VXI Technology VT2216A
VXI/SCSI Interface Module

User's Guide

Part Number 82-0072-000

Printed in U.S.A.
Print Date: July 30, 2004

2031 Main Street, Irvine, CA 92614-6509 U.S.A.

Notices

The information contained in this manual is subject to change without notice. VXI Technology makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. VXI Technology shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance or use of the material.

Trademarks

Window®, Windows NT®, Windows 2000®, and MS-DOS® are U.S. registered trademarks of Microsoft Corporation.

Netscape is a U.S. trademark of Netscape Communications Corporation.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227- 7013.

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509, USA

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Safety Summary

The following general safety precautions must be observed during all phases of operation of this instrument. 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 instrument. VXI Technology, Inc. assumes no liability for the customer's failure to comply with these requirements.

GENERAL

This product is a Safety Class 1 instrument (provided with a protective earth terminal). The protective features of this product may be impaired if it is used in a manner not specified in the operation instructions.

All Light Emitting Diodes (LEDs) used in this product are Class 1 LEDs as per IEC 60825-1.

ENVIRONMENTAL CONDITIONS

This instrument is intended for indoor use in an installation category II, pollution degree 2 environment. It is designed to operate at a maximum relative humidity of 95% and at altitudes of up to 2000 meters. Refer to the Technical Specifications document for the ac mains voltage requirements and ambient operating temperature range.

BEFORE APPLYING POWER

Verify that the product is set to match the available line voltage, the correct fuse is installed and all safety precautions are taken. Note the instrument's external markings described under Safety Symbols.

GROUND THE INSTRUMENT

To minimize shock hazard, the instrument chassis and cover must be connected to an electrical protective earth ground. The instrument must be connected to the ac power mains through a grounded power cable, with the ground wire firmly connected to an electrical ground (safety ground) at the power outlet. Any interruption of the protective (grounding) conductor or disconnection of the protective earth terminal will cause a potential shock hazard that could result in personal injury.

FUSES

Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuse holders. To do so could cause a shock or fire hazard.

DO NOT OPERATE IN AN EXPLOSIVE ATMOSPHERE

Do not operate the instrument in the presence of flammable gases or fumes.

DO NOT REMOVE THE INSTRUMENT COVER

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made only by qualified service personnel.

Instruments that appear damaged or defective should be made inoperative and secured against unintended operation until they can be repaired by qualified service personnel.

WARNING

The WARNING sign denotes a hazard. It calls attention to a procedure, practice or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a WARNING sign until the indicated conditions are fully understood and met.

Caution

The CAUTION sign denotes a hazard. It calls attention to an operating procedure or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a CAUTION sign until the indicated conditions are fully understood and met.

Safety Symbols

Warning, risk of electric shock

Warning, hot surface

Caution, refer to accompanying documents

Alternating current

Both direct and alternating current

Three-phase alternating current

Earth (ground) terminal

Protective earth (ground) terminal

Frame or chassis terminal

Terminal is at earth potential.
Used for measurement and control circuits designed to be operated with one terminal at earth potential.

Terminal for Neutral conductor on permanently installed equipment.

Terminal for Line conductor on permanently installed equipment.

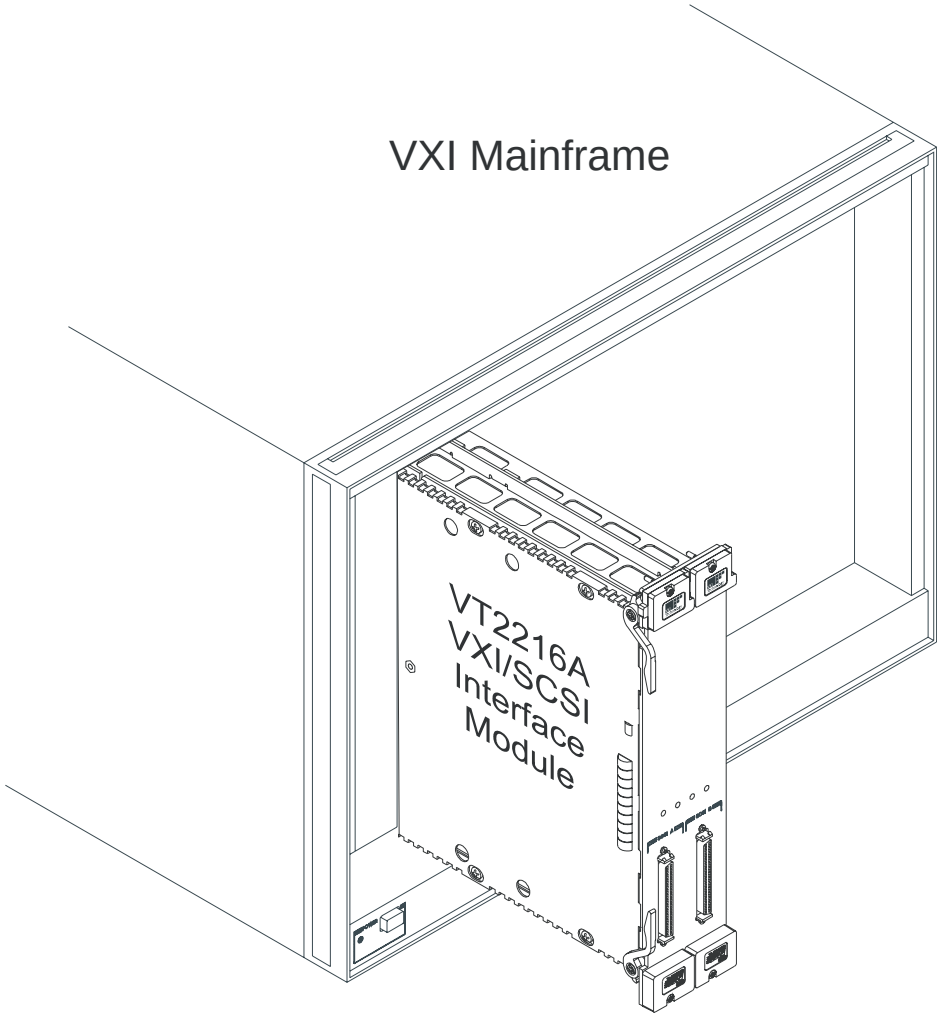
Standby (supply). Units with this symbol are not completely disconnected from ac mains when this switch is off.
To completely disconnect the unit from ac mains, either disconnect the power cord or have a qualified electrician install an external switch.

VT2216A at a Glance

The VT2216A VXI/SCSI Interface module is a high-speed dual SCSI interface with optional internal disk drives. Option 1 adds a 73 GB drive and Option 2 adds two 73 GB drives. The VT2216A is compatible with software written for the Agilent/HP E1562. However, its SCSI electrical interface is not compatible with the Agilent/HP E1562.

Caution

Do not connect high-voltage differential (HVD) or fast-wide differential devices to the module's SCSI connectors. The VT2216A contains low-voltage differential LVD circuits that may be damaged if connected to HVD circuits.



In This Book

This book documents the VT2216A VXI/SCSI Interface module. It provides:

- Installation and service procedures (calibration not required)
- Operating information
- VXI*plug&play* command reference
- Sequence operations reference
- SCPI command reference
- LIF library reference

Contents

Support Resources

Support resources for this product are available on the Internet and at VXI Technology customer support centers.

VXI Technology
World Headquarters

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509

Phone: (949) 955-1894
Fax: (949) 955-3041

VXI Technology
Cleveland Instrument Division

VXI Technology, Inc.
7525 Granger Road, Unit 7
Valley View, OH 44125

Phone: (216) 447-8950
Fax: (216) 447-8951

VXI Technology
Lake Stevens Instrument Division

VXI Technology, Inc.
1924 - 203 Bickford
Snohomish, WA 98290

Phone: (425) 212-2285
Fax: (425) 212-2289

Technical Support

Phone: (949) 955-1894
Fax: (949) 955-3041
E-mail: support@vxitech.com

Visit http://vxitech.com for worldwide support sites and service plan information.

Installing the VT2216A

Installing the VT2216A

This chapter contains instructions for installing the VT2216A VXI/SCSI Interface module and its libraries. This chapter also includes instructions for transporting and storing the module.

To Inspect the VT2216A

The VT2216A VXI/SCSI Interface module was carefully inspected both mechanically and electrically before shipment. It should be free of marks or scratches, and it should meet its published specifications upon receipt.

Note

The VT2216A does not require periodic calibration or performance testing.

If the module was damaged in transit, do the following:

- Save all packing materials.
- File a claim with the carrier.
- Call a VXI Technology sales and service office.

The VT2216A Checkist

The following items are included with the VT2216A VXI/SCSI Interface module:

- One CD-ROM containing VXI*plug&play* libraries, LIF libraries, sample programs, a PDF file of this book and online help (HTML files) for HP-UX 10.2, Windows NT 4.0 and Windows 2000 and later. The HTML files require a web browser that supports the HTML v3.2, JavaScript 1.2 and CSS1 standards, such as, Internet Explorer 4.0 or Netscape 4.0. In addition, the web browser's cookie support should be turned on to receive the full functionality of the online help.
- Two SCSI terminators (VXI Technology part number 1253-4010)
- VT2216A User’s Guide (this book)

To Install the VT2216A

If using the Agilent/HP E1406A Command module and an external computer with DOS based windows, use the Agilent/HP VXI Installation Consultant (VIC) to install the VT2216A module. VIC guides the user through the installation procedure, then tests the modules using the *TST? command. VIC may time out before the test is finished and display a "timed out" message. If this occurs, exit VIC and send the *TST? command. For instructions on sending the *TST? command, see "Troubleshooting the VT2216A" starting on page 29.

Caution

To protect circuits from static discharge, observe anti-static techniques whenever handling the VT2216A VXI/SCSI Interface module.

1. Set up the VXI mainframe. See the mainframe's installation guide.
2. Select two slots in the VXI mainframe for the VT2216A module.

   The VT2216A module's local bus receives ECL-level data from the module immediately to its left and outputs ECL-level data to the module immediately to its right. Every module using the local bus is keyed to prevent two modules from fitting next to each other unless they are compatible. If using the local bus, select two slots directly adjacent to the left of the data-receiving module.
3. Using a small screwdriver or similar tool, set the Logical address configuration switch on the VT2216A.

   Each module in the system must have a unique logical address. The factory default setting is 1001 0000 (144). If an Agilent/HP E1485 Signal Processor module will be controlling the VT2216A module, select an address within the Agilent E1485 module's servant area. If a GPIB command module will be controlling the VT2216A module, select an address that is a multiple of 8.
4. Using a small screwdriver or similar tool, set the Hardware configuration switches on the VT2216A.

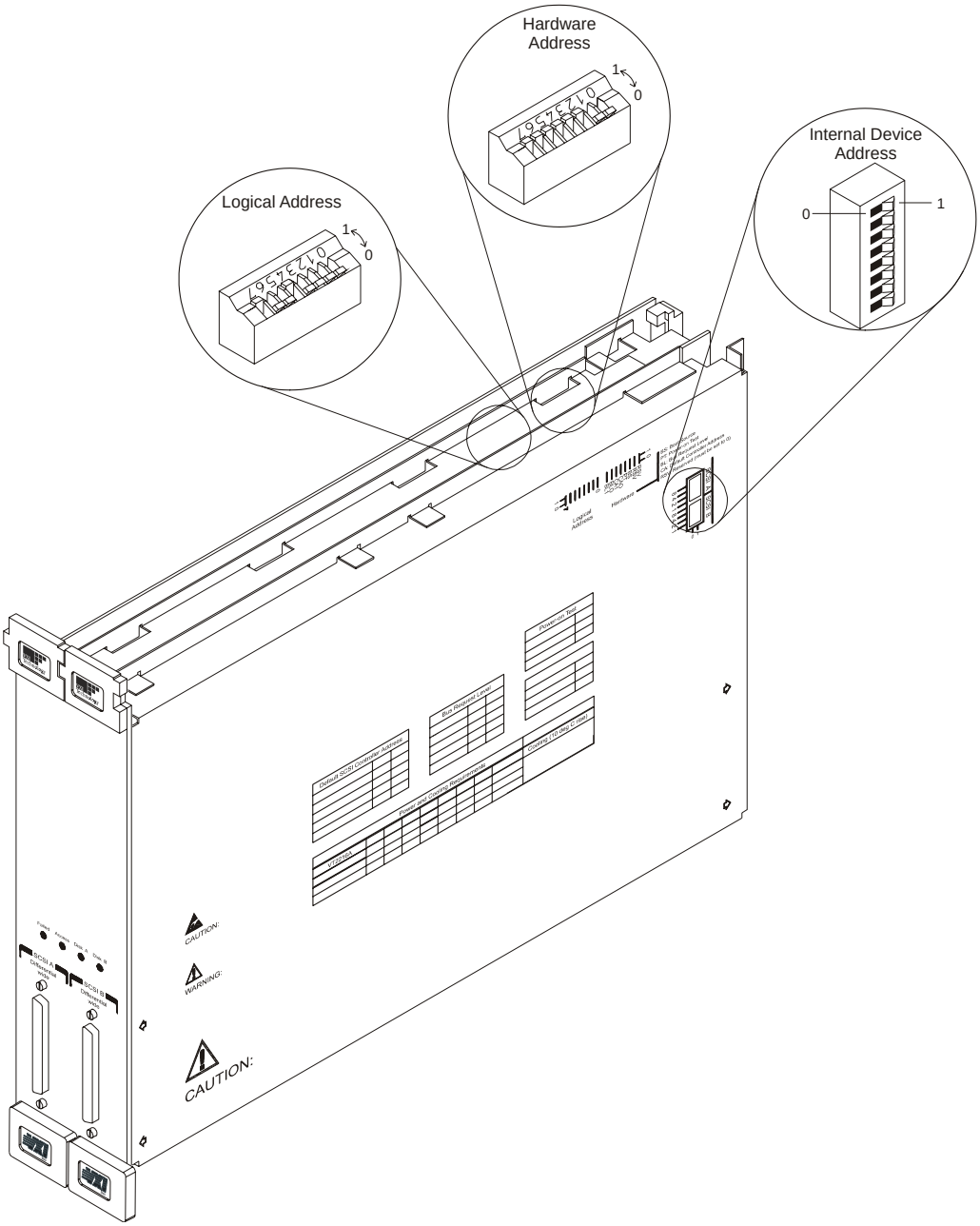
   The factory default setting is 1111 1100.

Note

Controller Address switches CA1 and CA0 set the SCSI Controller Address. SCSI Controller Address 7 (the factory default setting) identifies the SCSI Bus Master. If using more than one VT2216A on the same SCSI bus, set the Controller Address switches for the second VT2216A to Address 4, 5, or 6.

5. Using a small screwdriver or similar tool, set the Internal SCSI device address switches on the VT2216A.

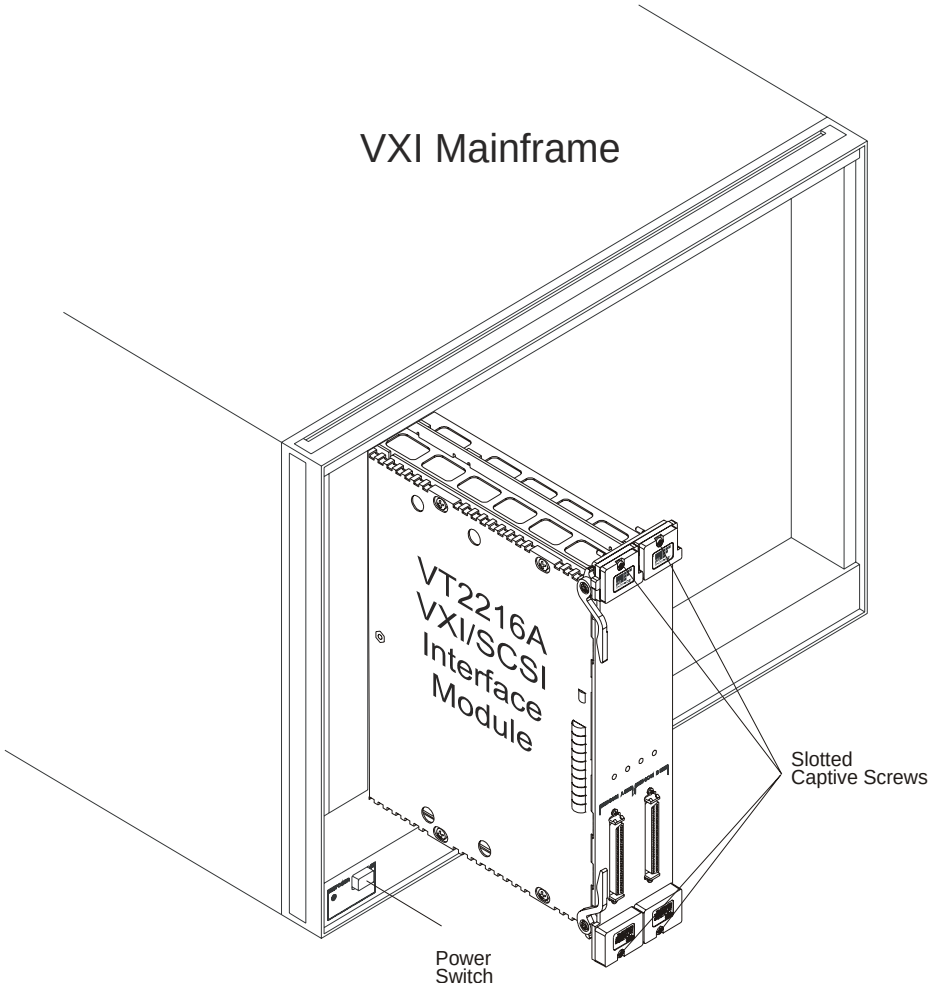
   The factory default setting is 0000 0000 (both disk drives at SCSI ID 0)


Hardware Address
Logical Address
Internal Device Address
1
0
CAUTION:
WARNING:
CAUTION:

6. Set the mainframe's power switch to standby.
7. Place the module's card edges (top and bottom) into the module guides in the slot.
8. With the extractor levers in the out position, slide the module into the mainframe until the module connects firmly with the backplane connectors. Make sure the module slides in straight.
9. Attach the module's front panel to the mainframe chassis using the module's captive mounting screws.
10. If an external SCSI disk or DAT drives are not being connected, terminate the SCSI connectors using the provided SCSI terminators.
11. If an external SCSI disk or DAT drives are being connecting, make sure they are low-voltage differential devices (LVD) and that the end of the SCSI bus is terminated.

Caution

Do not connect high-voltage differential (HVD) or fast-wide differential devices to the module's SCSI connectors. The VT2216A contains LVD circuits that may be damaged if connected to HVD circuits.



To Install the VT2216A Software

The VT2216A CD contains software for both MS Windows (Windows NT 4.0 and Windows 2000 or later) and HP-UX 10.2.

Before installing the VT2216A software, read the readme.txt file on the CD for updated information. On a PC, use Wordpad to read this and other readme.txt files.

Note

The VT2216A is software compatible with the HP E1562. The VT2216A software does not need to be installed for applications currently using the HP E1562.

Installing the VT2216A Software on Windows NT 4.0 or Windows 2000 and later

After installing the VISA library that comes with the VXI interface or the VXI embedded computer, do the following:

1. Insert the VXI*plug&play* Instrument Drivers and Product Manuals CD-ROM into the computer's CD-ROM drive.
2. Using Windows Explorer, select the CD-ROM drive. Navigate to \drivers\DAQ Drivers
3. Double-click the *driver_vxipnp_n2216_a_01_00.exe* file to begin the self-extracting installation program.
4. Follow the instructions on the screen to install the software.

The VXI*plug&play* library, the LIF library, include files, soft front panel application, commands, and other files are installed in the standard VXI*plug&play* directories in `C:\Vxipnp\winnt (bin, include, lib, agn2216)` for a Windows XP system.

For a standard installation, other files are installed in the following `C:\Program Files\Agilent\N2216\` sub-directories:

`e1562`	source for programs using SICL written for the Agilent/HP E1562
`e1562lif`	source for the LIF commands
`examc`	source for diskdiag and other programs using VISA
`help`	online help (HTML)
`pnpexamc`	source for examples that use the VT2216A VXI*plug&play* library
`pnplib`	source for the VT2216A VXI*plug&play* library
`pnpsfp`	source for the soft front panel (Visual Basic 6)
`pnpvee`	source for examples that are Agilent VEE compatible

The source files are provided as examples for using the VT2216A. The source directories contain project files for use with Visual Studio 6.0 and makefiles for HP-UX. For more information see the readme.txt in each directory.

Installing the VT2216A Software on HP-UX 10.2

After installing the VISA and/or SICL library that comes with the VXI interface or the VXI embedded computer, do the following:

1. Log in as root.
2. Insert the VXI*plug&play* Instrument Drivers and Product Manuals into the computer's CD-ROM drive.
3. Mount the CD-ROM file system. Use an appropriate modification of:
   `mount /dev/dsk/c201d2s0 /cdrom`
   where `c201d2s0` is the system file for the CD-ROM drive and `/cdrom` is the directory path of the root of the CD-ROM's file structure. (These names may differ from system to system.)
4. Type:
   `/usr/sbin/swinstall -s /cdrom/n2216a.tap`

   The installation program will proceed to install the software.

The VXI*plug&play* library, the LIF library (VISA version), include files, commands and other files are installed in the standard VXI*plug&play* directories in `/opt/vxipnp/hpux/` (`bin`, `include` and `agn2216`). The SICL versions of the LIF library and LIF commands are installed in `/opt/e1562/` (`bin`, `include`, `lib`).

Other files are installed in `/opt/agn2216/` sub-directories as described above.

To Transport the Module

- Package the module using the original factory packaging or packaging identical to the factory packaging.

  Containers and materials identical to those used in factory packaging are available through VXI Technology.
- If returning the module to VXI Technology for service, attach a tag describing the following:
  - Type of service required
  - Return address
  - Model number
  - Full serial number

  In any correspondence, refer to the module by model number and full serial number.
- Mark the container FRAGILE to ensure careful handling.
- If necessary to package the module in a container other than original packaging, observe the following (use of other packaging is not recommended):
  - Wrap the module in heavy paper or anti-static plastic.
  - Protect the front panel with cardboard.
  - Use a double-wall carton made of at least 350-pound test material.
  - Cushion the module to prevent damage.

Caution Do not use styrene pellets in any shape as packing material for the module. The pellets do not adequately cushion the module and do not prevent the module from shifting in the carton. In addition, the pellets create static electricity that can damage electronic components.

To store the module

Store the module in a clean, dry, and static free environment.

For other requirements, see storage and transport restrictions in the *VT2216A VXI/SCSI Interface Module Technical Specifications*.

Troubleshooting the VT2216A

Introduction

The troubleshooting procedure in this chapter uses a program that automatically runs the following commands from an MS-DOS® window:

- *IDN - identifies the module.
- *TST - checks the general operation of the Main assembly and SCSI assembly.
- DIAG:BOAR:MAIN - tests the Main assembly.
- DIAG:BOAR:SCSI - tests the SCSI assembly.
- DIAG:SCSI:DEV - checks the interface for a specific SCSI controller.
- DIAG:SCSI:DISK - checks the operation of a disk drive at the address specified.

For more information on these commands see “SCPI Command Reference” starting on page 199.

