, or vice versa.
+2136	Duplicate UDEF memory LADD	A logical address is specified more than once in the memory table. This does not apply to VME devices (address = -1).
+2137	Invalid UDEF CNFG table	The valid flag in the command/servant hierarchy table is not set to 1.
+2138	Invalid UDEF CNFG table data	There are more than 254 entries in the commander/servant hierarchy table.
+2139	Invalid UDEF DC table	The valid flag in the dynamic configuration table is not set to 1.
+2140	Invalid UDEF DC table data	There are more than 254 entries in the dynamic configuration table.
+2141	Invalid UDEF Interrupter	The logical address specified for an interrupter is a device that is not an interrupter.

Table B-2. Error Messages and Causes (*continued*)

Error Messages and Causes		
Code	**Message**	**Cause**
+2142	Invalid UDEF INTR table	The interrupter table valid flag is not 1.
+2143	Invalid UDEF MEM table	The valid flag in the memory table is not set to 1.
+2144	Invalid UDEF MEM table data	An invalid logical address is specified in the memory table.
+2145	Warning, Non-Volatile RAM contents lost	Non-volatile RAM was corrupted, a cold boot was executed, or non-volatile RAM was removed after the successful programming of the Flash ROMs.
+2146	MESG based open access failed	I or I4 device is violating VXI specification.
+2147	Granted device not found	An HP E1406A which is not a slot zero device or a resource manager could not find a module that was granted to its servant area.
+2148	Config warning 48, Driver RAM contents lost	Driver RAM was corrupted, a cold boot was executed, or Driver RAM was removed after the successful programming of the Flash ROMs.
+2149	VME system controller disabled	VME SYSTEM CONTROLLER switch is disabled on the HP E1406A module.
+2150	Extender not slot 0 device	VXIbus extender in remote mainframe is not in slot 0 of its mainframe.
+2151	Invalid extender LADD window	MXI extender cannot be configured with a valid LADD window.
+2152	Device outside of LADD window	A device is located outside the allowable logical address window range of an MXIbus extender.
+2153	Invalid extender A24 window	MXIbus extender cannot be configured with a valid A24 memory window.
+2154	Device outside of A24 window	An A24 memory device is located outside the allowable logical address window range of an MXIbus extender.
+2155	Invalid extender A32 window	MXIbus extender cannot be configured with a valid A32 memory window.
+2156	Device outside of A32 window	An A32 memory device is located outside the allowable logical address window range of an MXIbus extender.
+2157	Invalid UDEF LADD window	User defined logical address window has incorrect base address or size.
+2158	Invalid UDEF A16 window	User defined A16 memory window has incorrect base address or size.
+2159	Invalid UDEF A24 window	User defined A24 memory window has incorrect base address or size.
+2160	Invalid UDEF A32 window	User defined A32 memory window has incorrect base address or size.
+2161	Invalid UDEF EXT table	The valid flag in the extender table is not set to 1.
+2162	Invalid UDEF extender table data	There are more than 254 records in the extender table.
+2163	Unsupported UDEF TTL trigger	There is an extender table TTL trigger entry for a device which does not support TTL triggers.
+2164	Unsupported UDEF ECL trigger	There is an extender table ECL trigger entry for a device which does not support ECL triggers.
+2165	Device not in configure state	A message based device was not in CONFIGURE state during reboot.

Table B-2. Error Messages and Causes (*continued*)

Error Messages and Causes		
Code	**Message**	**Cause**
+2166	INTX card not installed	The INTX daughter card on the VXI-MXI module is not installed or is not functioning correctly.
+2167	Config warning, Flash ROM driver contents lost	The contents of the Flash ROM driver area have been corrupted.
+2201	Unexpected interrupt from message based card	A message based card interrupted when an interrupt service routine has not been set up.
+2202	Unexpected interrupt from non-message based card	A register based card interrupted when an interrupt service routine had not been set up.
+2809	Interrupt line has not been set up	A DIAG:INT:ACT or DIAG:INT:RESP command was executed before setting the interrupt with DIAG:INT:SET.
+2810	Not a handler for this line	An attempt was made to set up an interrupt with DIAG:INT:SET for a line that has no handler. (see VXI:CONF:ITAB).