To Troubleshoot the VT2216A

1. Type the following in an MS-DOS window then press enter:

   ```
   diskdiag
   ```

2. A response similar to the following identifies the module:

   ```
   diskdiag: running diagnostics for N2216 and E1562

   Sending:  *idn?
   Received:
   HEWLETT-PACKARD,N2216A (E1562E),US40200101,A.01.02
   ```

Note

This response will vary with the revision and date of the firmware used. A VXI Technology response will return a "VT" prefix (e.g. "VT2216A") while an HP response returns an "N" prefix (e.g. "N2216"). Both are acceptible.

   If the response is incorrect, use the resource manager to verify connection to the module. The resource manager must be running before the test is started.

3. A response similar to the following indicates that the self test passed:

   ```
   Sending:  *tst?
   Received:
   +0
   ```

   If the response is 1 through 11, the Main assembly is probably faulty.

   If the response is 12 through 14, the SCSI assembly is probably faulty.

4. A response similar to the following indicates that the Main assembly test passed:

   ```
   Sending:  diag:boar:main?
   Received:
   "E1562 version ?2, logical address 144, bus request level 3
   test_shared_ram: passed
   Test magic shared ram read: passed
   Test magic VXI a24 read: passed
   Passed
   "
   ```

   If this test failed, the Main assembly is probably faulty. If the Main assembly is replaced, the module's serial number and model number must be reprogramed into Flash ROM. See "To Reprogram the Main Assembly" on page 48.

5. A response similar to the following indicates that the SCSI assembly test passed:

   ```
   Sending:  diag:boar:scsi?
   Received:
   "Passed
   "
   ```

   If this test failed, the SCSI assembly is probably faulty.

6. A response similar to the following identifies the SCSI address of the device (in this case 00) followed by the type of device (in this case a Seagate disk drive):

```
Sending:  diag:scsi:dev? a
Received:
"00:SEAGATE ST373307LW-0001"

Sending:  diag:scsi:dev? b
Received:
"00:SEAGATE ST373307LW-0001"
```

If either response is incorrect, the fuse on the SCSI assembly, the internal SCSI cable or SCSI device is probably faulty. Check fuses on the SCSI assembly (F600 and F650). Swap the internal cables and retest. If the same connection fails, the disk drive is probably faulty. If the other connection fails, the internal cable is probably faulty.

If both responses are incorrect, the Power Supply assembly is probably faulty.

7. For two optional disk drives—a response similar to the following indicates that the "A" disk drive test passed:

```
Sending:  diag:scsi:disk? a,0
Received:
"SCSI0 la00:  device open -- blocksize=512   #blocks=97693755
SCSI0 la00:  read serial number -- LQ20983100001022 42TM
SCSI0 la00:  read VPD page 81 -- unknown information
SCSI0 la00:  read VPD page c0 -- 528000291679167000100010000000000000000000000000000
SCSI0 la00:  read VPD page c1 -- 21199052899
SCSI0 la00:  read VPD page c2 -- unknown information
SCSI0 la00:  read VPD page c3 -- unknown information
SCSI0 la00:  read VPD page d1 -- unknown information
SCSI0 la00:  read VPD page d2 -- unknown information
SCSI0 la00:  Passed
"
```

If the test Failed, the disk drive assembly connected to the "A" connector is probably faulty.

8. For one or two optional disk drives—a response similar to the following indicates that the "B" disk drive test passed:

```
Sending:  diag:scsi:disk? b,0
Received:
"SCSI1 la00:  device open -- blocksize=512   #blocks=97693755
SCSI1 la00:  read serial number -- LQ075951000010121KK4
SCSI1 la00:  read VPD page 81 -- unknown information
SCSI1 la00:  read VPD page c0 -- 528000291679167000100010000000000000000000000000000
SCSI1 la00:  read VPD page c1 -- 20799052899
SCSI1 la00:  read VPD page c2 -- unknown information
SCSI1 la00:  read VPD page c3 -- unknown information
SCSI1 la00:  read VPD page d1 -- unknown information
SCSI1 la00:  read VPD page d2 -- unknown information
SCSI1 la00:  Passed
"
```

If the test Failed, the disk drive assembly connected to the "B" connector is probably faulty.

Note

For ordering and replacement procedures, see “Replacing Assemblies” starting on page 33.

The VT2216A does not require calibration.

Replacing Assemblies

Replaceable Parts

Replacement parts are listed in the following tables:

- Assemblies
- Front panel
- Cables

Ordering Information

To order VXI Technology parts, please contact a local VXI Technology Service Center.

Replaceable Parts Table

The replaceable parts table contains the following columns:

- Ref Des = The reference designator allows for identificatoin of the part using the illustration provided.
- VTI Part Number = The part number assigned by VXI Technology to the part.
- Qty = Total quantity in instrument.
- Description = Description of the part.
- Mfg Code = A code number that identifies a manufacturer of the part. See “CAGE Code Numbers” for the manufacturer’s name and address.
- Mfg Part Num = The part number assigned by the manufacturer to the part.

Caution

The module is static sensitive. Use the appropriate precautions when removing, handling and installing to avoid unnecessary damage.

CAGE Code Numbers

The following table provides the name and address for the manufacturers' CAGE code numbers (Mfr Code) listed in the replaceable parts tables.

Mfr No.	Mfr Name	Address
03LB1	VXI Technology, Inc.	Irinve, CA 92614-6509 U.S.A.
00779	Tyco Electronics Corp. / AMP Products	Middleton, PA 17057-3608 U.S.A.
05791	Lyn-Tron, Inc.	Spokane, WA 99224-9406 U.S.A.
1QDV5	3M Corporation	Austin, TX 78726-4530 U.S.A.
22670	GM Nameplate, Inc.	Seattle, WA 98199-2728 U.S.A.
30817	Laird Technologies	Delaware Water Gap, PA 18327-0000 U.S.A.
5U402	House of Metrics, Ltd.	Spring Valley, NY 10977-4909 U.S.A.
63994	E-A-R Specialty Composite (Aearo Co.)	Indianapolis, IN 46268-1650 U.S.A.
65867	Seagate Technology, Inc.	Scotts Valley, CA 95066-4544 U.S.A.
86928	Seastrom Manufacturing Co.	Twin Falls, ID 83301-8526 U.S.A.
92912	Bel Fuse, Inc.	Jersey City, NJ 07302-44496 U.S.A.
93907	Textron Inc., Camcar Division	Rockford, IL 61104-5159 U.S.A.

Assemblies: VT2216

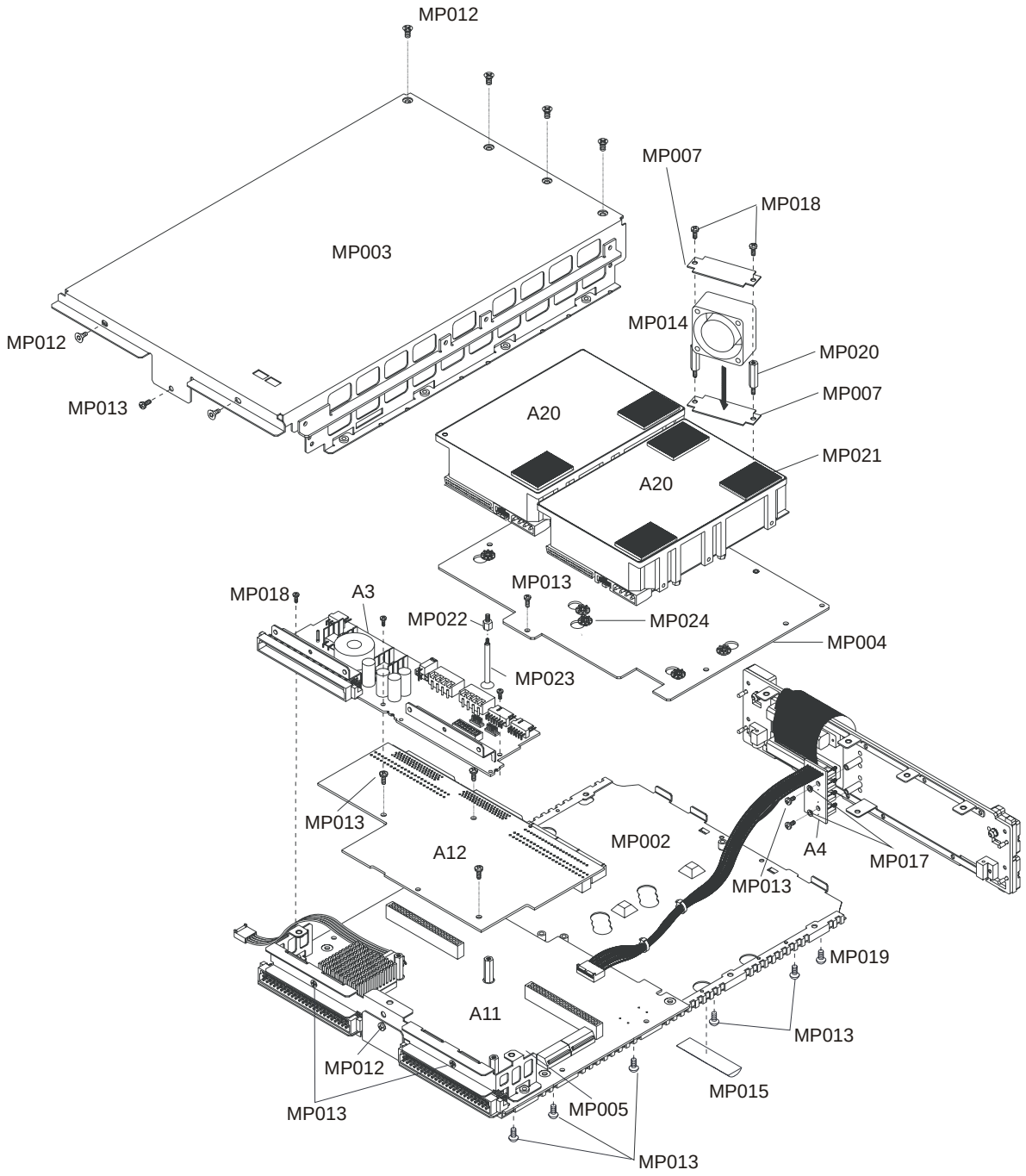


Table 1 Assemblies

Ref Des	VTI Part Number	Qty	Description	Mfr Code	Mfr Part Number
A3	N2216-66503	1	PC-ASSY DISK POWER SUPPLY	03LB1	N2216-66503
A4	N2216-66505	1	PC-ASSY LED	03LB1	N2216-66505
A11	N2216-66511	1	PC-ASSY MAIN	03LB1	N2216-66511
A12	N2216-66512	1	PC-ASSY SCSI INTERFACE	03LB1	N2216-66512
A20	0950-4201	2	OPTIONAL 73 GB DISK DRIVE	65867	ST373307LW-0001
MP012	0515-1946	8	SCREW-MACH M3 X 0.5 6MM-LG 90-DEG-FLH-HD	03LB1	0515-1946
MP013	0515-0372	21	SCREW-MACHINE ASSEMBLY M3 X 0.5 8MM-LG	03LB1	0515-0372
MP014	E1562-68501	1	FAN	03LB1	E1562-68501
MP015	8160-0686	1	RFI STRIP-FINGERS BE-CU SN-PL	30817	786-185
MP016	0380-2070	3	STANDOFF-HEX 14-MM-LG M3.0 X 0.5-THD	05791	SS5172-14.0-01
MP023	N2216-26001	8	SHLDR-SCREW	03LB1	N2216-26001
MP024	0400-0922	8	GROMMET-RND .158-IN-ID .057-IN-GRV-WD	63994	G-410-3
MP026	0515-0664	4	SCREW-MACHINE ASSEMBLY M3 X 0.5 12MM-LG	93907	0515-0664
MP027	0515-2733	4	SCREW-SPCL M2.5 X 0.45 17MM-LG PAN-HD	03LB1	0515-2733
MP028	0535-0031	2	NUT-HEX W/LKWR M3 X 0.5 2.4MM-THK	5U402	0535-0031

Note

If the Main assembly is replaced, the module's serial number and model number must be reprogramed into Flash ROM. See "To Reprogram the Main Assembly" on page 48.

Early VT2216A modules contain the N2216-66501 Main assembly. The N2216-66511 Main assembly is the replacement assembly for the N2216-66501.

Front Panel

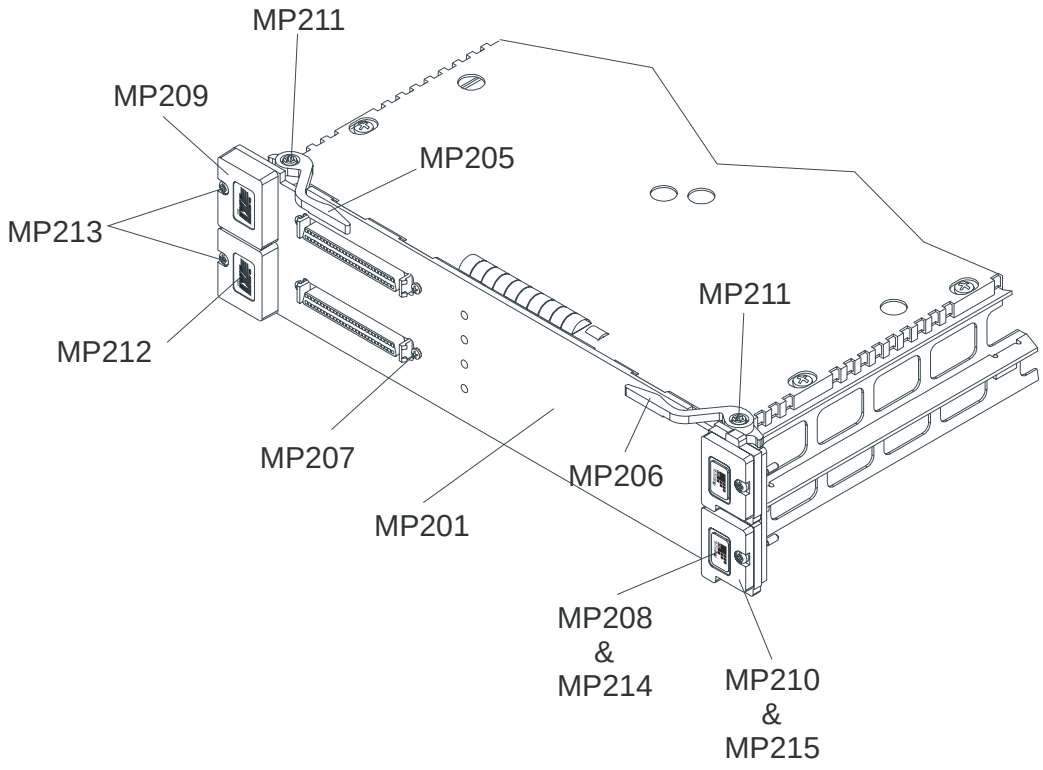


Table 2 Front Panel

Ref Des	VTI Part Number	Qty	Description	Mfr Code	Mfr Part Number
MP201	N2216-00201	1	FRONT PANEL	03LB1	N2216-00201
MP205	E1400-45101	1	MOLD KIT-BTTM EXTR HDL	03LB1	E1400-45101
MP206	E1400-45102	1	MOLD KIT-TOP EXTR HDL	03LB1	E1400-45102
MP207	1252-6155	4	SCREWLOCK FEMALE-SUBMIN D CONN	00779	786585-2
MP208	7121-7893	1	PLT-NAME,'SPARK'	03LB1	7121-7893
MP214	43-0016-003	2	LABEL, VXI EXT, 'VXI TECH', NEW LOGO	03LB1	43-0016-003
MP215	43-0016-002	2	LABEL, VXI EXT, 'VXIBUS'	03LB1	43-0016-002
	E1400-40104	2	L-BLOCK	03LB1	E1400-40104
	0515-0664	4	SCREW-MACHINE ASSEMBLY M3 X 0.5 12MM-LG	93907	0515-0664
	0535-0031	2	NUT-HEX W/LKWR M3 X 0.5 2.4MM-THK	5U402	0535-0031

Cables

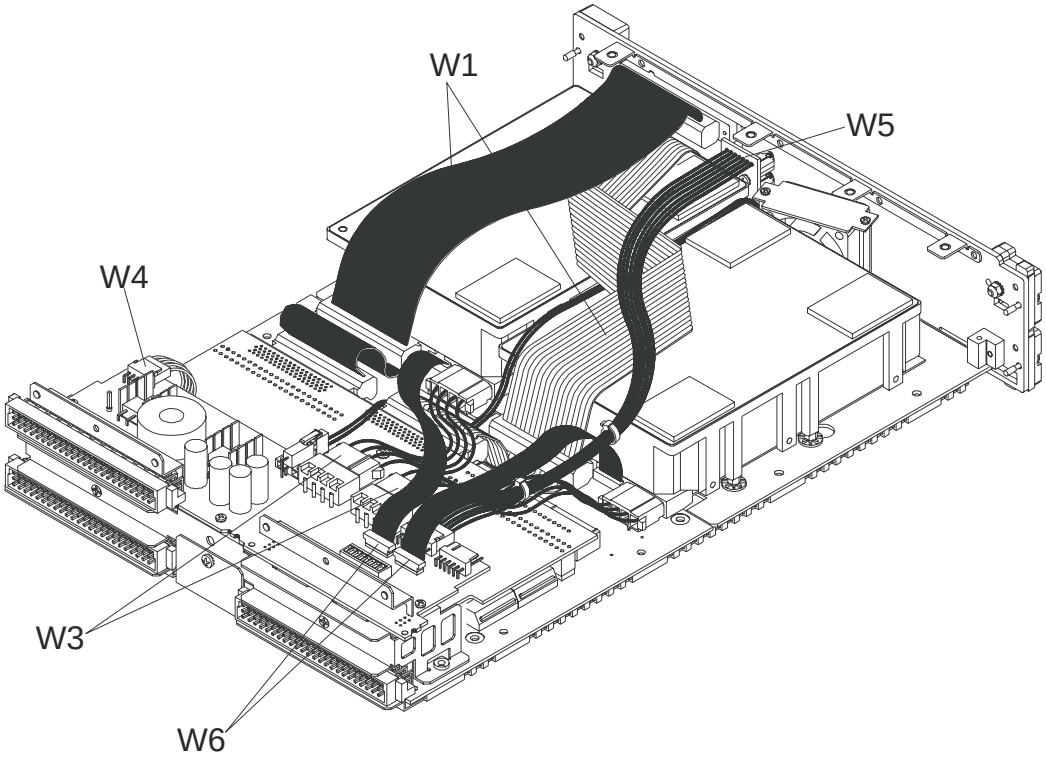
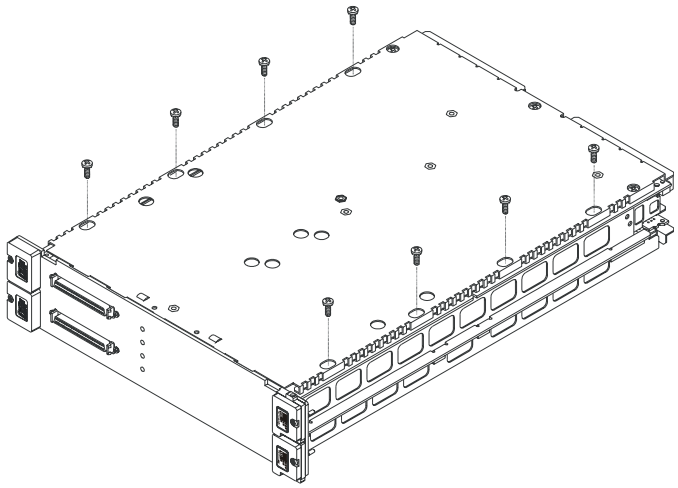


Table 3 Cables

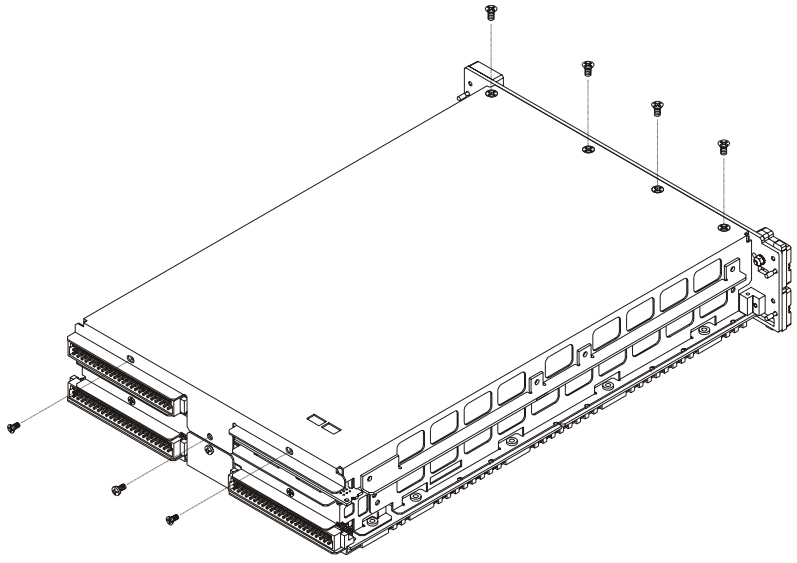
Ref Des	VTI Part Number	Qty	Description	Mfr Code	Mfr Part Number
W1	E1562-61601	2	CBL-SCSI	03LB1	E1562-61601
W3	E1562-61603	2	OPTIONAL CBL-ASM DSC DISK	03LB1	E1562-61603
W4	E1562-61604	1	CBL-AUX	03LB1	E1562-61604
W5	N2216-61605	1	CBL-CTRL BRD TO LED BRD	03LB1	N2216-61605
W6	E1562-61606	2	OPTIONAL CBL-ASM RBN DISK	03LB1	E1562-61606

To Remove the Top Cover

1 Using a T-10 Torx driver, remove the four screws along each side of the bottom cover.

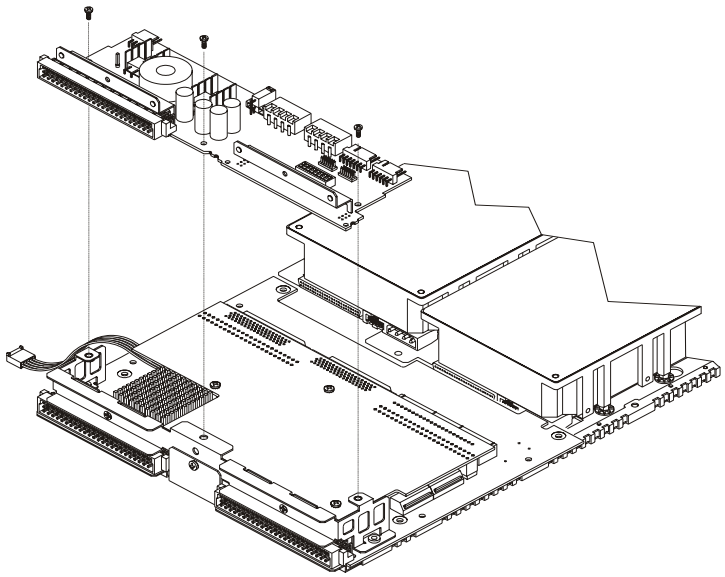


2 Using a T-10 Torx driver, remove the four screws along the top and the three screws along the back of the top cover.

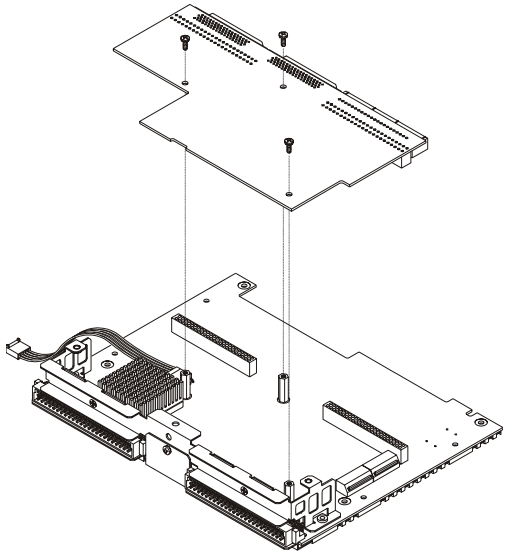


To Remove the Printed Circuit Assemblies

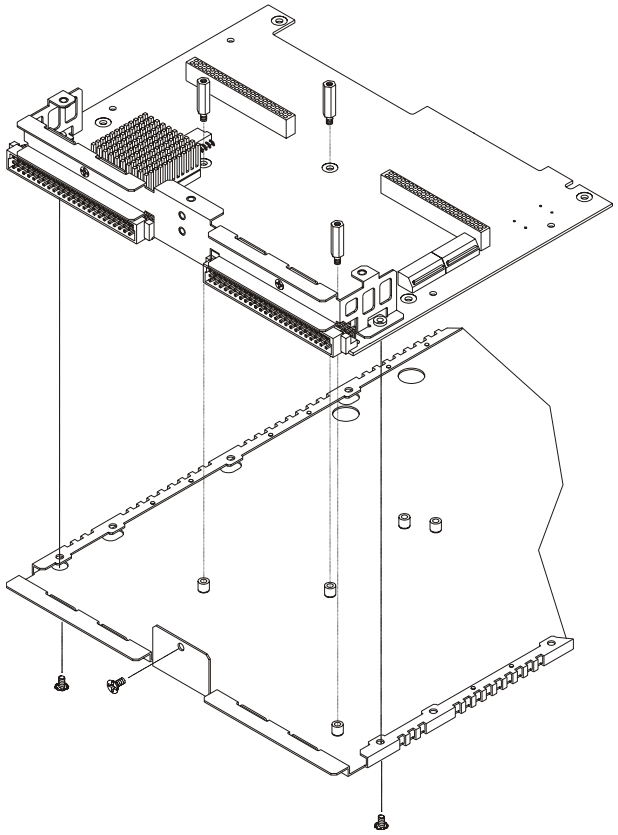
1 Remove top cover. Disconnect all cables from the printed circuit boards. Using a T-10 Torx driver, remove the three screws and the Disk Power Supply assembly.



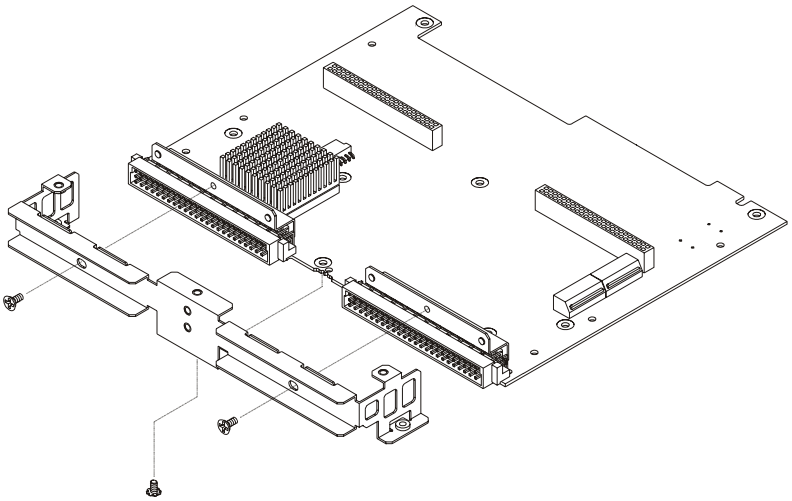
2 Using a T-10 Torx driver, remove the three screws on the SCSI Interface assembly. Carefully, to avoid stressing solder joints, pry the boards apart at the connectors and remove the assembly.



3 Using a T-10 Torx driver, remove the three screws from rear and sides of bottom cover. Using a 6 mm nut driver, remove the three standoffs and the Main assembly.

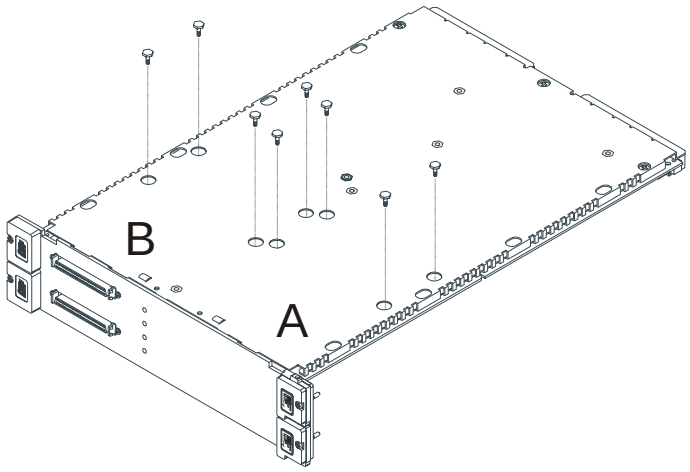


4 Using a T-10 Torx driver, remove the three screws and the rear cover from the Main assembly.



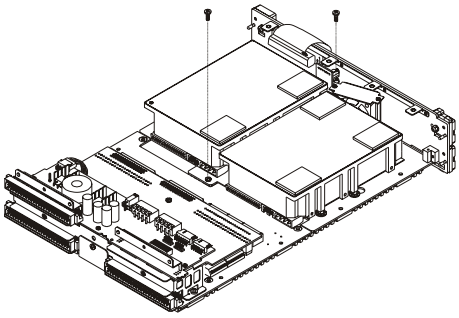
To Remove a Disk Drive

1 Remove top cover. Disconnect cables from the disk drive and move out of the way. While supporting the disk drive (A or B), remove the four shoulder srews using a 5/16 inch nut driver. If the disk drive is being replaced, remove the standoffs from the old disk drive and attach to the new disk drive.

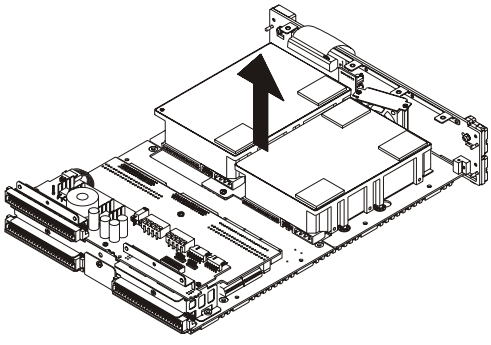


Note: If the nut driver does not fit through the access holes or other difficulty is encountered, remove the disk drive mounting plate then remove the disk drive. See the following steps.

1 Using a T-10 Torx driver, remove the two mounting plate screws.



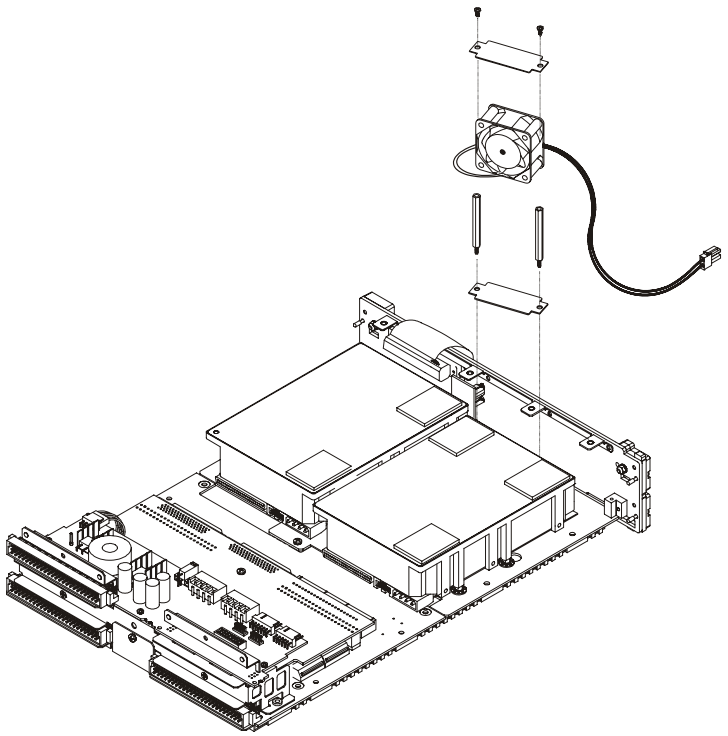
2 From the back of the assembly lift up and back to remove.



To Remove the Fan

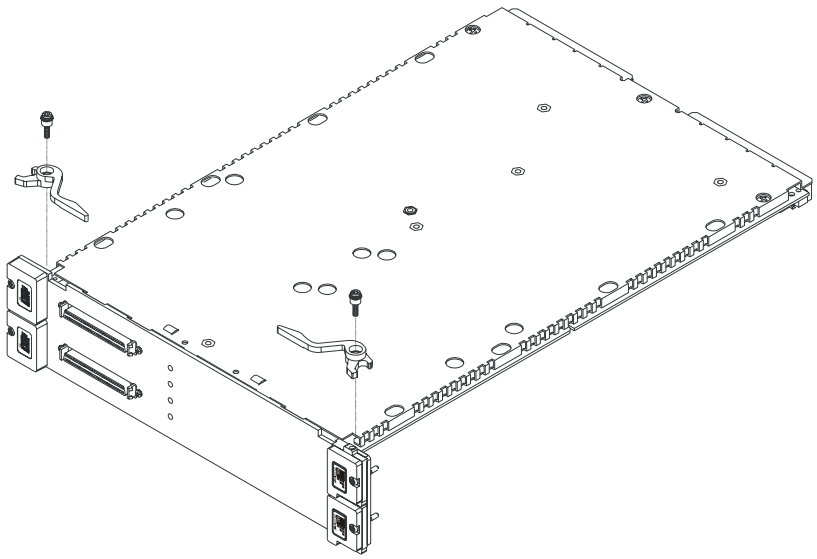
1 Remove top cover. Disconnect the cable from the LED assembly and move out of the way. Disconnect the fan cable.

2 Using a T-10 Torx driver, remove two screws then lift off fan plate. Using a 4.5 mm nut driver, remove the two standoffs then remove the fan and fan mounting plate.

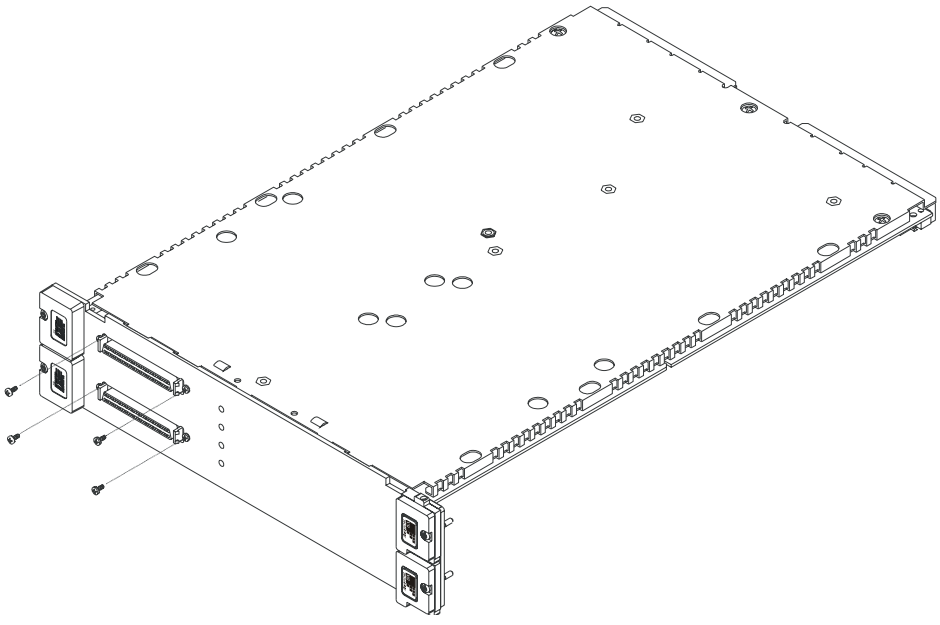


To Remove the Front Panel

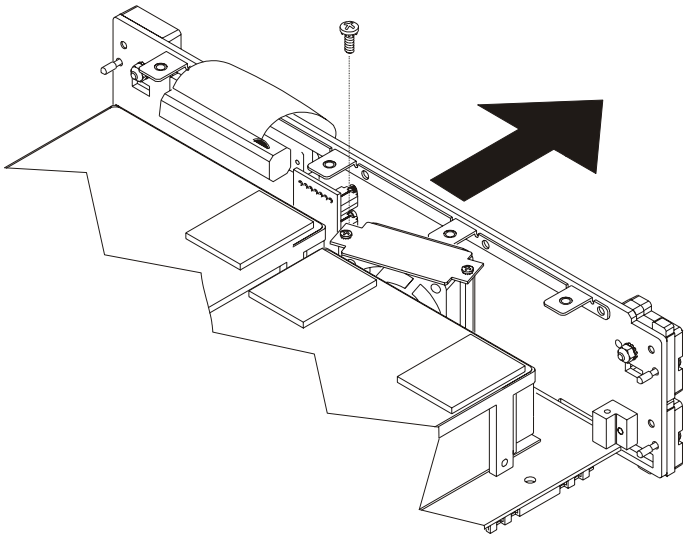
1 Remove top cover. Using a T-8 Torx driver, remove the two screws that attach the handles to the assembly.



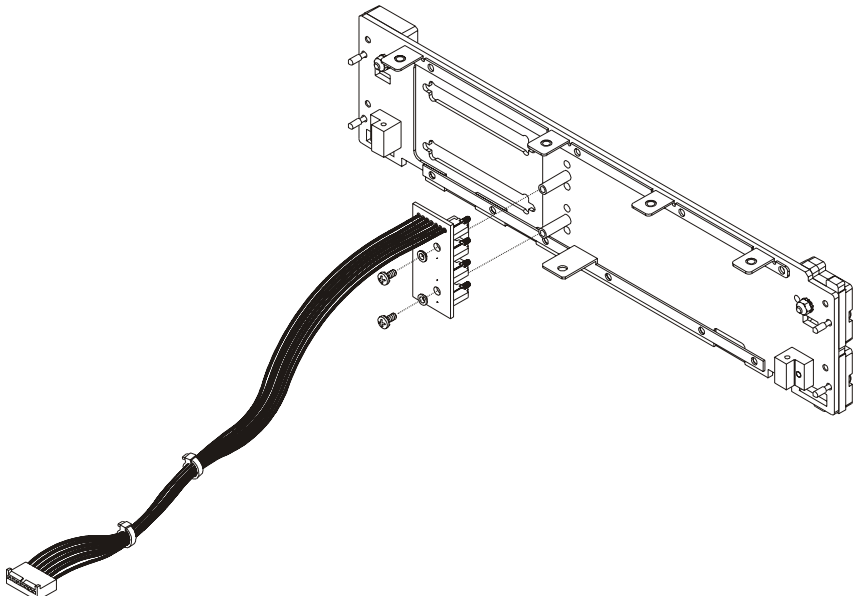
2 Using a thin flat-blade screwdriver, remove the screws that attach the connectors to the front panel. Be careful to avoid scratching the front panel.



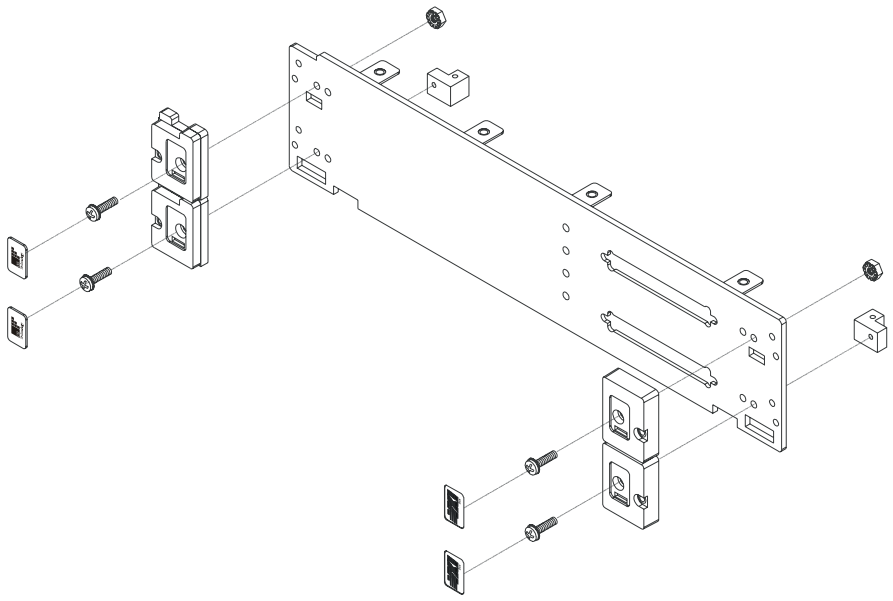
3 Using a T-10 Torx driver, remove the screw that attaches the front panel to the bottom cover. Lift the lower part of the front panel off of the tabs on the bottom cover and slide it away from the module.



4 Using a T-10 Torx driver, remove the two screws and the LED board from the front panel.



5 If the front panel is being replaced with another that does not have its own L-blocks, remove the L-blocks from the old front panel and attach to the new front panel using a T-10 Torx driver. Because alignment is critical, be sure to note the positioning of the brackets. To access the screws, remove the name plates by pushing a thin screwdriver through the slot on the back side of the front panel.



To Reprogram the Main Assembly

If the Main assembly is replaced, the module's serial number and model number must be reprogramed into Flash ROM.

Using a Windows NT 4.0 or Windows 2000 and later operating system

1. At an MS-DOS command prompt (C:\>), type the following replacing AAnnnnnnnn with the module's serial number then press enter:

   ```
   diskcmd "DIAG:MNSN ""N2216A" "(E1562E)""","""AAnnnnnnnn"""
   ```

2. Wait two minutes for the command to process, then type the following:

   ```
   diskcmd -r "DIAG:SNUMBER?"
   ```

3. Verify that the correct serial number was returned.
4. Type the following:

   ```
   diskcmd -r "DIAG:MName?"
   ```

5. Verify that "N2216A (E1562E)" was returned.

Using an HP-UX 10.2 operating system

1. Using a K shell, type the following replacing AAnnnnnnnn with the module's serial number:

   ```
   diskcmd "DIAG:MNSN \"N2216A (E1562E)\",\"AAnnnnnnnn\""
   ```

Note

If using an SICL only system, use diskcmd_s instead of diskcmd.

2. Wait two minutes for the command to process, then type the following:

   ```
   diskcmd -r "DIAG:SNUMBER?"
   ```

3. Verify that the correct serial number was returned.
4. Type the following:

   ```
   diskcmd -r "DIAG:MNAME?"
   ```

5. Verify that "N2216A (E1562E)" was returned.

Hardware Description

General Description

The VT2216A VXI/SCSI Interface module is a high-speed dual SCSI interface with optional internal disk drives. Option 1 adds a 73 GB drive and Option 2 adds two 73 GB drives. The VT2216A is compatible with software written for the Agilent/HP E1562. However, its SCSI electrical interface is not compatible with the Agilent/HP E1562.

Caution

Do not connect high-voltage differential (HVD) or fast-wide differential devices to the module's SCSI connectors. The VT2216A contains low-voltage differential (LVD) circuits that may be damaged if connected to HVD circuits.

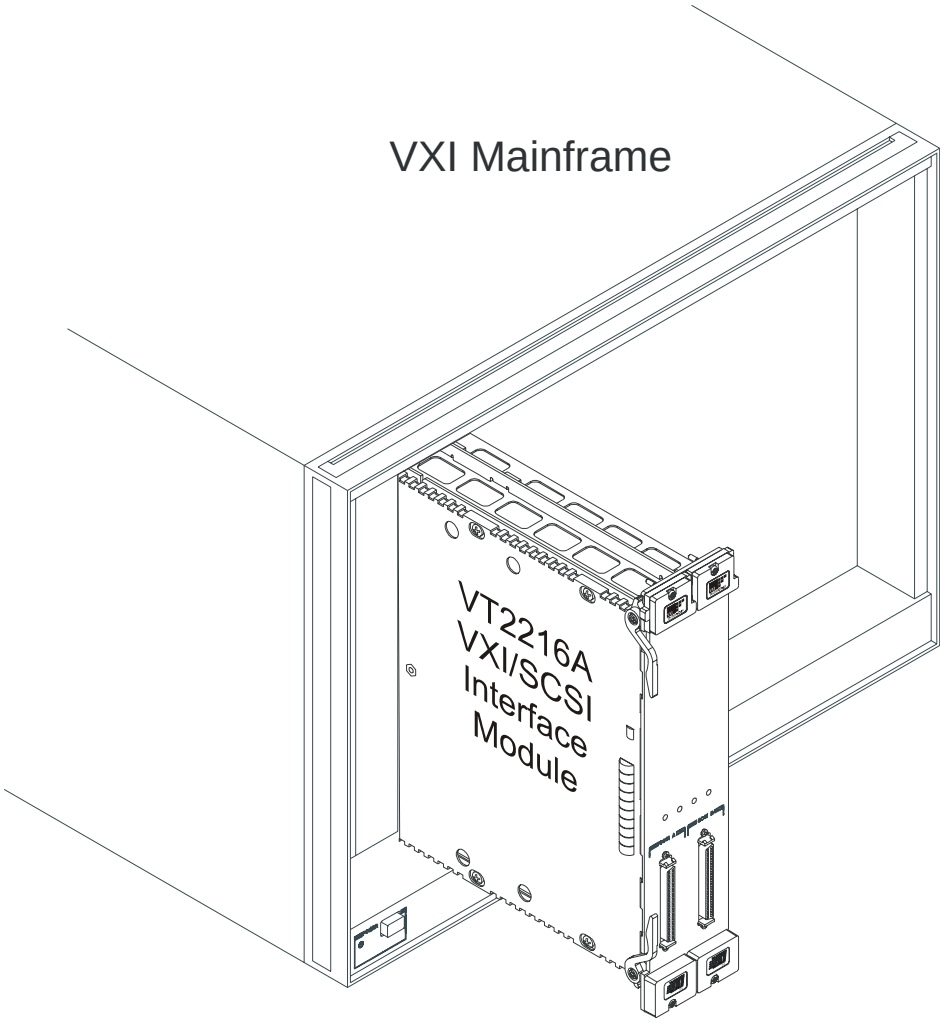


Figure 1 VT2216A VXI/SCSI Interface Module

The VT2216A occupies two slots in a C-size VXI mainframe and does not require calibration.

The VT2216A can be used for local bus disk throughput from input modules such as the VT1432A 16-channel Digitizer and the Agilent/HP E1430A Digitizer.