Start-up Error Messages and Warnings

Start-up error messages and warnings are most often generated just after the mainframe is powered-up or rebooted (DIAG:BOOT command). These messages can be read from the error queue using the SYST:ERR? command. We recommend that you include a routine at the beginning of your application programs which checks for start-up errors before the program tries to access individual instruments. See your *VXIbus Configuration Guide* for an example program.

Table B-3. Start-Up Error Messages and Warnings

Start-Up Error Messages and Warnings		
Code	**Message**	**Cause**
1	Failed Device	VXI device failed its self test.
2	Unable to combine device	Device type can not be combined into an instrument such as a scanning voltmeter or a switchbox.
3	Config warning, Device driver not found	ID of device does not match list of drivers available. Warning only.
4	DC device block too big	Dynamically configured device address block is greater than 127.
5	Config error 5, A24 memory overflow	More A24 memory is installed in the mainframe than can be configured into the available A24 memory space.
6	A32 memory overflow	More A32 memory is installed in the mainframe than can be configured into the available A32 memory space.
7	DC device move failed	A dynamically configured device failed to move to a new logical address.
8	Config error 8, Inaccessible A24 memory	An A24 memory device overlaps a memory space reserved by the mainframe's operating system.
9	Unable to move DC device	The block size for a set of address-blocked Dynamically Configured devices is too large for the available space or an attempt was made to move a Dynamically Configured device to an already assigned Logical Address.Cannot configure instruments. Only the system instrument is started.

Table B-3. Start-Up Error Messages and Warnings (*continued*)

Start-Up Error Messages and Warnings		
Code	**Message**	**Cause**
10	Config error 10, Insufficient system memory	Too many instruments installed for the amount of RAM installed in the mainframe. Cannot configure instruments. Only the system instrument is started.
11	Config error 11, Invalid instrument address	A device's logical address is not a multiple of 8 and the device is not part of a combined instrument.
12	Invalid user defined commander logical address	The commander assigned to a device by a user defined Configuration Table does not assign it a secondary address.
14	Invalid user defined secondary address	A secondary address assigned by a user configuration table is illegal.
15	Duplicate secondary address	A secondary address specified by a user configuration table is used more than once.
16	Invalid servant area	The logical address plus servant area of a commander is greater than 255 or greater than that of a superior commander within this tree.
17	Slot 0 functions disabled	A command module is in slot 0 but slot 0 switches are in the disabled position.
18	Invalid commander logical address	A device does not have a valid commander.
19	BNO failed	Sending a BEGIN Normal Operation command to a device failed.
20	Write ready timeout	A message based device failed to become write ready.
21	Read ready timeout	A message based device failed to become read ready.
22	ERR* asserted	The ERR* bit is asserted in a device's response register.
23	ENO failed	Sending an End Normal Operation command to a device failed.
24	Interrupt line unavailable	No line is available for a programmable interrupt handler. All lines are used or duplicate.
25	Invalid user defined handler	The user defined interrupt table specifies a device that is not a programmable interrupt handler, or does not exist.
26	Invalid user defined interrupter	The user defined interrupt table specifies a device that is not a programmable interrupter, or does not exist.
27	Diagnostic mode on	HP-IB address switch bit 6 is set wrong (warning only).
28	Resource Manager not in Slot 0	A command module is configured for slot 0 and resource manager but is installed in another slot (warning only).
29	Warning, Sysfail detected	A device was asserting SYSFAIL on the backplane during start-up.
30	Pseudo instrument logical address unavailable	A physical device has the same logical address as IBASIC (240).
31	File system startup failed	Insufficient system resources to allow the IBASIC file system to start.
32	Inaccessible A32 memory	Device has A32 memory below 200000000_{16} or above $DFFFFFFF_{16}$
33	Invalid UDEF memory block	Invalid memory block in user defined Memory table.
34	UDEF memory block unavailable	The same base address or memory are specified more than once in the memory table, or the addresses in the specified block are already in use.
35	Invalid UDEF address space	The address specified in the memory table is A24 but the device is A32, or vice versa.