The VT2216A can also be used for throughput from high channel count applications using the VT1413C 64-channel Input, which does not have a local bus interface. In this case, the VT2216A orchestrates high-speed throughput transfers without the interaction of a controller. For high-speed applications, the VT2216A Option 2 (with two internal disk drives) is required.

In Static+Dynamic applications, the VT2216A can store data from both local-bus-based modules and non-local-bus-based modules simultaneously.

The cache size for the optional disks in the VT2216A is 1 megabytes (MB).

The Local Bus is a flexible daisy-chain structure connecting the modules of a VXI system. It is twelve lines wide in each direction through the P2 connector and an additional 24 lines wide though the P1 connector.

Circuit Description

A block diagram for the VT2216A is shown below.

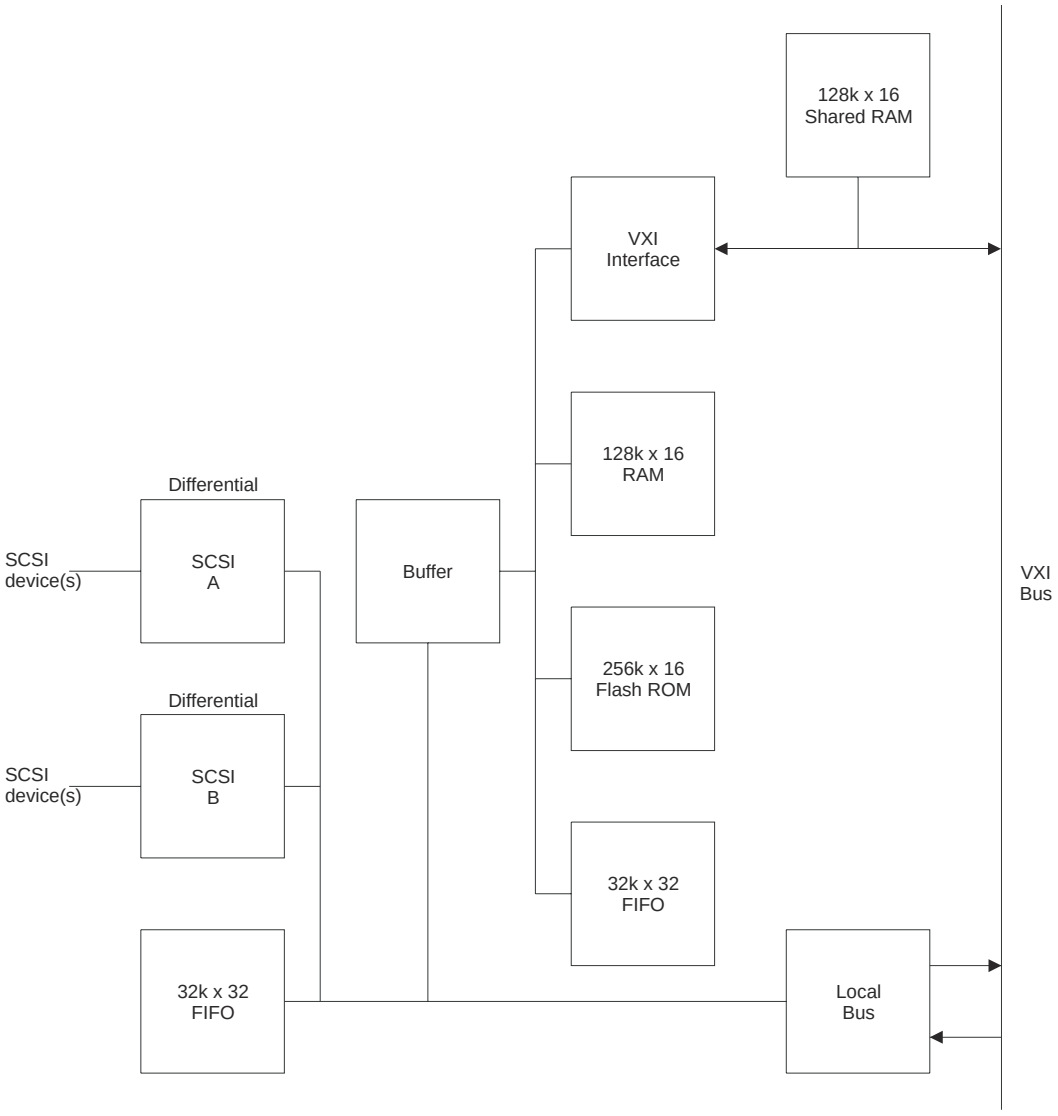


Figure 2 Block diagram of the VT2216A

The following is a simplified block diagram of the VT2216A.

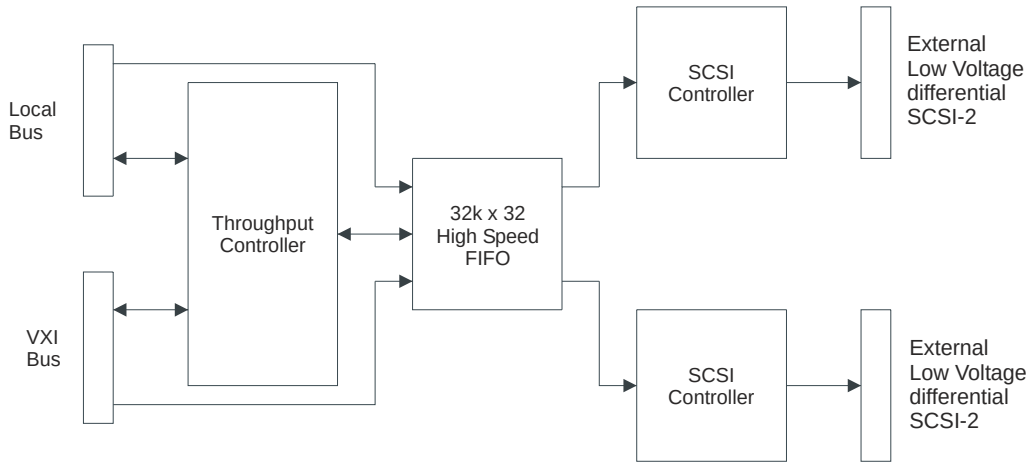


Figure 3 Block diagram of the VT2216A

The following is a simplified block diagram of the VT2216A Option 1. Option 1 adds a 73 GB disk drive to channel B in the standard VT2216A.

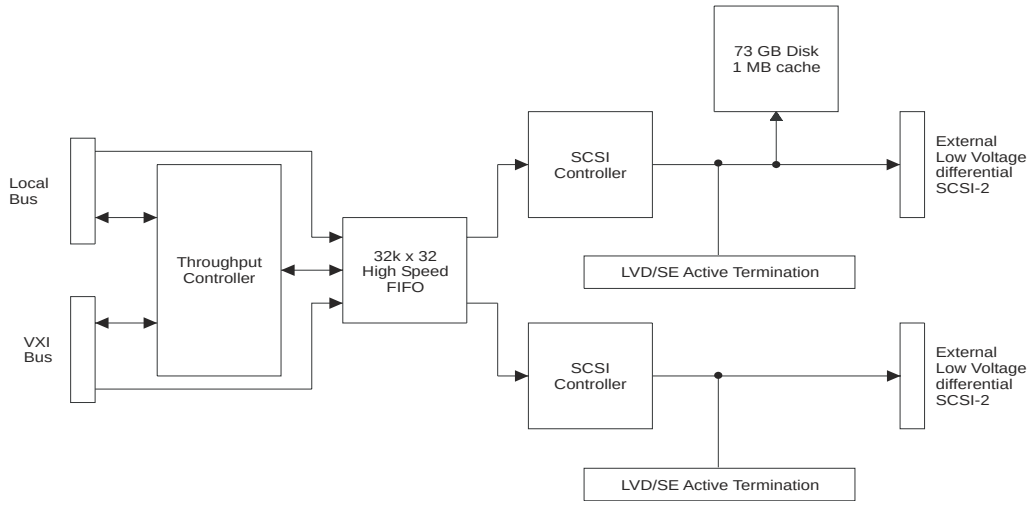


Figure 4 Block diagram of the VT2216A Option 1

The following is a simplified block diagram of the VT2216A Option 2. Option 2 adds two 73 GB disk drives to the standard VT2216A.

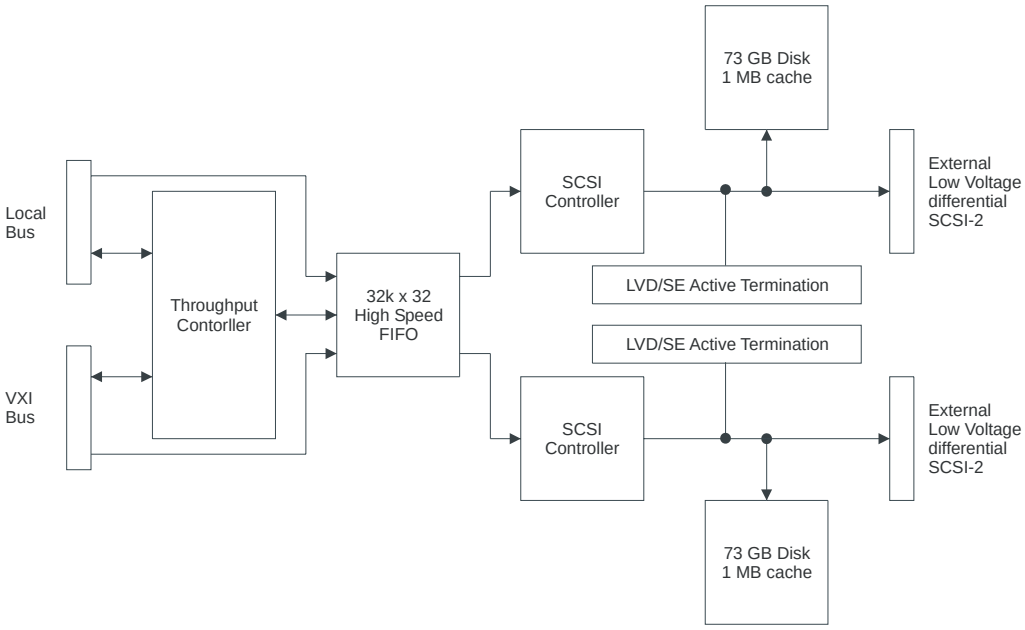


Figure 5 Block diagram of the VT2216A Option 2

VT2216A Front Panel Description

Status LEDs

- Failed
- Access
- Disk A (only lit during disk access on VT2216A's with Option 2)
- Disk B (only lit during disk access on VT2216A's with Option 1 or 2)

The Disk LED lights when the corresponding disk is in use.

SCSI connectors

The VT2216A has two multi-mode—low voltage differential (LVD) and single ended (SE)—SCSI connectors (SCSI A and SCSI B). The VT2216A uses both interfaces to increase the overall transfer rate.

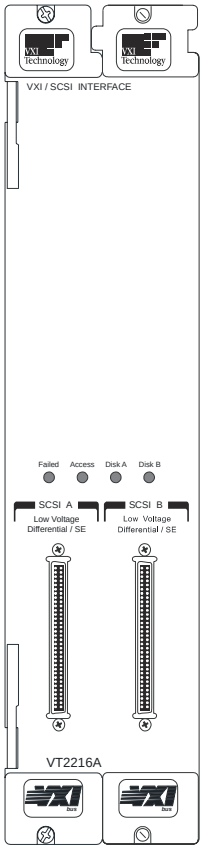


Figure 6 VT2216A

Using the VT2216A

VXI and SCPI

Message-based VXI devices

The VT2216A is a message-based VXI module. A message-based device is typically the most intelligent device of a VXIbus system. High performance instruments are typically available as message-based devices. Besides the basic configuration registers supported by the register-based devices, the message-based device has common communication elements and a Word Serial Protocol to allow ASCII-level communication with other message-based modules. This allows easier multi-manufacturer support, though at some sacrifice in speed to interpret the ASCII messages. Since the Word Serial Protocol mandates only a byte transfer per transition, which then must be interpreted by the onboard micro-processor, message-based devices are typically limited to IEEE-488 speeds.

SCPI

In the past, system instruments spoke many different languages. This caused test system developers to spend valuable time learning instrument control languages. Test programs written using these languages were hard to modify and the substitution of one instrument with another was nearly impossible.

SCPI (Standard Commands for Programmable Instruments) is a closely defined, but broadly accepted, standard instrument command language. SCPI has the advantage that test programmers need to learn only one language. Also, test programs written in SCPI can be easily understood and easily modified and test systems can be easily upgraded.

The VXI Registers

The VT2216A is a message-based VXI device and cannot be programmed by way of registers like a register-based device. However, it does use the following VXI registers:

- Offset Register
- Status/Control Register
- Device Type Register
- ID/Logical Address Register
- Data Low Register
- Response/Data Extended Register
- Protocol/Signal Register

These registers are common to many VXI devices. Refer to VXI documentation for more information.

Throughput Terminology

SCPI Commands

The following is an overview of the some of the capabilities of the VT2216A that are controlled by SCPI commands. See “Programming Using SCPI” starting on page 185 and “SCPI Command Reference” starting on page 199 for details.

One group of commands begins with the command MMEMory. These are all commands that refer to mass storage capabilities. They are VT2216A defined commands and not part of the SCPI standard.

The process by which the VT2216A transfers data can be organized as shown in the following illustrations. Data for the individual devices is organized into *Transfer Units* (TUNITs) and *Sessions* that are controlled by *Sequences*. The following sections explain more about these terms.

Individual SCSI Devices

The VT2216A provides two SCSI buses with a controller on each bus. The controller’s SCSI logical address is set by switches to SCSI address 4, 5, 6, or 7, but may be changed via the SYST:COMM:SCSI:SELF:ADDR command in case of an address conflict.

The MMEMory:SCSI subsystem provides a means of initializing and controlling a single SCSI device. Special configurations are set up using this subsystem. Higher level data Transfer Units are built using this lowest level entity — an individual device.

The subsystem that refers to devices is MMEMory:SCSI[1|2|...|30]. These are commands that refer to individual devices on a SCSI bus.

All commands in this subsystem refer to a single device. Commands are provided to open and close these devices as well as configure special aspects of these devices. Data reads and writes are *not* done using the MMEMory:SCSI commands. See the MMEMory:TUNit and MMEMory:SESSion commands for further explanation about how to completely configure the SCSI system on a VT2216A.

The MMEMory:SCSI*x*:* commands do *not* refer to a specific SCSI device depending upon the value of *x*. Instead, the SCSI controller, logical address and logical unit are specified in the MMEMory:SCSI*x*:OPEN command. The letter *x* just provides a convenient means to refer to one of several open SCSI devices.

See “SCPI Command Reference” starting on page 199 for more information.

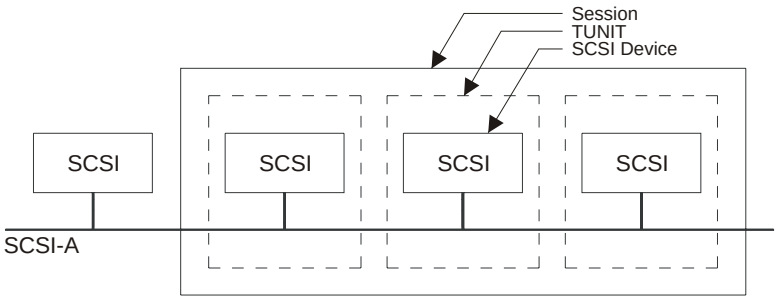


Figure 7 An example of SCSI devices, Sessions, and TUNITs

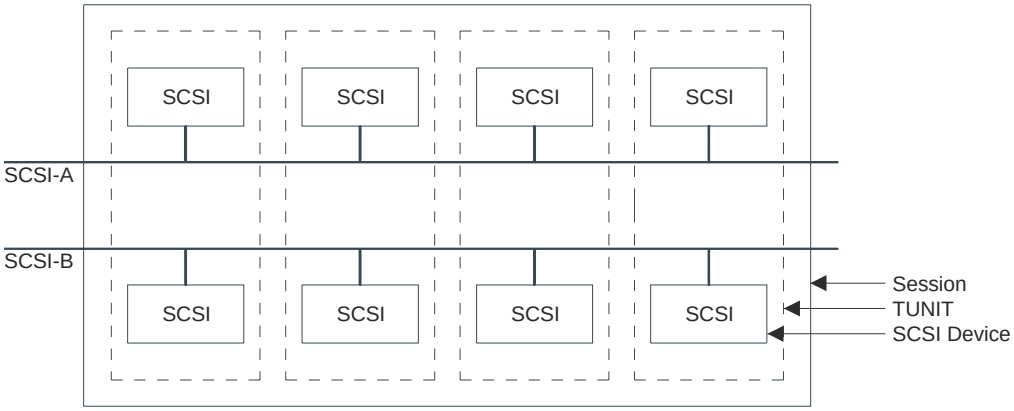


Figure 8 An example of SCSI devices, Sessions, and TUNITs

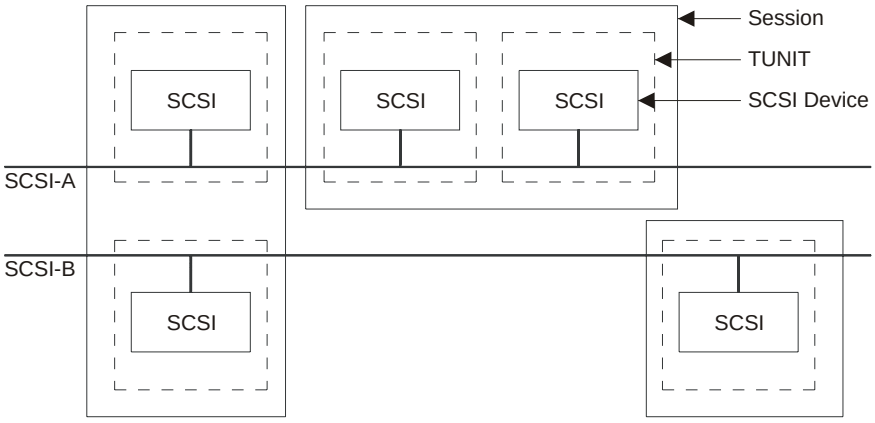


Figure 9 An example of SCSI devices, Session, and TUNITs

Transfer Unit (TUNIT)

A Transfer Unit can refer to data from either one or two devices.

The subsystem that refers to Transfer Units is MMEMory:TUNit[1|2|...|15]. These are commands that refer to a simultaneous data transfer.

Since the VT2216A provides a pair of SCSI buses and may contain two internal devices, a means is provided to send data to this pair of devices (one on each controller). The MMEMory:TUNit subsystem informs the VT2216A whether it will be transferring data from just a single device or a pair of devices. When data is sent to a pair of devices, the throughput rate is twice that of a single device, but the data is "split" between the devices — two bytes to one device and two bytes to the other device. Special internal hardware makes it possible for these four bytes to be transferred at the same time for the highest possible throughput rate. In order to transfer data to/from a pair of devices, it is necessary for the SCSI blocksize to be the same for the two devices.

This subsystem refers to a continuous sequential stream of data. The name "TUNIT" means Transfer UNIT. This data may be transferring to/from a single SCSI device. Or for the VT2216A, a TUNIT may refer to data that is split across devices in such a way as to make the upper 16 bits of a 32 bit quantity go to one device and the lower 16 bits of the quantity go to another device. This type of data split requires that the two devices be on different SCSI controllers. This subsystem does *not* refer to data that is split in terms of blocks such that N logical blocks reside on device 1 and N blocks reside on device 2 or some more complicated scheme. See the MMEMory:SESSion commands for data that is split in this manner.

This subsystem was introduced to describe data split across the two SCSI controllers supported by the VT2216A. It is also a core element in creating a Session (see the MMEMory:SESSion subsystem).

See "SCPI Command Reference" starting on page 199 for more information.

Sessions and Striping

A Session provides the ability to combine one or more Transfer Units together into one logical data repository.

The subsystem that refers to Sessions is MMEMory:SESSion[1|2|3|4]. These are commands that refer to a complete repository of data.

The reason for using more than one Transfer Unit is to use more of the overall SCSI bandwidth by writing enough data to one (pair of) disk to fill up its cache, then switching to another (pair of) disk while the first one writes its data to its media. In this manner, several disks can be supported on each SCSI bus which increases the overall SCSI throughput. The MMEMory:SESSion subsystem is the main point of interaction when reading and writing data — it makes the number of disks involved in the data transfer transparent.

This subsystem describes how data is divided between one or more Transfer Units. Sessions using multiple Transfer Units will contain data that has N blocks on Transfer Unit 1, M blocks on Transfer Unit 2 and so on. This is called disk *striping*. Where a *TUNIT* describes a width-wise split, a *Session* describes a length-wise split. Sessions are useful in high-speed throughputs as a means of keeping several slower devices busy at the same time. For instance, the wide SCSI bus has a maximum data transfer rate of 20 MB per second. However, most disks have a maximum continuous transfer rate to media of ≈15 MB per second. It is easy to see that by using the cache on the disk, data split across several disks could attain a higher overall throughput than data written to a single disk.

Disk striping can also be used to optimize disk storage. For an example, see Figure 10. Using only one pair of SCSI devices would allow 100 GB of storage. Striping allows the data to be spread across two pairs of devices for a total of 200 GB.

In most cases, throughputs and playbacks require both a Sequence and a Session. A Session is required for all reads and writes including throughputs and playbacks. However, a simple throughput from a single non-LBUS device or a non-LBUS playback can be done without a Sequence.

During a throughput Sequence using multiple Transfer Units, each Transfer Unit will have a specified number of logical blocks written to it before switching to the next Transfer Unit in the Session. When the last Transfer Unit in the Session has completed its set of logical blocks, the first Transfer Unit is again accessed.

There are some constraints upon Sessions that are difficult to describe in the individual command descriptions. The first constraint is that every Transfer Unit in a Session must have the same number of SCSI devices in it. The second constraint is that if each Transfer Unit is made up of only a single device, each device must be on the same SCSI controller. An error will be returned from the MMEMory:SESSion:ADD command if these constraints are not followed.

It is also important to know the cache size of the disks the data is being written to. For the optional disks in the VT2216A, the cache size is 1 MB. This means that the parameter <Count> in the MMEMory:SESSion:ADD command should be no more than 2048 blocks. A count larger than this would require that the disk be read more often and slow down the data transfer.

See “SCPI Command Reference” starting on page 199 for more information.

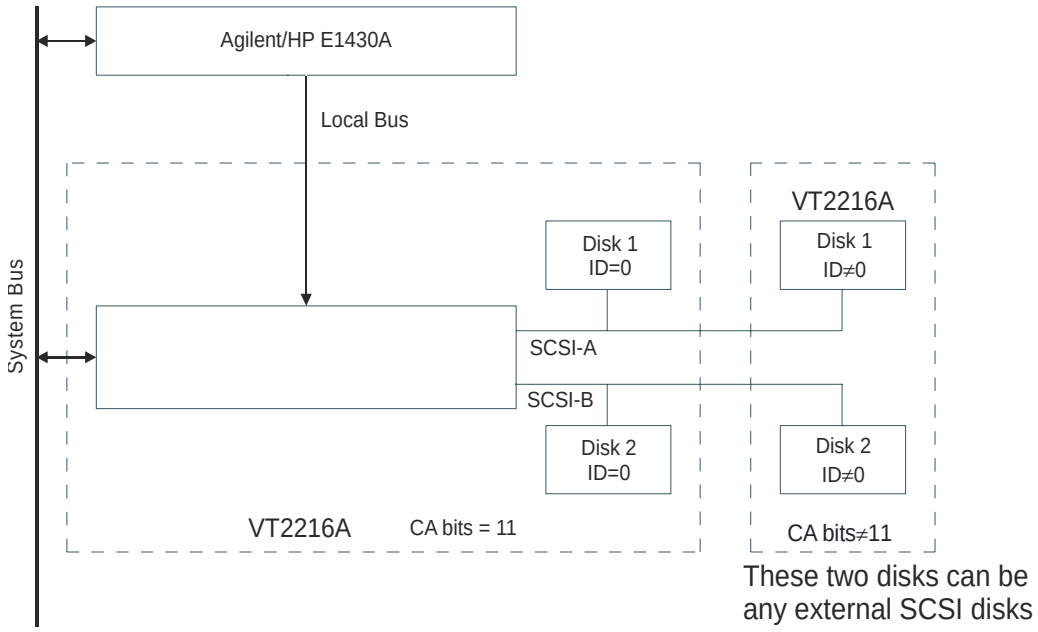


Figure 10 Example of Disk Striping - Schematic View

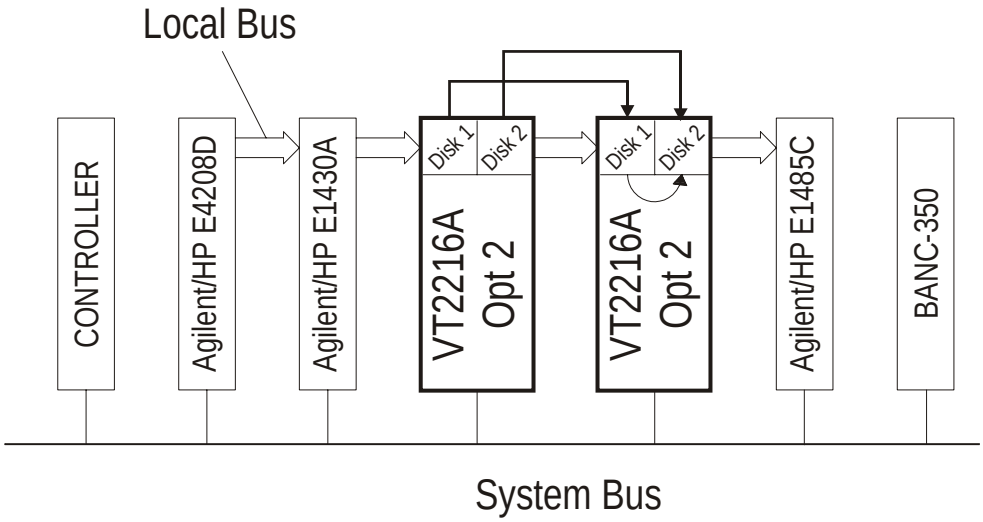


Figure 11 Example of Disk Striping - View Representing Modules in Mainframe

Sequence

A Sequence specifies the order of operations for a throughput or playback Session.

The subsystem that refers to Sequences is SEQuence[1|2|3|4]. These are commands that provide a means to specify a complex throughput or playback.

The SEQuence subsystem is used to specify the order of operations for a throughput or playback Session. This list of operations may contain data transfer requests from both the local bus and the VME bus in throughput Sequences, but may contain only VME or LBUS data transfers upon playback. Synchronization and control operations are provided for both throughput and playback.

The fields contained in every element of the Sequence list are: operation, count, address and miscellaneous. The operation field specifies the action to be done: data transfer, synchronization or control. The count field is used by many operations to indicate how many operation units will be transferred. The unit of count is sometimes bytes and sometimes blocks — see the description of the operation to determine which. The address field is used by operations that do VME data transfers. The value of address is an offset from the beginning of one of the address spaces. The miscellaneous field has various meanings depending upon the operation. Not every operation uses all fields, but every Sequence element contains all four fields. Fields that are not used should be set to zero.

A Session must be initialized before starting a Sequence. See the MMEMory:SESSion subsystem.

The behavior of a Sequence is undefined if a throughput operation is requested in a playback Sequence or vice versa. It is also undefined if an LBUS playback operation is included in a VME playback Sequence.

SEQuence is *not* a SCPI supported subsystem.

The Sequence operations that are labeled as utility may be used in either playback or throughput Sequences. They are intended to help provide synchronization between the Sequence and the devices that are generating/receiving the data.

See “Sequence Operations Reference” starting on page 141 for details on using Sequence commands.

Operation Status Register

The subsystem that refers to the operation status register is STATus:OPERation. These are commands that provide the necessary commands to interface with the operation register.

For more information about the operation status register and other status registers, see “Programming Using SCPI” starting on page 185.

LIF Directories and Files

This diagram represents the way files are laid out on a disk using the LIF format (Logical Interchange Format). The first field, the volume label, references the directory that follows. The directory contains a number of entries each of which references one of the user files, which are on the remainder of the disk.

For more information about LIF functions see “LIF Library Reference” starting on page 277.

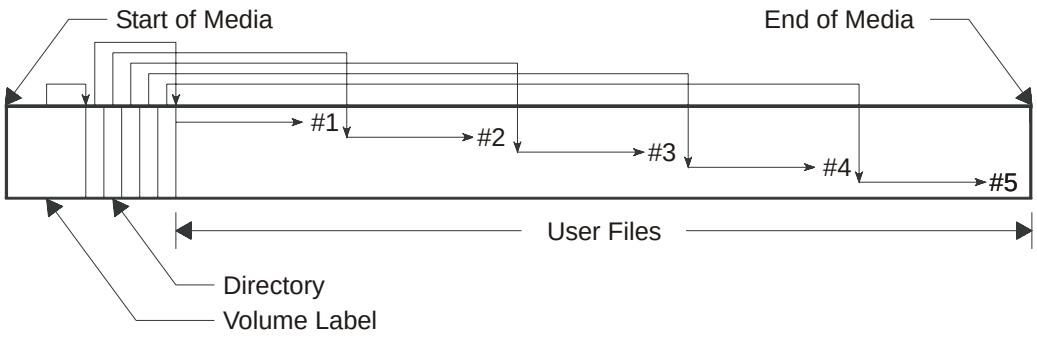


Figure 12 Logical Interchange Format (LIF) - Media Layout

Address Space

The VXI system architecture defines three types of address space. A16 space consists of 64 kB, A24 is 16 MB, and A32 is 4 GB.

The VT2216A has access to A16, A24, and A32 space through a 16-bit port. Or, if devices support it, it can also use a 32-bit port using D32. The type of VME cycle performed depends on the type of processor cycle (two cycles for 16-bit or one cycle for 32-bit).

Shared Memory

Shared memory provides a way for the VT2216A to transfer data to a controller. The shared memory in the VT2216A is mapped to the A24 VXI address space. The controller can then access that same address space to receive or write data. Note that if SCPI commands or Sequences refer to shared memory in the VT2216A, the addresses begin at zero. However, if they refer to shared memory in the A24 space, they may begin at a different value, depending on how the A24 memory has been allocated among devices.

TTLTRG

TTLTRG consist of eight lines on the VXI backplane on connector P2. They are available to provide synchronization between devices. The VT2216A can use the TTLTRG lines for simple communication with other devices. For example, it can wait for a line to go high before taking an action or it can assert a line as a signal to another device.

The VT2216A Throughput/Playback Process

Acquisition

Local Bus

The following illustration shows the VT2216A acquiring 24 channels of dynamic data over the local bus. Each VT1433B module takes in eight channels of data and sends it to the VT2216A over the local bus. In this example, the VT2216A Option 2 places the data in its two disk drives.

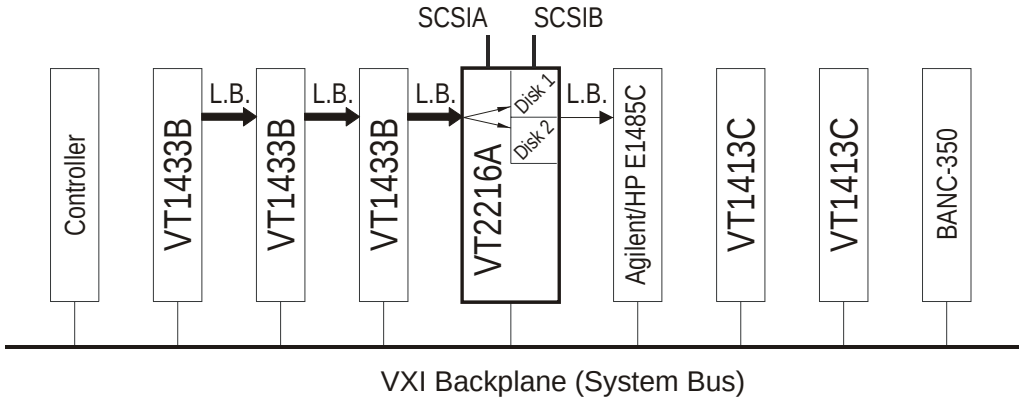


Figure 13 Data Acquisition Using the Local Bus

System Bus

The following illustration shows the VT2216A acquiring data over the System Bus.

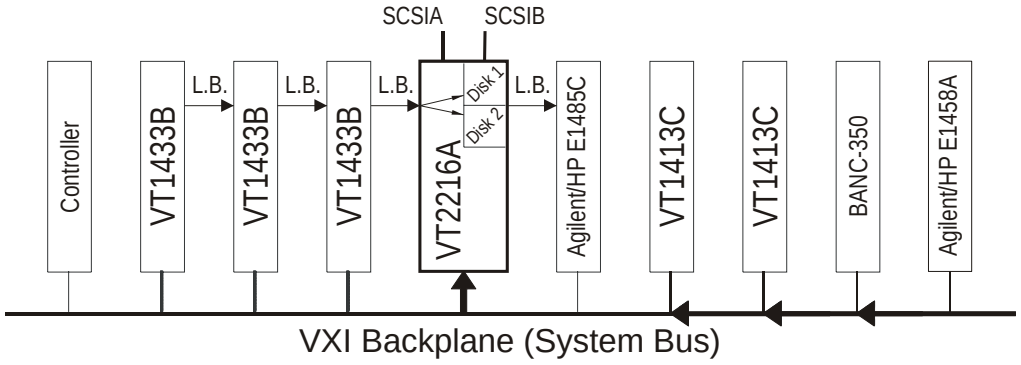


Figure 14 Data Acquisition Using the VXI System Bus

Mixed System Bus and Local Bus

In the following illustration, the VT2216A combines the data from the two busses prior to storing it on the disk.

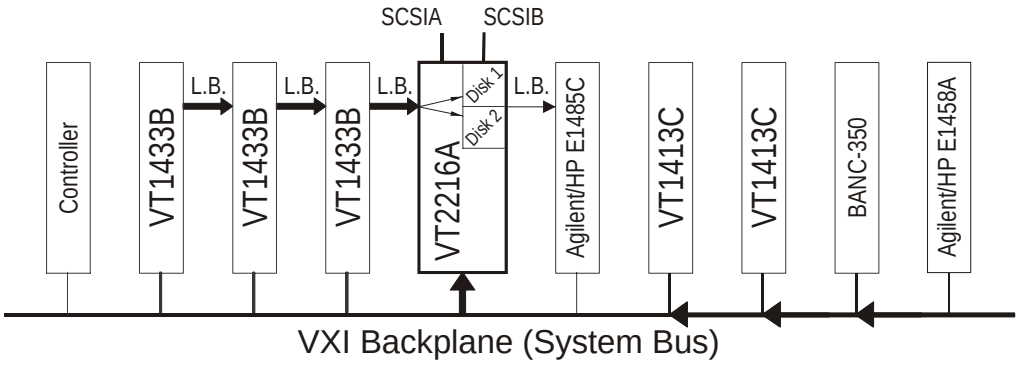


Figure 15 Data Acquisition Using the VXI System Bus and the Local Bus

Monitoring the Local Bus during Throughput

In the following illustration, the VT2216A copies a subset of the channels from the local bus to the system bus. The data is then monitored over the system bus by a controller.

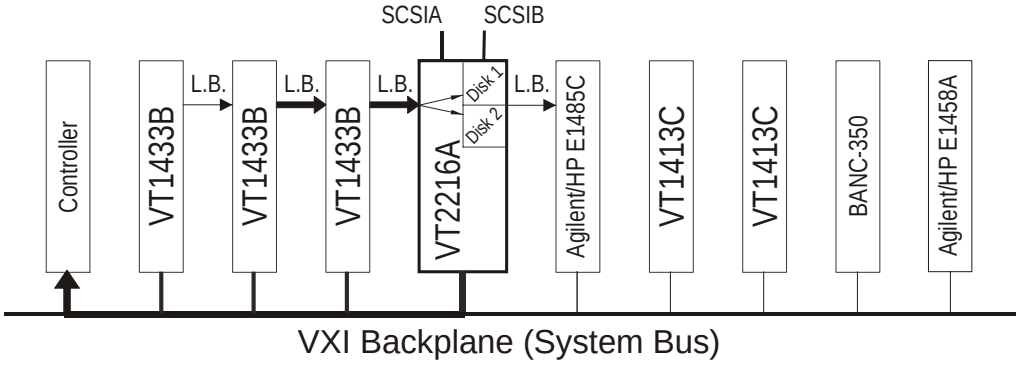


Figure 16 Monitoring During Local Bus Throughput

Monitoring the System Bus during Throughput (Using CVT)

In the following illustration, the VT1413C FIFO is accessed for real-time acquisition of all data to the VT2216A data disk. The VT1413C Current Value Table (CVT) is monitored by the controller. The VT2216A cannot provide data to be monitored.

Monitoring by way of the Current Value Table allows higher bandwidth compared to monitoring the local bus. A disadvantage is that some samples may be missed, but for many applications this does not present a problem.

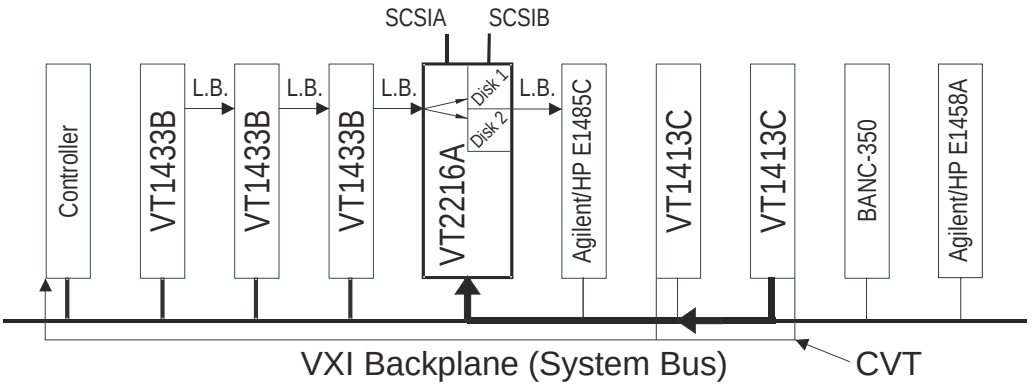


Figure 17 Monitoring During System Bus Throughput (Using CVT)

Monitoring the System Bus During Throughput (via the VT2216A)

In the following illustration, all data input by the VT2216A is “reflected” back out by way of the System Bus for monitoring purposes. The destination for the data could be controller-shared memory.

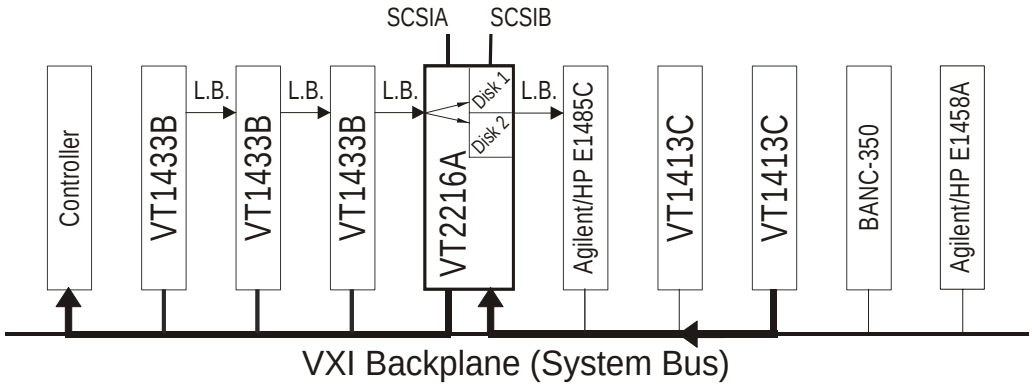


Figure 18 Monitoring During System Bus Throughput (via the VT2216A)

Data Flow

The following illustration shows the data flow in a system using the VT2216A. This system is set up using eight VT1413C scanning A/D (Analog-to-Digital) Converter modules to acquire input and send it to a VT2216A.

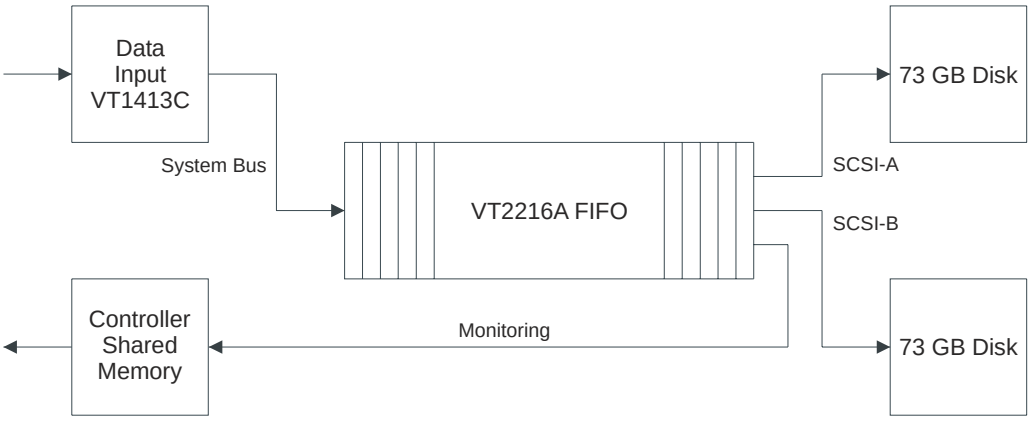


Figure 19 Data Flow

Throughput Directly to an External Digital Recorder

In the following illustration, data from the four VT1433B modules is input to the VT2216A. The VT2216A then outputs the data from its SCSI A connector to an external SCSI device.

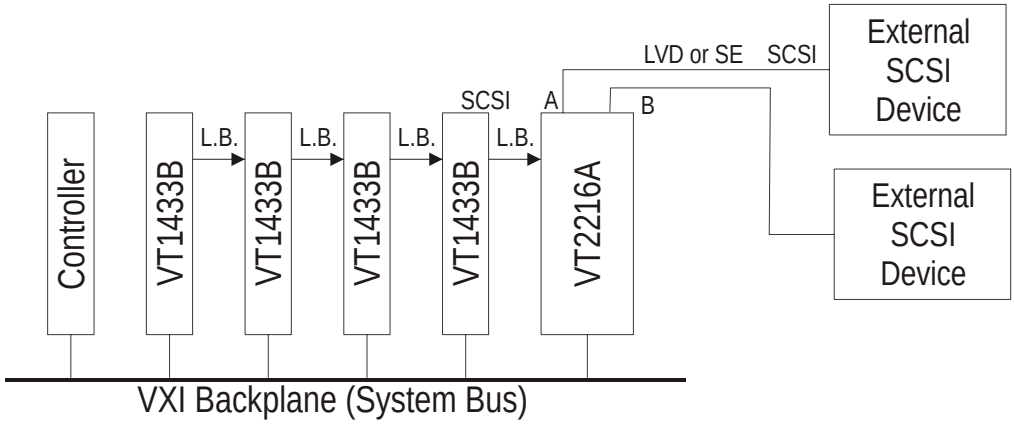


Figure 20 Throughput Directly to an External Digital Recorder (using the VT2216A)

Post-Processing

In post-processing, a Sequence can be used to unwind the data from the disk in the same order as the corresponding acquisition Sequence.

Post-Processing Using the Agilent/HP E1485C VXI Signal Processor

In the following illustration, the Agilent/HP E1485C VXI Signal Processor reads data from the VT2216A using the local bus.

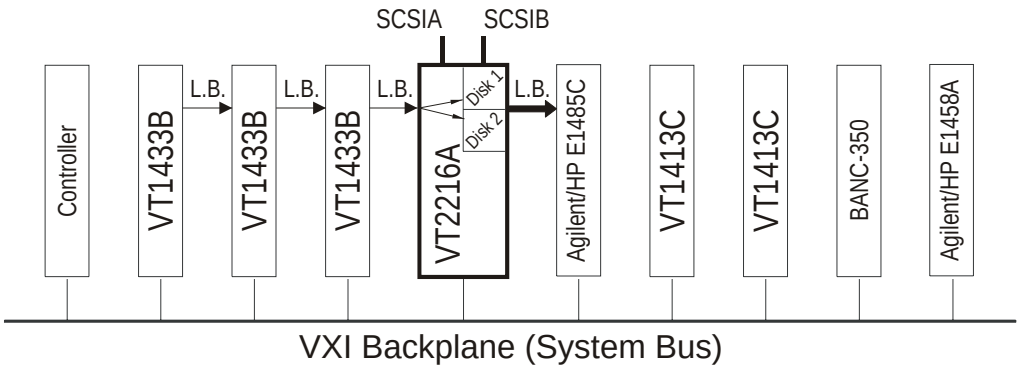


Figure 21 Data Post-Processing Using the Agilent/HP E1485C

Post-Processing Using an Embedded Host

The data can be transferred in several ways. The controller can read the data from the VT2216A disks via shared memory or directly via SCPI commands. Or, the VT2216A can place the data directly into the controller's shared memory. The following illustration shows the controller transferring the data into its own local memory.

Shared memory is memory space in the controller and in the VT2216A that can be accessed by both modules.

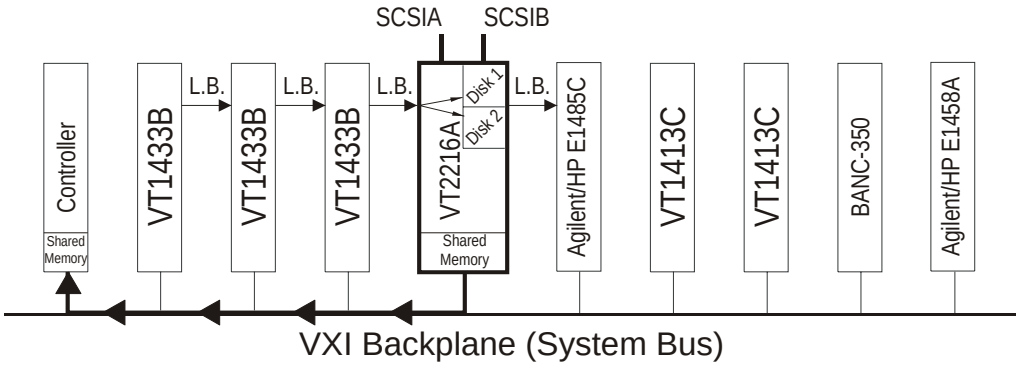


Figure 22 Data Post-Processing Using an Embedded Host Controller

Pre-Processing using the Agilent/HP E1485C VXI Signal Processor

The following illustration shows the Agilent/HP E1485C VXI Signal Processor acting as a pre-processor for data on the local bus that is destined for the VT2216A data disk.