Table B-3. Start-Up Error Messages and Warnings (*continued*)

Start-Up Error Messages and Warnings		
Code	**Message**	**Cause**
36	Duplicate UDEF memory LADD	A logical address is specified more than once in the memory table. This does not apply to VME devices (address = -1).
37	Invalid UDEF CNFG table	The valid flag in the command/servant hierarchy table is not set to 1.
38	Invalid UDEF CNFG table data	There are more than 254 entries in the commander/servant hierarchy table.
39	Invalid UDEF DC table	The valid flag in the dynamic configuration table is not set to 1.
40	Invalid UDEF DC table data	There are more than 254 entries in the dynamic configuration table.
41	Invalid UDEF Interrupter	The logical address specified for an interrupter is a device that is not an interrupter.
42	Invalid UDEF INTR table	The interrupter table valid flag is not 1.
43	Invalid UDEF MEM table	The valid flag in the memory table is not set to 1.
44	Invalid UDEF MEM table data	An invalid logical address is specified in the memory table.
45	Warning, NVRAM contents lost	NVRAM was corrupted or a cold boot was executed.
46	MESG based open access failed	I or I4 device is violating VXI specification.
47	Granted device not found	
48	Warning, DRAM contents lost	Driver RAM was corrupted or a cold boot was executed.
49	VME system controller disabled	VME SYSTEM CONTROLLER switch is disabled on the HP E1406A module.
50	Extender not slot 0 device	VXIbus extender in remote mainframe is not in slot 0 of its mainframe.
51	Invalid extender LADD window	MXI extender cannot be configured with a valid LADD window.
52	Device outside of LADD window	A device is located outside the allowable logical address window range of an MXIbus extender.
53	Invalid extender A24 window	MXIbus extender cannot be configured with a valid A24 memory window.
54	Device outside of A24 window	An A24 memory device is located outside the allowable logical address window range of an MXIbus extender.
55	Invalid extender A32 window	MXIbus extender cannot be configured with a valid A32 memory window.
56	Device outside of A32 window	An A32 memory device is located outside the allowable logical address window range of an MXIbus extender.
57	Invalid UDEF LADD window	User defined logical address window has incorrect base address or size.
58	Invalid UDEF A16 window	User defined A16 memory window has incorrect base address or size.
59	Invalid UDEF A24 window	User defined A24 memory window has incorrect base address or size.
60	Invalid UDEF A32 window	User defined A32 memory window has incorrect base address or size.
61	Invalid UDEF EXT table	The valid flag in the extender table is not set to 1.

Table B-3. Start-Up Error Messages and Warnings (*continued*)

Start-Up Error Messages and Warnings		
Code	**Message**	**Cause**
62	Invalid UDEF extender table data	There are more than 254 records in the extender table.
63	Unsupported UDEF TTL trigger	There is an extender table TTL trigger entry for a device which does not support TTL triggers.
64	Unsupported UDEF ECL trigger	There is an extender table ECL trigger entry for a device which does not support ECL triggers.
65	Device not in configure state	A message based device was not in CONFIGURE state during reboot.
66	INTX card not installed	The INTX daughter card on the VXI-MXI module is not installed or is not functioning correctly.
67	Flash ROM driver contents lost	The contents of the Flash ROM driver area have been corrupted.

Appendix C
HP E1406A Command Module A16 Address Space

About This Appendix

Many Hewlett-Packard VXIbus devices are register-based devices which do not support the VXIbus word serial protocol. When an SCPI command is sent to a register-based device, the HP E1406A Command Module parses the command and programs the device at the register level.

Register-based programming is a series of **reads** and **writes** directly to the device registers. This increases throughput since it eliminates command parsing.

This appendix contains an address map of A16 address space in the command module. It shows how to determine the base address and register offset for register-based devices mapped into A16 space. Refer to the individual plug-in module manuals for details on device is programming at the register level.