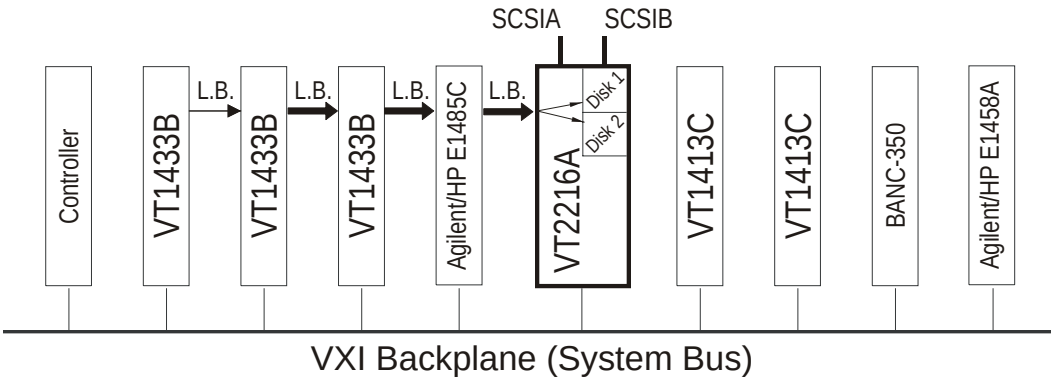


Figure 23 Data Pre-Processing Using the Agilent/HP E1485C

Backup

Backup via Local Bus and Post-Processing

The following illustration shows data sent up to the host controller after first passing through the Agilent/HP E1485C VXI Signal Processor for preliminary processing.

The archive shown in the diagram can be a disk.

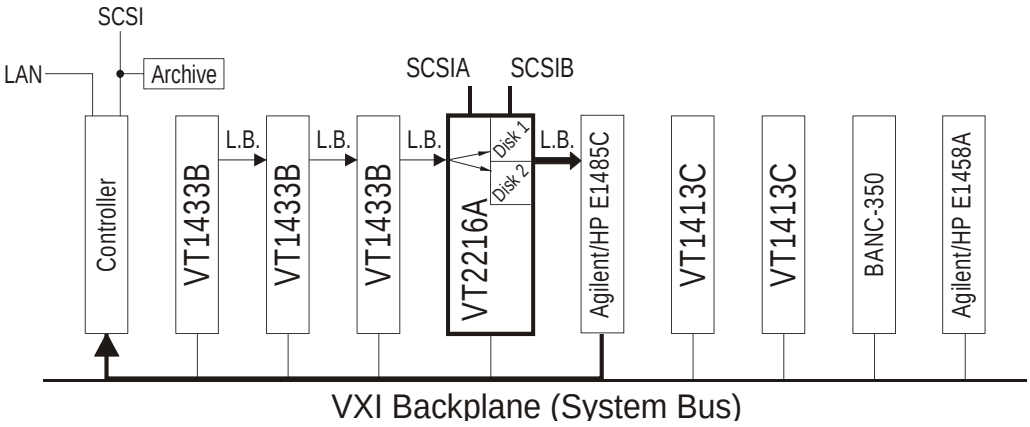


Figure 24 Backup via Local Bus and Post-Processing

Backup via System Bus (VME)

The following illustration shows a throughput Session directly backed up to the host controller. This is the same as the data post-processing using an embedded host controller, previously described.

From the controller, the data can be archived to disk.

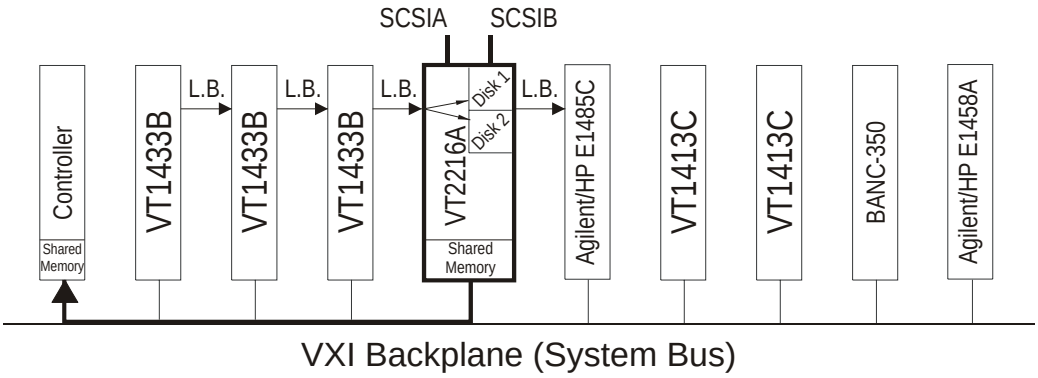


Figure 25 Backup via System Bus

Backup to External DAT

The following illustration shows a throughput Session that has been saved on the disk drive backed up to an external digital audio tape (DAT).

The SCPI command used for backup is MMEMory:SESSion:COPY. See “SCPI Command Reference” starting on page 199 for more information.

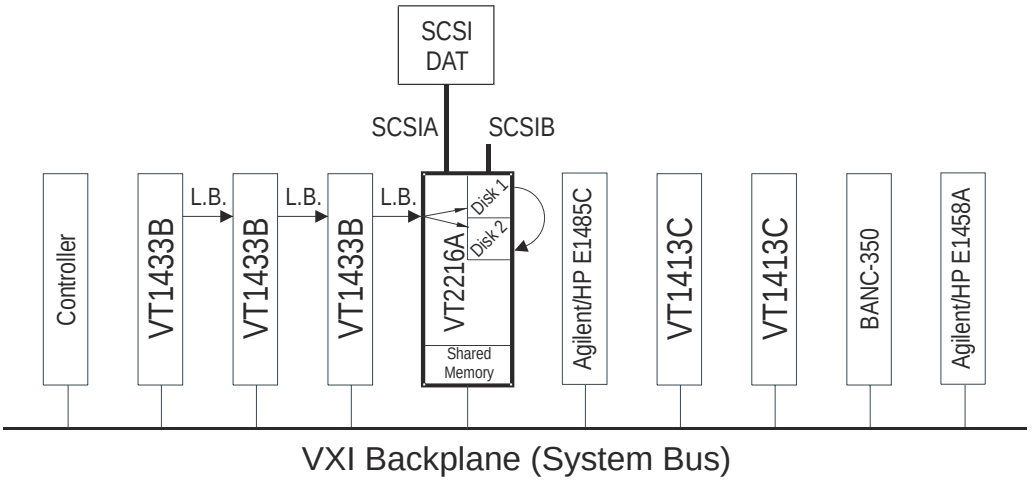


Figure 26 Backup Disk file to External DAT

Backup to Host via SCSI

The following illustration shows copying a throughput Session to the host controller via SCSI. If the throughput Session has been formatted with LIF and exists on a single SCSI device, then that device may be mounted under HP-UX and the file copied out under control of the host computer.

For this type of backup, the host must have a LVD (low-voltage differential) or SE (low-voltage single-ended) SCSI interface and that data must be sent to a LIF file.

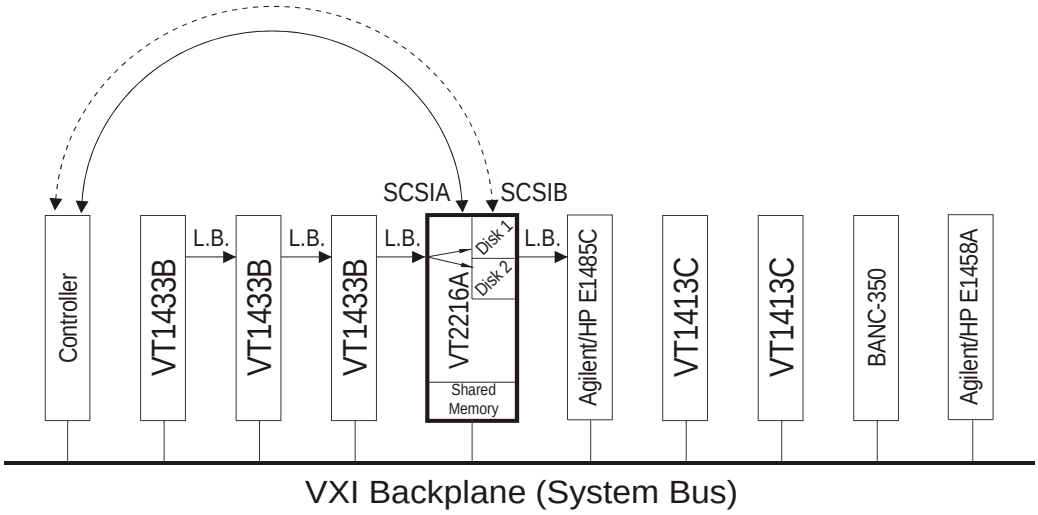


Figure 27 Backup to Host via SCSI

Copying a Split Session to One Disk File

A split Session (a TUNIT split between two SCSI disk drives) can be copied to a Session on a single disk.

In the following illustration, data from the two disks of a VT2216A Option 2 is placed on one of the disks. An alternative is to send the combined data to an external disk, which could then be used as a backup file for the Session.

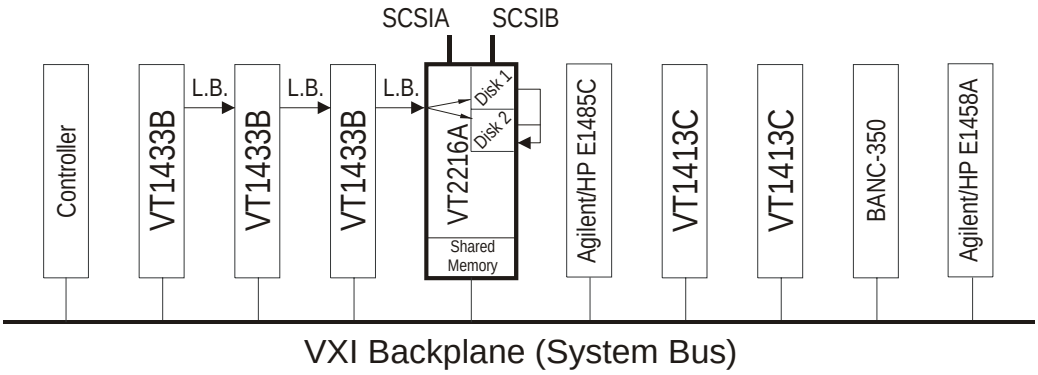


Figure 28 Copying a Split Session

VXI*plug&play* Reference

What is VXI*plug&play*?

VXI Technology uses VXI*plug&play* technology in the VT2216A. This section outlines some of the details of VXI*plug&play* technology.

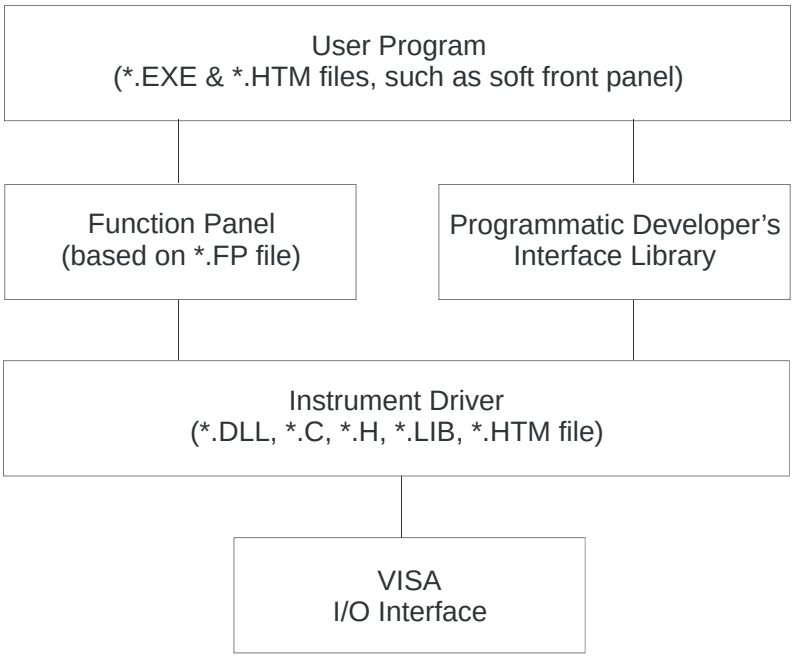
Overview

The fundamental idea behind VXI*plug&play* is to provide VXI users with a level of standardization across different vendors well beyond what the VXI standard specifications delineate. The VXI*plug&play* Alliance specifies a set of core technologies centered on a standard instrument driver technology.

VXI Technology offers VXI*plug&play* drivers for Agilent VEE-Windows. The VXI*plug&play* instrument drivers exist relative to so-called "frameworks." A framework defines the environment in which a VXI*plug&play* driver can operate. The VT2216A has VXI*plug&play* drivers for the following frameworks: Windows NT, Windows 2000 and later and HP-UX.

VXI*plug&play* Drivers

The VT2216A VXI*plug&play* driver is based on the following architecture:



It is most useful to discuss this architecture from the bottom up. The VISA I/O interface allows interoperability of the VXI*plug&play* driver technology across interfaces.

The actual instrument driver is a DLL (Dynamic Linked Library) created from:

- A set of source (.C) files.
- A set of header (.H) files, used for compiling the file as well as to describe the driver's calls to any program using the driver.
- A standard driver library (.LIB) file, to provide the standard functionality all the drivers would require.

This DLL is a set of calls to perform instrument actions—at heart, that's all a VXI*plug&play* driver is—a library of instrument calls.

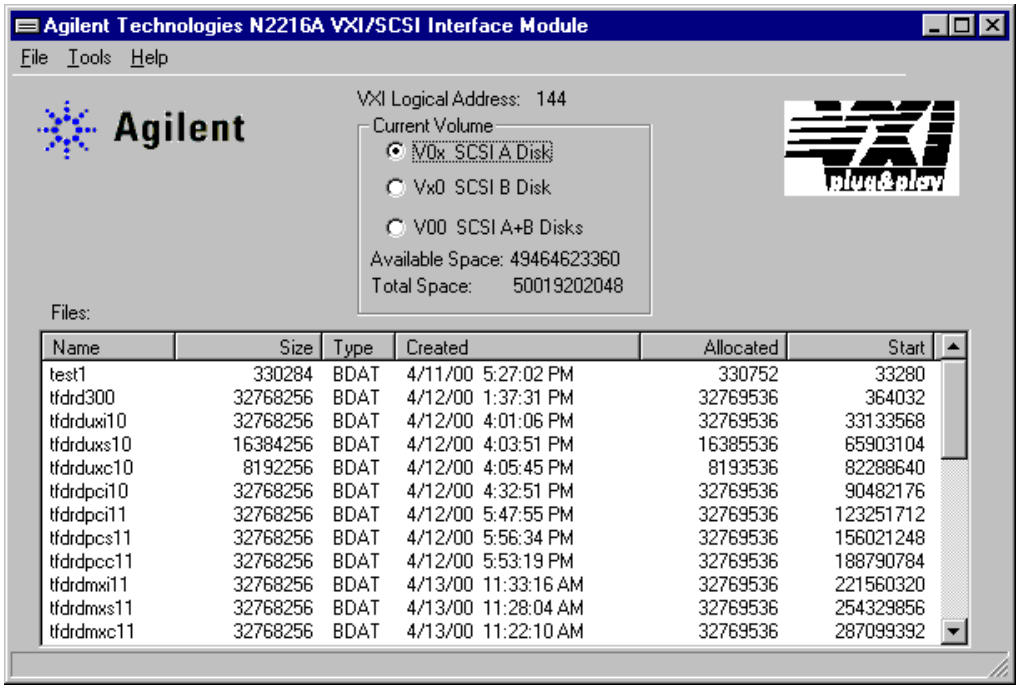
This driver is accessed by Windows applications programs written in languages such as Agilent VEE or NI LabView.

HTM help files are included to provide descriptive information for the functions in the VXI*plug&play* DLL. The HTM help files require a web browser that supports the HTML v3.2, JavaScript 1.2 and CSS1 standards.

The VXI*plug&play* Soft Front Panel

If the VT2216A software is running on a Microsoft Windows NT or a Windows 2000 OS or later, the Soft Front Panel (SFP) program can be used to interface with the VT2216A.

The VT2216A Soft Front Panel helps confirm that the VT2216A is installed correctly. It can also be used to format, list contents, determine version, and test the VT2216A. However, it is not a throughput data viewer or throughput session controller. It cannot be controlled from a program and it does not access all of the VT2216A's functionality.



Getting Updates

For the latest instrument drivers, visit VXI Technology's web site at http://www.vxitech.com for downloads.

Using the VT2216A VXI*plug&play* Library

The VT2216A VXI*plug&play* library simplifies the programming required to record and playback data with the VT2216A. This section is a programming overview. For more details on function usage and parameters see the examples programs. The location of the example programs is listed in the readme file. For specific usage information see the Function Reference (page 82) and VT2216A LIF Commands (page 304) sections.

Recording from the VXI Local Bus

Format the VT2216A

Before using the VT2216A for data recording, format each disk as a LIF volume. A disk can be formated using the VT2216A Soft Front Panel (agn2216.exe on a PC) or typing the LIF command e1562in from the command line (MS-DOS and HP-UX).

Programming Steps for Recording Data

1. Initialize the data source and configure as needed.
2. Call agn2216_init() to initialize the VT2216A module and library.

   When using Agilent VEE, this call is done by Agilent VEE and is not a part of the application.
3. Determine the size in bytes of the data (throughput) file.

   File size = header size (if used) + (size of scans × number of scans). The scan size is the block size times the number of channels.
4. Call agn2216_tputfile_open_record() to create a LIF throughput file of needed size and to open it for writing.

   If a header is used, it can be written by calling agn2216_tputfile_write_aint16() or other agn2216_tputfile_write functions. The header size and structure must be known to the playback application so that it can be read properly and that the beginning of the data can be determined.
5. Call agn2216_tput_setup_record() to setup the VT2216A for local bus throughput recording.
6. Call agn2216_tput_reset_localbus(tputhandle, 1) to reset the VT2216A local bus.
7. Set up the local bus on the data source.
8. Call agn2216_tput_reset_localbus(tputhandle, 0) to enable the VT2216A local bus.
9. Call agn2216_tput_start_record() to start recording data on the VT2216A. The data does not actually begin flowing across the local bus until the data source is started.
10. Start the data source.

11. Call agn2216_tput_finished() to check for the completion of the data recording. Repeat as needed.

12. When the data recording is finished, call agn2216_tput_bytes() to get the number of bytes recorded.

13. Stop the data source from sending more data to the local bus.

14. If it is desired to update a header, call agn2216_tputfile_open_update() to open the file for writing and use an agn2216_tputfile_write function to rewrite the header as needed.

15. Call agn2216_close() to close the VT2216A module and library.

Playing Back Data from a Throughput File

Programming Steps for Playing Back Recorded Data

1. Call agn2216_init() to initialize the VT2216A module and library.

   When using Agilent VEE, this call is done by Agilent VEE and is not a part of the application.

2. Call agn2216_tputfile_open_playback() to open the LIF throughput file for reading.

   If a header is used, it MUST be read by calling agn2216_tputfile_read_aint16() or other agn2216_tputfile_read functions. The header size and structure must be the same as used for recording.

3. Determine the size in bytes of each data transfer (scan) being read.
4. Call agn2216_tput_setup_playback() to setup VT2216A for throughput data file playback.
5. Determine the size in bytes of the data to be read. Exclude header, which should have been read. Data size = size of scans × number of scans.
6. Call agn2216_tput_start_playback() to start the VT2216A reading data into shared RAM.
7. Allocate the memory needed to transfer and process the data to be read.
8. Call agn2216_tput_playback_read_aint16 or other agn2216_tput_playback_read functions to read each scan of data.
9. Process or display data as needed.
10. Repeat the reading of data scans until all desired data has been read.
11. After the data has been read, call agn2216_tput_abort() to stop the playback from the VT2216A.
12. Free the memory allocated earlier.
13. Call agn2216_close() to close the VT2216A module and library.

Function Reference

The VT2216A VXI*plug&play* driver consists of functions, DLLs, and libraries to allow one to program the VT2216A or Agilent/HP E1562 using different program languages. On Windows NT or Windows 2000 and later, Agilent VEE, Visual Basic or Visual C/C++ can be used. On HP-UX 10.2, Agilent VEE or C may be used.

Alphabetical Function Reference

Hierarchical Function Reference

The hierarchical function reference lists the VT2216 VXI*plug&play* functions by classes as defined by the VT2216A function panel. The function panel runs under Agilent VEE, LabWindows® and LabVIEW®.

DRIVER: agn2216.fp

Initialize

- agn2216_init (page 101)

Data Store

- Configure
  - agn2216_init_volume (page 103)
  - agn2216_tput_reset_localbus (page 117)
  - agn2216_tput_setup_playback (page 118)
  - agn2216_tput_setup_record (page 119)
  - agn2216_tputfile_close (page 122)
  - agn2216_tputfile_open_playback (page 123)
  - agn2216_tputfile_open_record (page 124)
  - agn2216_tputfile_open_update (page 125)
  - agn2216_tputfile_seek (page 130)
- Initiate
  - agn2216_tput_abort (page 110)
  - agn2216_tput_bytes (page 111)
  - agn2216_tput_finished (page 112)
  - agn2216_tput_start_playback (page 120)
  - agn2216_tput_start_record (page 121)
- Read-Write
  - agn2216_tput_playback_read_aint16 (page 113)
  - agn2216_tput_playback_read_aint32 (page 114)
  - agn2216_tput_playback_read_aint32_16 (page 115)
  - agn2216_tput_playback_read_char (page 116)
  - agn2216_tputfile_read_aint16 (page 126)
  - agn2216_tputfile_read_aint32 (page 127)
  - agn2216_tputfile_read_areal64 (page 128)
  - agn2216_tputfile_read_char (page 129)
  - agn2216_tputfile_write_aint16 (page 131)
  - agn2216_tputfile_write_aint32 (page 132)

agn2216_close

Close the VXI*plug&play* library and release all resources.

Syntax:

```
ViStatus _VI_FUNC agn2216_close(ViSession vi);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN

Comments:

Close the VXI*plug&play* library, close the VT2216A library, release all resources for use by other programs.

It is good programming practice to call this agn2216_close() function when finished with the resources. It is possible that not calling this function could cause a DLL on the PC to hold some memory or hardware resources until a power down.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

Example:

```
/* close and deallocate resources */
vierr=agn2216_close(session);
if(vierr)
{
    vierr2=agn2216_error_message(session,vierr,st);
    printf("error %d = %s\n",vierr,st);
    exit(0);
}
```

agn2216_cmd

Sends a string to the instrument, for commands where no response is expected.

Syntax: `ViStatus _VI_FUNC agn2216_cmd(ViSession vi, ViString cmd);`

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
cmd	command string sent to the instrument Data Type: ViString Input/Output: IN

Comments: Passes a SCPI command string to the instrument. This must be a null terminated string that does not exceed 8 kB.

For SCPI commands that return numbers use agn2216_cmd_query_int32 or agn2216_cmd_query_real64.

For SCPI commands that return a string use agn2216_cmd_query_string.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_cmd_query_int32

Sends a string to the instrument and returns a numeric response.

Syntax:

```
ViStatus _VI_FUNC agn2216_cmd_query_int32(ViSession vi, ViString cmd,
ViPInt32 response);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
cmd	command string sent to the instrument Data Type: ViString Input/Output: IN
response	response from instrument Data Type: ViPInt32 Input/Output: OUT

Comments:

Passes a SCPI command string to the instrument. This must be a null terminated string that does not exceed 8 kB.