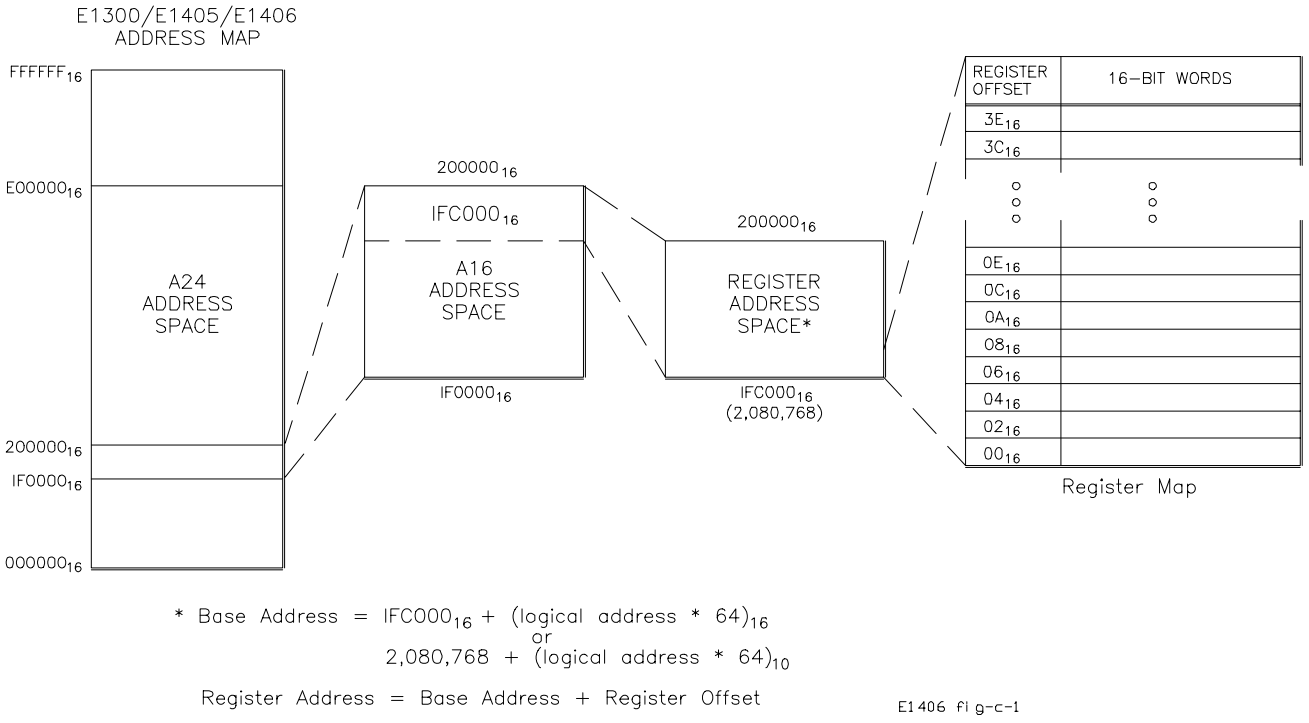


Figure C-1. HP E1406A Command Module A16 Address Space

Register Addressing

Register addresses for register-based devices are located in the upper 25% of VXI A16 address space. Every VXI device (up to 256 devices per Command Module) is allocated a 64 byte block of addresses. A device may or may not use the entire block of addresses. Figure C-1 shows the location of A16 address space in the HP E1406A Command Module.

The Base Address

When you are reading or writing to a device register, a hexadecimal or decimal register address is specified. This address consists of a base address plus a register offset.

Determining the Base Address

The base address of a device in A16 address space is computed as:

$$1FC000_{16} + (LADDR * 64)_{16}$$

or

$$2{,}080{,}768_{10} + (LADDR * 64)_{10}$$

where $1FC000_{16}$ ($2{,}080{,}768_{10}$) is the starting location of the VXI A16 addresses, LADDR is the device's logical address, and 64 is the number of address bytes per register-based device. For example, the HP E1411B multimeter has a factory set logical address of 24. If this address is not changed, the multimeter will have a base address of:

$$1FC000_{16} + (24 * 64)_{16}$$

$$1FC000_{16} + 600_{16} = \mathbf{1FC600}_{16}$$

or

$$2{,}080{,}768_{10} + (24 * 64)_{10}$$

$$2{,}080{,}768_{10} + 1536_{10} = \mathbf{2{,}082{,}304}_{10}$$

Register Offset

The register offset is the register's location in the block of 64 address bytes. For example, the HP E1411B multimeter's Command Register has an offset of 08_{16}. When you write a command to this register, the offset is added to the base address to form the register address:

$$1FC600_{16} + 08_{16} = \mathbf{1FC608}_{16}$$

or

$$2{,}082{,}304_{10} + 8_{10} = \mathbf{2{,}082{,}312}_{10}$$

Appendix D
Sending Binary Data Over RS-232

About This Appendix

This appendix describes the procedure for sending pure binary data over an RS-232 interface. The formatting described is used in the DIAG:DOWN:CHEC[:MADD], DIAG:DOWN:CHEC:SADD, and DIAG:DRIV:LOAD:CHEC commands. This appendix contains the following main sections.