The numeric response is returned as an Int32, space for which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_cmd_query_real64

Sends a string to the instrument and returns a numeric response.

Syntax:

```
ViStatus _VI_FUNC agn2216_cmd_query_real64(ViSession vi, ViString cmd,
ViPReal64 response);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
cmd	command string sent to the instrument Data Type: ViString Input/Output: IN
response	response from instrument Data Type: ViPReal64 Input/Output: OUT

Comments:

Passes a SCPI command string to the instrument. This must be a null terminated string that does not exceed 8 kB.

The numeric response is returned as a Real64, space for which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_cmd_query_string

Sends a string to the instrument and returns a numeric response.

Syntax:

```
ViStatus _VI_FUNC agn2216_cmd_query_string(ViSession vi, ViString cmd,
ViInt32 size, ViChar response[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
cmd	command string sent to the instrument Data Type: ViString Input/Output: IN
size	Size preallocated for response from instrument. Data Type: ViInt32 Input/Output: IN Values: AGN2216_INT32POS_MIN 0 AGN2216_INT32_MAX 2147483647
response	Response from instrument. Data Type: ViChar [] Input/Output: OUT

Comments:

Passes a SCPI command string to the instrument. This must be a null terminated string that does not exceed 8 kB.

The response is returned in a ViChar[] array, which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_error_message

Translates a status number to a string description.

Syntax:

```
ViStatus _VI_FUNC agn2216_error_message(ViSession vi, ViStatus error,
ViChar _VI_FAR message[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
error	This is the ViStatus number for which a description is desired. Data Type: ViStatus Input/Output: IN Values: AGN2216_INT32_MIN -2147483648 AGN2216_INT32_MAX 2147483647
message	Returns the error description string. Data Type: ViChar _VI_FAR [] Input/Output: OUT

Comments:

agn2216_error_message() accepts an error number and buffer and will write a string into the buffer describing the error. The buffer should be at least 256 bytes long.

The error number referred to above is the return status that is returned from every function in the VXI*plug&play* library.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

Example:

```
void main()
{
ViSession       session;
char            st[255];
ViStatus        vierr;
ViStatus        vierr2;

   /* initialize instrument  This code ASSUMES VXI/SCSI Interface module
at address 144 */
    vierr=agn2216_init("VXI0::144::INSTR",0,1,&session);
    if(vierr)
    {
        vierr2 = agn2216_error_message(session,vierr,st);
        printf("error %d = %s\n",vierr,st);
        exit(0);
    }
```

agn2216_error_query

Queries the instrument and returns instrument specific error information.

Syntax:

```
ViStatus _VI_FUNC agn2216_error_query(ViSession vi, ViPInt32 error,
ViChar _VI_FAR error_message[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
error	Instrument error code. Data Type: ViPInt32 Input/Output: OUT
error_message	Returns the instrument error message. Data Type: ViChar _VI_FAR [] Input/Output: OUT

Comments: Sends a SYST:ERR? to the instrument and returns the response. The returned string can be up to 256 characters including null and must be allocated in advance.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

Example:

```
void main()
{
ViSession       session;
char            st[256];
ViStatus        vierr;
ViInt32         instError
  ...
  vierr = agn2216_error_query(session,(ViPInt32)&instError,st);
        printf("error %d = %s\n",instError,st);
        exit(0);
}
```

agn2216_find

Find all VXI/SCSI Interface modules in the VXI bus.

Syntax:

```
ViStatus _VI_FUNC agn2216_find(ViSession vi, ViInt32 addList[ ], ViInt32
listSize, ViPInt32 numFound, ViChar rsrc[ ], ViInt32 rsrcLen);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init() or a VI_NULL. Data Type: ViSession Input/Output: IN
addList	An array to hold VXI logical addresses of VXI/SCSI Interface modules. Data Type: ViInt32 [] Input/Output: OUT
listSize	Size of the addList array. Data Type: ViInt32 Input/Output: IN Values: AGN2216_FIND_LIST_SIZE_MIN 0 AGN2216_FIND_LIST_SIZE_MAX 255
numFound	Returns the number of VXI/SCSI Interface modules found. Data Type: ViPInt32 Input/Output: OUT
rsrc	Returns the resource name for the first found VXI/SCSI Interface module. Data Type: ViChar [] Input/Output: OUT
rsrcLen	Sets the size of the rsrc[] buffer. Data Type: ViInt32 Input/Output: IN Values: AGN2216_FIND_RSRC_LEN_MIN 0 AGN2216_FIND_RSRC_LEN_MAX 255

Comments:

agn2216_find() searches the VXI mainframe and returns the VXI Logical Address for every VXI/SCSI Interface module found.

Be sure to allocate the ViInt32 array before calling agn2216_find().

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine "agn2216_error_message"

agn2216_find_default_volume

Find the first disks on SCSI channel A and B.

Syntax:

```
ViStatus _VI_FUNC agn2216_find_default_volume(ViSession vi, ViPString volA, ViPString volB);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
volA	Returns the string of the hex address of the first disk on SCSI A (0--f). Data Type: ViPString Input/Output: OUT
volB	Returns the string of the hex address of the first disk on SCSI B (0--f). Data Type: ViPString Input/Output: OUT

Comments: agn2216_find_default_volume() searches the VT2216A SCSI buses for disks and returns the lowest address found as hex ASCII. Returns x if no disk is found on that SCSI channel.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine "agn2216_error_message"

agn2216_get_debuglevel

Returns the current VT2216A VXI*plug&play* library debuglevel setting for that vi.

Syntax:

```
ViStatus _VI_FUNC agn2216_get_debuglevel(ViSession vi, ViPInt16
timeout);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
timeout	Debug level Data Type: ViPInt16 Input/Output: OUT

Comments: The default debuglevel is 0 and no VT2216A VXI*plug&play* library debug information is printed.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_get_dir_entry

Gets the next file directory entry.

Syntax:

```
ViStatus _VI_FUNC agn2216_get_dir_entry(ViSession vi, ViString volume,
ViPString filename, ViPString date, ViPString time, ViPString type,
ViPReal64 size, ViPReal64 allocated, ViPReal64 start);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
volume	A 3 character string for the volume, Examples "V00", "Vx0", "V0x." Data Type: ViString Input/Output: IN
filename	Returns the string containing the filename for this entry. Data Type: ViPString Input/Output: OUT
date	Returns the string containing the date in the form "mmm dd,yyyy." Data Type: ViPString Input/Output: OUT
time	Returns the string containing the time in the form "hh:mm:ss." Data Type: ViPString Input/Output: OUT
type	Returns the string containing the file type (such as BDAT). Data Type: ViPString Input/Output: OUT
size	File size used in bytes. Data Type: ViPReal64 Input/Output: OUT
allocated	File size allocated in bytes. Data Type: ViPReal64 Input/Output: OUT
start	Starting location, bytes. Data Type: ViPReal64 Input/Output: OUT

Comments: agn2216_get_dir_entry() returns the next file directory entry and advances the current get_dir_entry location. agn2216_get_first_dir_entry() must be called before agn2216_get_dir_entry(), to reset the current get_dir_entry location.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine "agn2216_error_message"

agn2216_get_first_dir_entry

Gets the first entry in the file directory.

Syntax:

```
ViStatus _VI_FUNC agn2216_get_first_dir_entry(ViSession vi, ViString
volume, ViPString filename, ViPString date, ViPString time, ViPString
type, ViPReal64 size, ViPReal64 allocated, ViPReal64 start);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
volume	A three character string for the volume, Examples "V00", "Vx0", "V0x." Data Type: ViString Input/Output: IN
filename	Returns the string containing the filename for this entry. Data Type: ViPString Input/Output: OUT
date	Returns the string containing the date in the form "mmm dd,yyyy." Data Type: ViPString Input/Output: OUT
time	Returns the string containing the time in the form "hh:mm:ss." Data Type: ViPString Input/Output: OUT
type	Returns the string containing the file type (such as BDAT). Data Type: ViPString Input/Output: OUT
size	File size used in bytes. Data Type: ViPReal64 Input/Output: OUT
allocated	File size allocated in bytes. Data Type: ViPReal64 Input/Output: OUT
start	Starting location, bytes. Data Type: ViPReal64 Input/Output: OUT

Comments: agn2216_get_first_dir_entry() returns the first entry in the file directory. Sets the current get_dir_entry location to the 2nd entry.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine "agn2216_error_message"

agn2216_get_timeout

Returns the current timeout in milliseconds.

Syntax:

```
ViStatus _VI_FUNC agn2216_get_timeout(ViSession vi, ViPInt32 timeout);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
timeout	Timeout in milliseconds Data Type: ViPInt32 Input/Output: OUT

Comments: Some disk operations are slow, so the default timeout, set by agn2216_init, is thirty seconds.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_init

Initialize the VXI*plug&play* library and register all VXI/SCSI Interface modules.

Syntax:

```
ViStatus _VI_FUNC agn2216_init(ViRsrc rsrcName, ViBoolean id_query,
ViBoolean reset, ViPSession vi);
```

Parameter	Description
rsrcName	The Instrument Description. Data Type: ViRsrc Input/Output: IN

id_query	If(VI_TRUE) Perform In-System Verification.
	
	If(VI_FALSE) Do not perform In-System Verification.
	
	Data Type: ViBoolean
	Input/Output: IN

reset	If(VI_TRUE) Perform Reset Operation.
	
	If(VI_FALSE) Do not perform Reset operation.
	
	Data Type: ViBoolean
	Input/Output: IN

vi	Instrument Handle. This is VI_NULL if an error occurred during the init. Data Type: ViPSession Input/Output: OUT

Comments:

The initialize function initializes the software connection to the instrument and optionally verifies that instrument is in the system. It also initializes the VT2216A library. If the VT2216A library is already in use an error may occur.

If the agn2216_init() function encounters an error, then the value of the vi output parameter will be VI_NULL.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

Example:

```
void main()
{
ViSession       session;
char            st[100];
ViStatus        vierr;
ViStatus        vierr2;

    /* initialize instrument  This code ASSUMES VXI/SCSI Interface module
at address 144 */
```

```
vierr=agn2216_init("VXI0::144::INSTR",0,1,&session);
if(vierr)
{
    vierr2=agn2216_error_message(session,vierr,st);
    printf("error %d = %s\n",vierr,st);
    exit(0);
}
```

agn2216_init_volume

Initializes (formats) the volume with a LIF directory.

Syntax:

```
ViStatus _VI_FUNC agn2216_init_volume(ViSession vi, ViString volume);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
volume	A string for the volume, Examples "V00", "Vx0", "V0x." Data Type: ViString Input/Output: IN

Comments: The init_volume function initializes a LIF directory on the disk(s) corresponding to the volume. Existing data will be lost.

Return Value: **VI_SUCCESS**: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_reset

The function returns the instrument to the reset state.

Syntax:

```
ViStatus _VI_FUNC agn2216_reset(ViSession vi);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN

Comments: The function sends the instrument a "*RST", returning the instrument to the reset state.

In addition, this function cancels any pending command or query.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_revision_query

Returns revision information for both the driver on the host and the software in the VXI/SCSI Interface module. The returned string can be up to 256 characters including null.

Syntax:

```
ViStatus _VI_FUNC agn2216_revision_query(ViSession vi, ViChar driver_
rev[ ], ViChar instr_rev[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
driver_rev	Returns a string containing the software revision information. Data Type: ViChar [] Input/Output: OUT
instr_rev	Returns a string containing the instrument driver and firmware revision. Data Type: ViChar [] Input/Output: OUT

Comments: The returned string can be up to 256 characters including null and must be allocated in advance.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_self_test

Performs a selftest of the hardware.

Syntax:

```
ViStatus _VI_FUNC agn2216_self_test(ViSession vi, ViPInt16 testResult,
ViChar testMessage[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
testResult	result of test Data Type: ViPInt16 Input/Output: OUT
testMessage	string result of test Data Type: ViChar [] Input/Output: OUT

Comments:

agn2216_self_test() performs a selftest of the hardware. It returns 0 if all tests pass or a negative error code if the self test fails.

testMessage is a short message written back, so be sure to allocate at least eighty characters.

Specify which boards to test by calling agn2216_init(). Even if agn2216_init() fails, the board logical addresses are saved for use by agn2216_self_test().

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

Example:

```
char          message[100];
ViStatus      vierr;
ViStatus      vierr2;
ViSession     session;
ViInt16       result;

   /* initialize instrument  This code ASSUMES VXI/SCSI Interface module
at address 144 */
    vierr=agn2216_init("VXI0::144::INSTR",0,1,&session);
    if(vierr)
    {
        vierr2=agn2216_error_message(session,vierr,message);
        printf("error %d = %s\n",vierr,st);
    }
```

```
vierr=agn2216_self_test(session,&result,message)
if(vierr)
{
    vierr2=agn2216_error_message(session,vierr,message);
    printf("error %d = %s\n",vierr,st);
    exit(0);
}
printf("selftest reports %d = %s\n",result,message);
```

agn2216_set_debuglevel

Sets the debug level for the VT2216A VXI*plug&play* library.

Syntax:

```
ViStatus _VI_FUNC agn2216_set_debuglevel(ViSession vi, ViInt16
debuglevel);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
debuglevel	 Data Type: ViInt16 Input/Output: IN Values: AGN2216_INT16POS_MIN 0 AGN2216_INT16_MAX 32767

Comments: The default debuglevel is 0 and no VT2216A VXI*plug&play* library debug information is printed. If debuglevel 1, then error information is printed to stderr. If debug > 1, then additional debug information is printed to stdout.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_set_timeout

Sets the timeout in milliseconds.

Syntax:

```
ViStatus _VI_FUNC agn2216_set_timeout(ViSession vi, ViInt32 timeout);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
timeout	Data Type: ViInt32 Input/Output: IN Values: AGN2216_INT32POS_MIN AGN2216_INT32_MAX 2147483647

Comments: Some disk operations are slow, so the default timeout, set by agn2216_init, is thirty seconds.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_abort

Halt a disk module sequence.

Syntax: `ViStatus _VI_FUNC agn2216_tput_abort(ViSession vi);`

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN

Comments: This can be used to abort a throughput record before it would normally complete. This function may also be used to stop a playback sequence early and place the VXI/SCSI Interface module in an idle state.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_bytes

Gets the number of bytes for the finished session.

Syntax: `ViStatus _VI_FUNC agn2216_tput_bytes(ViSession vi, ViPReal64 dataBytes);`

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
dataBytes	Data Type: ViPReal64 Input/Output: OUT

Comments: Gets the number of bytes for the finished session.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_finished

Checks to see if the throughput session is finished.

Syntax: `ViStatus _VI_FUNC agn2216_tput_finished(ViSession vi, ViPInt32 flag);`

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
flag	Data Type: ViPInt32 Input/Output: OUT

Comments: The flag is set when record is finished. If the session is finished, this function also updates the size of the thruput session, which is returned by agn2216_init().

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_playback_read_aint16

Copies bytes from playback to buffer.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_playback_read_aint16(ViSession vi,
ViInt32 size, ViPInt32 readSize, ViInt16 buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
size	Number of bytes (not data elements) to be read. Must be a multiple of two bytes. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes (not data elements) returned in buf. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViInt16 [] Input/Output: OUT

Comments:

Reads throughput file via shared memory. Returns data as ViInt16[], array. Performs the required synchronization with the VT2216A to get bytes transferred into its shared RAM and copy the bytes into the host. The value size should be less than or equal to 262142 ((256*1024) - 2), AGN2216_DATA_SRAM_MAX and should be the same as the value bytesPerScan passed to agn2216_tput_setup_playback(). The response is returned in a buffer array, which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_playback_read_aint32

Copies bytes from playback to buffer.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_playback_read_aint32(ViSession vi,
ViInt32 size, ViPInt32 readSize, ViInt32 buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
size	Number of bytes (not data elements) to be read. Must be a multiple of two bytes. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes (not data elements) returned in buf. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViInt32 [] Input/Output: OUT

Comments:

Reads throughput file via shared memory. Returns data as ViInt32[], array. Performs the required synchronization with the VT2216A to get bytes transferred into its shared RAM and copy the bytes into the host. The value size should be less than or equal to 262142 ((256*1024) - 2), AGN2216_DATA_SRAM_MAX and should be the same as the value bytesPerScan passed to agn2216_tput_setup_playback(). The response is returned in a buf array, which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_playback_read_aint32_16

Copies bytes from playback to buffer converting int16 to int32.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_playback_read_aint32_16(ViSession vi,
ViInt32 size, ViPInt32 readSize, ViInt32 buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
size	Number of bytes (not data elements) to be read. Must be a multiple of four bytes. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes (not data elements) returned in buf. This will be 2*size. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViInt32 [] Input/Output: OUT

Comments:

Reads throughput file via shared memory. Returns ViInt16 data as ViInt32[] array. Performs the required synchronization with the VT2216A to get bytes transferred into its shared RAM and copy the bytes into the host. The value size should be less than or equal to 262142 ((256*1024) - 2), AGN2216_DATA_SRAM_MAX and should be the same as the value bytesPerScan passed to agn2216_tput_setup_playback(). The response is returned in a buf array, which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_playback_read_char

Copies bytes from playback to buffer.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_playback_read_char(ViSession vi, ViInt32
size, ViPInt32 readSize, ViChar buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
size	Number of bytes to be read. Must be a multiple of two bytes. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes returned in buf[]. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViChar [] Input/Output: OUT

Comments:

Reads throughput file via shared memory. Returns data as ViChar[] array. Performs the required synchronization with the VT2216A to get bytes transferred into its shared RAM and copy the bytes into the host. The value size should be less than or equal to 262142 ((256*1024) - 2), AGN2216_DATA_SRAM_MAX and should be the same as the value bytesPerScan passed to agn2216_tput_setup_playback(). The response is returned in a buf[] array, which must be allocated before calling this function.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_reset_localbus

Put the disk module local bus into or out of reset.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_reset_localbus(ViSession vi, ViInt16 resetState);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
resetState	Reset action. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUT_LBUSRESET_OUT 0 AGN2216_TPUT_LBUSRESET_IN 1

Comments:

Allow the application to put the disk module local bus into reset or take it out of reset. This is needed to provide for the safe reset of all devices on the local bus. For example, a safe reset consists of first placing all adjacent local bus devices into reset, then from left to right in the VXI card cage, take each device's local bus out of reset.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_setup_playback

Initialize the disk module sequence to perform a playback.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_setup_playback(ViSession vi, ViReal64
dataOffset, ViInt32 bytesPerScan);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
dataOffset	Byte offset into file. Must be a multiple of two bytes. Data Type: ViReal64 Input/Output: IN Values: AGN2216_TPUT_BYTES_MIN 0 AGN2216_TPUT_BYTES_MAX 4503599627370503
bytesPerScan	Scan size, bytes. Must be a multiple of two bytes. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142

Comments:

Playback uses an disk module sequence to read bytes into the shared RAM on disk module. Synchronization between the host program and the disk module is done through the use of the first two bytes of disk module shared RAM as a flag. By using the routine agn2216_tput_playback_read(), all synchronization is encapsulated. Be aware that the maximum value for bytesPerScan is 262142. A session is setup to be used with the disk module sequence. The open playback file is closed. The functions agn2216_tput_playback_read_aint16(), etc., are used to perform the read and the required synchronization with the disk module after agn2216_tput_start_playback() has been called.

Data_offset is a ViReal64 because ViInt32 does not have enough bits to represent the number of bytes in a very large file.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_setup_record

Initialize the disk module sequence to perform a local bus throughput.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_setup_record(ViSession vi, ViInt32
bytesPerInputBlock, ViInt16 numberInputs, ViPInt32 retPadByte);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
bytesPerInputBlock	Bytes per input channel block. Must be a multiple of four bytes. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
numberInputs	Number of input channels. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUT_INT16POS_MIN 0 AGN2216_TPUT_INT16_MAX 32767
retPadByte	Number of bytes to the next disk sector. Data Type: ViPInt32 Input/Output: OUT

Comments:

Initialize the disk module sequence to perform a local bus throughput with the specified number of inputs and blocksize. A constant blocksize for all channels is assumed. A disk module session is setup to write at the next SCSI block boundary in the file opened in agn2216_tputfile_open_record(). The open file is then closed, as the session is used for the throughput instead of the file. Any header information to be written to the file should be written before calling this function as the current position of the file pointer is used to determine where to start writing the data. The number of bytes needed for padding is returned so that the offset in the file to the recorded data can be determined.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_start_playback

Start the disk module playback sequence.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_start_playback(ViSession vi, ViReal64
lengthInBytes, ViInt32 firstScanOffset);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
lengthInBytes	Playback length. Must be a multiple of two bytes. Data Type: ViReal64 Input/Output: IN Values: AGN2216_TPUT_BYTES_MIN 0 AGN2216_TPUT_BYTES_MAX 4503599627370503
firstScanOffset	Scan number of the first scan. Data Type: ViInt32 Input/Output: IN Values: AGN2216_INT32POS_MIN 0 AGN2216_INT32_MAX 2147483647

Comments: Start the disk module playback sequence. The argument firstScanOffset is the scan number of the first scan—it is zero otherwise.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tput_start_record

Start the VXI/SCSI Interface module record sequence.