Formatting Binary Data for RS-232 Transmission

The most straightforward way to send a block of data is to open the data file, read the next byte from the file, and send it to the System Instrument until you reach the end of file. However, binary data cannot be sent to the system instrument as is. It must be converted into a format that will not conflict with the special characters that the RS-232 interface recognizes. This is done by sending only one half byte (a nibble) at a time.

To prevent this nibble from being confused with a special character, bit 7 of the nibble is set to one. This gives all data bytes in the block values greater than 127 so they are not confused with ASCII characters. It also doubles the size of the file to be sent and the transmission time for the file. Since a transmission error that required re-transmission of the entire data block would be very time consuming, a 3-bit error code (which allows for correction of single bit errors) is added to the transmission byte. The following format is sent for each nibble:

Bit #	7	6	5	4	3	2	1	0
	1	Correction Code			Data			

The error correction code is based on the nibble of data sent. The easiest way to implement this code is to use Table D-1. It is indexed based on the value of the nibble to send out, so there are 16 elements to the table.

Table D-1. Correction Codes for RS-232 Transmission

Data Value	Correction Code	Byte in Hex	Byte in Decimal
0	0	80_{16}	128
1	7	$F1_{16}$	241
2	6	$E2_{16}$	226
3	1	93_{16}	147
4	5	$D4_{16}$	212
5	2	$A5_{16}$	165
6	3	$B6_{16}$	182
7	4	$C7_{16}$	199
8	3	$B8_{16}$	184
9	4	$C9_{16}$	201
10	5	DA_{16}	218
11	2	AB_{16}	171
12	6	EC_{16}	236
13	1	$9D_{16}$	157
14	0	$8E_{16}$	142
15	7	FF_{16}	255

Sending Binary Data Over RS-232

The RS-232 interface differs from the HP-IB interface in that there is no device addressing built into the interface definition. Device addressing must be done on top of the RS-232 functions. This addressing is done through the same mechanism as the terminal-based front panel, and must be done either by the transfer program or manually before starting the transfer program.

Setting Up the Mainframe

There are two commands (SI - Select Instrument and SA - Select Address) that can be used at the `Select an instrument` interface. The `Select an instrument` interface can always be reached by sending the CTRL-D character (ASCII 4) over the RS-232 line. Once there, the System instrument can be reached by sending the command SI SYSTEM followed by a carriage return. All output after this command will be directed to/from the System instrument until another CTRL-D is received. The following sequence will make sure that the mainframe is set up and ready.

1. Send CTRL-D (ASCII 4) to get to the `Select an instrument` interface.
2. Send ST UNKNOWN and a carriage return to insure that the interface is set to dumb terminal mode.
3. Send SI SYSTEM and a carriage return to get the attention of the System instrument.
4. Send CTRL-C to clear the system.
5. Send *RST and a carriage return to put the System instrument in a known state.

The program must then send the binary data. This block of data should include the command DIAG:DOWN:CHEC followed by the address to download to, and an IEEE 488.2 arbitrary block header. This block header can be either definite or indefinite. The advantage of using an indefinite block header is that you do not need to know the length of the data block. The indefinite block header is #0. With the DIAG:DOWN:CHEC command, an indefinite block is terminated with the "!" character followed by a carriage return. The "!" character is not considered part of the block.

A definite block only requires the ASCII carriage return character as terminator. The definite block starts with #. This is followed by a single digit that shows the number of digits in the length field, which is followed by the actual length of the block, not counting the header. For instance, a block of 1000 bytes would have a definite block header of #41000. Due to the formatting required, the size of the block when using the DIAG:DOWN:CHEC command is twice the length of the data in bytes.

Once the block header has been sent, the actual data is sent. Since the buffer size of the System instrument RS-232 Interface is limited to 79 bytes, the buffer must be flushed (passed to an instrument parser) before it reaches 79 bytes. This can be done by sending a carriage return. The first carriage return should be included in the binary file after the buffer header. Sending it before this would result in the parser determining that there are not enough parameters and producing an error condition. Once transmission of the actual data begins, a carriage return should be included after every 78 bytes.

Note The carriage returns are not considered part of the block count.

After the last byte of data, there must be a carriage return to terminate the transmission for a definite block or a "!" and carriage return for an indefinite block.

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X