Syntax:

```
ViStatus _VI_FUNC agn2216_tput_start_record(ViSession vi, ViReal64
lengthInBytes);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
lengthInBytes	Record length. Must be a multiple of four bytes. Data Type: ViReal64 Input/Output: IN Values: AGN2216_TPUT_BYTES_MIN 0 AGN2216_TPUT_BYTES_MAX 4503599627370503

Comments: Start the record sequence.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_close

Close a LIF file.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_close(ViSession vi, ViInt16
tputfileId);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32

Comments: Closes an existing LIF file, removing it from the open file table.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_open_playback

Open a disk module LIF file in preparation for a playback or reading.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_open_playback(ViSession vi, ViString filename, ViPString fullFilename, ViPInt16 tputfileid);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
filename	File to open. Data Type: ViString Input/Output: IN
fullFilename	Full filename returned. Data Type: ViPString Input/Output: OUT
tputfileid	Throughput file ID returned. Data Type: ViPInt16 Input/Output: OUT

Comments:

Opens an existing LIF file in preparation for a throughput post-process. The open file's tputfileid is returned so the application may read any header information contained in the file. The volume name may be defaulted so the full filename including the volume is returned.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_open_record

Create and open a LIF file in preparation for a throughput record.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_open_record(ViSession vi, ViString
filename, ViReal64 totalBytes, ViPString fullFilename, ViPInt16
tputfileid);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
filename	File to open. Data Type: ViString Input/Output: IN
totalBytes	Data Type: ViReal64 Input/Output: IN Values: AGN2216_TPUT_BYTES_MIN 0 AGN2216_TPUT_BYTES_MAX 4503599627370503
fullFilename	Full filename returned. Data Type: ViPString Input/Output: OUT
tputfileid	Throughput file ID returned. Data Type: ViPInt16 Input/Output: OUT

Comments:

The total file size must be known at open time, as a LIF file cannot be extended in size after it has been created. This means that the value of totalBytes must include the size of the data to be recorded, the size of any header information and the size of any information following the recorded data. The value passed to totalBytes will be padded with one SCSI blocksize, since the recorded data must start at a SCSI block boundary. This will mean that any directory listing may show a different size for the file than what was passed to this function. The allocated file size, in bytes, is returned.

Since this routine may be called with just a filename allowing the volume to be defaulted, the full filename is returned to the calling function. It is possible to default the volume name only if a single disk is used as the LIF volume, or, if a pair of disks, one on each SCSI bus of the VXI/SCSI Interface module, is used as the LIF volume. Also, the SCSI devices must be the lowest addressed devices on the SCSI bus. If the volume name is specified, any number of SCSI devices may be used to make up the file system. This function expects that a file system already exists on the devices. This can be done using the e1565in.exe command from the LIF utilities or from the soft front panel.

Return Value:

VI_SUCCESS: No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_open_update

Open an existing file for modification.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_open_update(ViSession vi, ViString fullFileName, ViPInt16 tputfileid);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
fullFileName	Full file name. Data Type: ViString Input/Output: IN
tputfileid	Throughput file ID returned. Data Type: ViPInt16 Input/Output: OUT

Comments: Open an existing file for modification.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_read_aint16

Read file from current location.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_read_aint16(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViPInt32 readSize, ViInt16 buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be read. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes (not data elements) returned in buf. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViInt16 [] Input/Output: OUT

Comments: Read file from current location using the VT2216A shared memory. The response is returned in a ViInt16 buf[] array, which must be allocated before calling this function.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_read_aint32

Read file from current location.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_read_aint32(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViPInt32 readSize, ViInt32 buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be read. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes (not data elements) returned in buf. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViInt32 [] Input/Output: OUT

Comments: Read file from current location using the VT2216A shared memory. The response is returned in a ViInt32 buf[] array, which must be allocated before calling this function.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_read_areal64

Read file from current location.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_read_areal64(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViPInt32 readSize, ViReal64 buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be read. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
readSize	Number of bytes (not data elements) returned in buf. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViReal64 [] Input/Output: OUT

Comments: Read file from current location using the VT2216A shared memory. The response is returned in a ViReal64 buf[] array, which must be allocated before calling this function.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_read_char

Read file from current location.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_read_char(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViPInt32 readSize, ViChar buf[ ]);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes to be read. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0
readSize	Number of bytes returned in buf[]. Data Type: ViPInt32 Input/Output: OUT
buf	Data buffer. Data Type: ViChar [] Input/Output: OUT

Comments: Read file from current location using the VT2216A shared memory. The response is returned in a ViChar buf[] array, which must be allocated before calling this function.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_seek

Seek to an absolute location in a file.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_seek(ViSession vi, ViInt16
tputfileId, ViReal64 dataOffset);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
dataOffset	Absolute seek location, bytes. Data Type: ViReal64 Input/Output: IN Values: AGN2216_TPUT_BYTES_MIN 0 AGN2216_TPUT_BYTES_MAX 4503599627370503

Comments: Seek to an absolute location in a file.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_write_aint16

Write data from ViInt16 buf[] to a VT2216A LIF file.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_write_aint16(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViInt16 buf[ ], ViPInt32 writeSize);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be written. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
buf	Data buffer. Data Type: ViInt16 [] Input/Output: IN
writeSize	Number of bytes (not data elements) written from buf. Data Type: ViPInt32 Input/Output: OUT

Comments: Write data from ViInt16 buf[] to current file location using the VT2216A shared memory.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_write_aint32

Write data from ViInt32 buf[] to a VT2216A LIF file.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_write_aint32(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViInt32 buf[ ], ViPInt32 writeSize);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	 Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be written. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
buf	Data buffer. Data Type: ViInt32 [] Input/Output: IN
writeSize	Number of bytes (not data elements) written from buf. Data Type: ViPInt32 Input/Output: OUT

Comments: Write data from ViInt32 buf[] to current file location using the VT2216A shared memory.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_write_areal64

Write data from ViReal64 buf[] to a VT2216A LIF file.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_write_areal64(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViReal64 buf[ ], ViPInt32 writeSize);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be written. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
buf	Data buffer. Data Type: ViReal64 [] Input/Output: IN
writeSize	Number of bytes (not data elements) written from buf. Data Type: ViPInt32 Input/Output: OUT

Comments: Write data from ViReal64 buf[] to current file location using the VT2216A shared memory.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

agn2216_tputfile_write_char

Write data from ViChar buf[] to a VT2216A LIF file.

Syntax:

```
ViStatus _VI_FUNC agn2216_tputfile_write_char(ViSession vi, ViInt16
tputfileId, ViInt32 size, ViChar buf[ ], ViPInt32 writeSize);
```

Parameter	Description
vi	Instrument Handle returned from agn2216_init(). Data Type: ViSession Input/Output: IN
tputfileId	File id returned from agn2216_tputfile_open_ () functions. Data Type: ViInt16 Input/Output: IN Values: AGN2216_TPUTFILEID_MIN 0 AGN2216_TPUTFILEID_MAX 32
size	Number of bytes (not data elements) to be written. Data Type: ViInt32 Input/Output: IN Values: AGN2216_TPUT_TRANSFER_MIN 0 AGN2216_TPUT_TRANSFER_MAX 262142
buf	Data buffer. Data Type: ViChar [] Input/Output: IN
writeSize	Number of bytes (not data elements) written from buf. Data Type: ViPInt32 Input/Output: OUT

Comments: Write data from ViChar buf[] to current file location using the VT2216A shared memory.

Return Value: **VI_SUCCESS:** No error

Non VI_SUCCESS: Indicates error condition. To determine error message, pass the return value to routine agn2216_error_message.

VXI*plug&play* Library Errors

Error Number	Description
-1074003967	Parameter 1 is invalid
-1074003966	Parameter 2 is invalid
-1074003965	Parameter 3 is invalid
-1074003964	Parameter 4 is invalid
-1074003963	Parameter 5 is invalid
-1074003962	Parameter 6 is invalid
-1074003961	Parameter 7 is invalid
-1074003960	Parameter 8 is invalid
-1074003951	Instrument IDN does not match
-1074001184	VT2216A Error Unknown
-1074001183	VT2216A Error 1 - No session available (see Error Number 1 on page 311)
-1074001182	VT2216A Error 2 - Invalid volume name (see Error Number 2 on page 311)
-1074001181	VT2216A Error 3 - Missing volume name (see Error Number 3 on page 311)
-1074001180	VT2216A Error 4 - Volume already open (see Error Number 4 on page 311)
-1074001179	VT2216A Error 5 - Interface error (see Error Number 5 on page 311)
-1074001178	VT2216A Error 6 - Out of memory (see Error Number 6 on page 311)
-1074001177	VT2216A Error 7 - System error (see Error Number 7 on page 311)
-1074001176	VT2216A Error 8 - Invalid id (see Error Number 8 on page 311)
-1074001175	VT2216A Error 9 - Invalid file size (see Error Number 9 on page 311)
-1074001174	VT2216A Error 10 - Invalid file name (see Error Number 11 on page 311)
-1074001173	VT2216A Error 11 - Invalid file mode (see Error Number 11 on page 311)
-1074001172	VT2216A Error 12 - File does not exist (see Error Number 12 on page 311)
-1074001171	VT2216A Error 13 - File does not have size (see Error Number 13 on page 311)
-1074001170	VT2216A Error 14 - File size already specified (see Error Number 14 on page 311)
-1074001169	VT2216A Error 15 - End of file before transfer completed (see Error Number 15 on page 311)
-1074001168	VT2216A Error 16 - Invalid file type (see Error Number 16 on page 311)
-1074001167	VT2216A Error 17 - End of directory before transfer completed (see Error Number 17 on page 311)
-1074001166	VT2216A Error 18 - Not a LIF volume (see Error Number 18 on page 311)

Error Number	Description
-1074001165	VT2216A Error 19 - File rename volume error (see Error Number 19 on page 311)
-1074001164	VT2216A Error 20 - File position is past end of file (see Error Number 20 on page 311)
-1074001163	VT2216A Error 21 - End of file cannot be set beyond file size (see Error Number 21 on page 311)
-1074001162	VT2216A Error 22 - File open, cannot rename, move, copy, or delete (see Error Number 22 on page 311)
-1074001152	N2216 SCPI Error Unknown
-1074001052	N2216 SCPI Error -100 - Command error (see Error Number -100 on page 271)
-1074001051	N2216 SCPI Error -101 - Invalid character (see Error Number -101 on page 271)
-1074001050	N2216 SCPI Error -102 - Syntax error (see Error Number -102 on page 271)
-1074001049	N2216 SCPI Error -103 - Invalid separator (see Error Number -103 on page 271)
-1074001048	N2216 SCPI Error -104 - Data type error (see Error Number -104 on page 271)
-1074001047	N2216 SCPI Error -105 - GET not allowed (see Error Number -105 on page 271)
-1074001044	N2216 SCPI Error -108 - Parameter not allowed (see Error Number -108 on page 271)
-1074001043	N2216 SCPI Error -109 - Missing parameter (see Error Number -109 on page 271)
-1074001042	N2216 SCPI Error -110 - Command header error (see Error Number -110 on page 271)
-1074001041	N2216 SCPI Error -111 - Header separator error (see Error Number -111 on page 271)
-1074001040	N2216 SCPI Error -112 - Program mnemonic too long (see Error Number -112 on page 271)
-1074001039	N2216 SCPI Error -113 - Undefined header (see Error Number -113 on page 271)
-1074001038	N2216 SCPI Error -114 - Header suffix out of range (see Error Number -114 on page 271)
-1074001032	N2216 SCPI Error -120 - Numeric data error (see Error Number -120 on page 271)
-1074001031	N2216 SCPI Error -121 - Invalid character in number (see Error Number -121 on page 272)
-1074001029	N2216 SCPI Error -123 - Exponent too large (see Error Number -123 on page 272)
-1074001028	N2216 SCPI Error -124 - Too many digits (see Error Number -124 on page 272)
-1074001024	N2216 SCPI Error -128 - Numeric data not allowed (see Error Number -128 on page 272)
-1074001022	N2216 SCPI Error -130 - Suffix error (see Error Number -130 on page 272)
-1074001021	N2216 SCPI Error -131 - Invalid suffix (see Error Number -131 on page 272)
-1074001018	N2216 SCPI Error -134 - Suffix too long (see Error Number -134 on page 272)
-1074001014	N2216 SCPI Error -138 - Suffix not allowed (see Error Number -138 on page 272)
-1074001012	N2216 SCPI Error -140 - Character data error (see Error Number -140 on page 272)
-1074001011	N2216 SCPI Error -141 - Invalid character data (see Error Number -141 on page 272)
-1074001008	N2216 SCPI Error -144 - Character data too long (see Error Number -144 on page 272)
-1074001004	N2216 SCPI Error -148 - Character data not allowed (see Error Number -148 on page 272)
-1074001002	N2216 SCPI Error -150 - String data error (see Error Number -150 on page 272)
-1074001001	N2216 SCPI Error -151 - Invalid string data (see Error Number -151 on page 272)
-1074000994	N2216 SCPI Error -158 - String data not allowed (see Error Number -158 on page 272)

Error Number	Description
-1074000992	N2216 SCPI Error -160 - Block data error (see Error Number -160 on page 272)
-1074000991	N2216 SCPI Error -161 - Invalid block data (see Error Number -161 on page 272)
-1074000984	N2216 SCPI Error -168 - Block data not allowed (see Error Number -168 on page 272)
-1074000982	N2216 SCPI Error -170 - Expression error (see Error Number -170 on page 273)
-1074000981	N2216 SCPI Error -171 - Invalid expression (see Error Number -171 on page 273)
-1074000974	N2216 SCPI Error -178 - Expression data not allowed (see Error Number -178 on page 273)
-1074000971	N2216 SCPI Error -181 - Invalid outside macro definition (see Error Number -181 on page 273)
-1074000969	N2216 SCPI Error -183 - Invalid inside macro definition (see Error Number -183 on page 273)
-1074000952	N2216 SCPI Error -200 - Execution error (see Error Number -200 on page 273)
-1074000932	N2216 SCPI Error -220 - Parameter error (see Error Number -220 on page 273)
-1074000931	N2216 SCPI Error -221 - Settings conflict (see Error Number -221 on page 273)
-1074000930	N2216 SCPI Error -222 - Data out of range (see Error Number -222 on page 273)
-1074000929	N2216 SCPI Error -223 - Too much data (see Error Number -223 on page 273)
-1074000928	N2216 SCPI Error -224 - Illegal parameter value (see Error Number -224 on page 273)
-1074000912	N2216 SCPI Error -240 - Hardware error (see Error Number -240 on page 273)
-1074000911	N2216 SCPI Error -241 - Hardware missing (see Error Number -241 on page 273)
-1074000902	N2216 SCPI Error -250 - Mass storage error (see Error Number -250 on page 273)
-1074000901	N2216 SCPI Error -251 - Missing mass storage (see Error Number -251 on page 273)
-1074000900	N2216 SCPI Error -252 - Missing media (see Error Number -252 on page 274)
-1074000899	N2216 SCPI Error -253 - Corrupt media (see Error Number -253 on page 274)
-1074000898	N2216 SCPI Error -254 - Media full (see Error Number -254 on page 274)
-1074000894	N2216 SCPI Error -258 - Media protected (see Error Number -258 on page 274)
-1074000880	N2216 SCPI Error -272 - Macro execution error (see Error Number -272 on page 274)
-1074000879	N2216 SCPI Error -273 - Illegal macro label (see Error Number -273 on page 274)
-1074000876	N2216 SCPI Error -276 - Macro recursion error (see Error Number -276 on page 274)
-1074000875	N2216 SCPI Error -277 - Macro redefinition not allowed (see Error Number -277 on page 274)
-1074000874	N2216 SCPI Error -278 - Macro header not found (see Error Number -278 on page 274)
-1074000842	N2216 SCPI Error -310 - System error (see Error Number -310 on page 274)
-1074000841	N2216 SCPI Error -311 - Memory error (see Error Number -311 on page 274)
-1074000837	N2216 SCPI Error -315 - Configuration memory lost (see Error Number -315 on page 274)
-1074000831	N2216 SCPI Error -321 - Out of memory (see Error Number -321 on page 274)
-1074000822	N2216 SCPI Error -330 - Self-test failed (see Error Number -330 on page 274)
-1074000802	N2216 SCPI Error -350 - Queue overflow (see Error Number -350 on page 274)
-1074000752	N2216 SCPI Error -400 - Query error (see Error Number -400 on page 275)
-1074000742	N2216 SCPI Error -410 - Query INTERRUPTED (see Error Number -410 on page 275)

Error Number	Description
-1074000732	N2216 SCPI Error -420 - Query UNTERMINATED (see Error Number -420 on page 275)
-1074000722	N2216 SCPI Error -430 - Query DEADLOCKED (see Error Number -430 on page 275)
-1074000712	N2216 SCPI Error -440 - Query UNTERMINATED after indefinite response (see Error Number 440 on page 275)
-1074000672	N2216 Device Error Unknown
-1074000671	N2216 Device Error 6201 - Device not open (see Device Error 6201 on page 275)
-1074000670	N2216 Device Error 6202 - Device not ready (see Device Error 6202 on page 275)
-1074000669	N2216 Device Error 6203 - Device already open (see Device Error 6203 on page 275)
-1074000668	N2216 Device Error 6204 - Device incompatible (see Device Error 6204 on page 275)
-1074000667	N2216 Device Error 6205 - Device error (see Device Error 6205 on page 275)
-1074000666	N2216 Device Error 6206 - Session full (see Device Error 6206 on page 275)
-1074000665	N2216 Device Error 6207 - Session busy (see Device Error 6207 on page 275)
-1074000664	N2216 Device Error 6208 - Session empty (see Device Error 6208 on page 275)
-1074000663	N2216 Device Error 6209 - Sequence full (see Device Error 6209 on page 276)
-1074000662	N2216 Device Error 6210 - Sequence busy (see Device Error 6210 on page 276)
-1074000661	N2216 Device Error 6211 - Sequence empty (see Device Error 6211 on page 276)
-1074000660	N2216 Device Error 6212 - Local bus busy (see Device Error 6212 on page 276)
-1074000659	N2216 Device Error 6213 - Require even block count (see Device Error 6213 on page 276)
-1074000658	N2216 Device Error 6214 - Device timeout (see Device Error 6214 on page 276)
-1074000657	N2216 Device Error 6215 - Sequence bus error (see Device Error 6215 on page 276)
-1074000656	N2216 Device Error 6216 - Max safe disk temp exceeded (see Device Error 6216 on page 276)
-1074000655	N2216 Device Error 6217 - Write to a read-only device (see Device Error 6217 on page 276)
-1074000640	Not VXI
-1074000639	Mem allocation failure
-1074000638	NULL pointer detected
-1074000637	reset failed
-1074000636	An unexpected error occurred
-1074000635	ViSession (parmeter 1) was not created by this driver
-1074000634	String not found in table
-1074000633	Instrument Error Detected, call agn2216_error_query()
-1074000632	No SCSI devices found
-1074000631	Invalid SCSI device
-1074000630	Invalid file ID
-1074000629	Could not find a pair of SCSI devices
-1074000628	Too many files open
-1074000627	Thruput file not open

Error Number	Description
-1074000626	Playback scan too large for shared memory
-1074000625	Read size too large for shared memory
-1074000624	Error reading N2216 status
-1074000623	Bad N2216 status
-1074000622	N2216 read error
-1074000621	N2216 write error
-1074000620	Can't map N2216 shared RAM
-1074000619	Error reading N2216 throughput size
-1074000618	Error sending seq:cont
-1074000617	Record size too small
-1074000616	Record size not a 4 byte multiple
-1074000615	Record size exceeds file size
-1074000614	Playback scan too small
-1074000613	Playback scan not a 2 byte multiple
-1074000612	Read size too small
-1074000611	Read size not an element multiple
-1074000610	Volume not found
-1074000609	LIF directory not found
-1074000608	LIF directory entry not found

Sequence Operations Reference

Sequence Overview

What is a Sequence?

Sequence operations are the primary method of transferring data to or from VT2216A Sessions on either the Local Bus or the VXI System Bus. A Sequence is a list of data transfer operations that are performed repeatedly until an entire throughput or playback is complete. Throughput Sequences may contain operations that transfer data from the Local and/or VXI Bus to a Session which consists of one or more SCSI devices. Playback Sequences contain operations that transfer data from the Session to either the Local or the VXI Bus, but not to both. Sequences may also contain synchronization and control operations. See “Using the VT2216A” starting on page 57 for an overview of the VT2216A and explanation of these terms.

Four Sequences may be defined in Sequence memory at any one time but only one can run at a time. An individual Sequence may perform either throughput or playback operations but throughput and playback operations may not be mixed in an individual Sequence. The behavior of a Sequence is undefined if a throughput operation is requested in a playback Sequence or vice versa. The behavior is also undefined if both VXI and Local Bus playback operations are included in a Sequence.

How are Sequences and SCPI related?

Sequences are defined by SCPI commands, but may be run either independently or within a SCPI program. It may noticed that SCPI commands exist that may also be used to throughput and playback data to the VT2216A. In some cases, the data transfer could be accomplished either with SCPI commands (`MMEMory:SESSion:READ:*` and `MMEMory:SESSion:WRITE:*`) or with Sequence operations (LBUS Consume, LBUS Generate, Throughput and Playback). Typically, users find that Sequences provide an easier way (and in some cases the only way) to perform complex throughput and playback operations.

It is best to use Sequences for nearly all throughput operations because of the ability of Sequences to handle large data transfers and multiple devices. SCPI throughput commands are limited to small amounts of data with a single device.

Either SCPI commands or Sequences can be used to playback data that involves all of the data from a Session. However, Sequences must be used for any playback operations that involve only parts of the data (such as one channel from multiple-channel data).

Creating Sequences

Adding Sequence Elements

Sequences are defined by an instrument-specific SCPI subsystem. All the Sequence operations documented in this chapter are implemented by using the SCPI command:

```
SEQuence[1|2|3|4]:ADD <Operation>,<Count>,<Address>,<Misc>
```

Each time this command is issued an element representing one operation is added to the end of the Sequence queue in memory. The maximum number of operations in a single sequence is 100.

It is normally more convenient to define Sequence operations programmatically so that a Sequence may be altered by changing the program and re-executing it. By creating a SCPI program to define the Sequence, the Sequence can be altered more easily.

If the Sequence number [1|2|3|4] is not specified the Sequence element is added to Sequence 1.

Required Fields

Every Sequence element requires that all four fields: (`<Operation>`,`<Count>`,`<Address>`,`<Misc>`) be filled though not every operation uses all fields. For some Sequence operations certain fields represent two pieces of information as indicated in the Sequence operation descriptions.

The `<Operation>` field specifies what type of action will take place: data transfer, synchronization or control. This value corresponds to the code listed in the programming reference section of this chapter for the specific type of operation.

The `<Count>` field is used by many operations to indicate how many units will be transferred. The unit of `<Count>` may be either bytes or blocks, as indicated in the description of each operation. For some Sequence operations, this field represents two pieces of information as indicated in the Sequence operation descriptions.

The `<Address>` field is used mainly by operations that transfer data over the VXI System Bus. The value of `<Address>` is an offset from the beginning of one of the address spaces. The Shared RAM space is local to the VT2216A.

The miscellaneous `<Misc>` field has various meanings depending on the operation.

Accepted Field Values

Field values must be specified as numeric values. All decimal representations, including signs, decimal points and scientific notation are accepted as field values:

```
123, 123E2, -123, -1.23E2, .123, 1.23E-2, 1.23000E-01
```

Note, however, that negative numbers will generate an error and fractional values will be automatically rounded to the nearest integer.

The fields may be specified in decimal, hex, octal, or binary:

```
123, #h7B, #q173, #b1111011
```

Related SCPI commands

Other SCPI commands in addition to SEQuence:ADD that can or must be used with relation to Sequence operations are documented in detail in the SCPI programming section of this book. These commands include:

- MMEMory:SCSI, MMEMory:TUNit and MMEMory:SESSion. These subsystems must be used to initialize the Session before starting a Sequence.
- SEQuence:BEGin starts Sequence execution.
- SEQuence:DELete:ALL deletes all operations from the current Sequence list. This command should be sent before adding elements to a Sequence.
- SEQuence:SIZE? returns the number of elements in the Sequence.

Sequence Quick Reference

Operation	Code (hex)	Count	Address	Misc	Page
Control operations					
Do Nothing	0000	N/A	N/A	N/A	150
Terminate Sequence	0001	N/A	N/A	N/A	151
Pause N msec	0002	Milliseconds	N/A	N/A	152
Pause N loops	000a	Loops	N/A	N/A	161
Execute New Sequence	0004	Seq nbr	N/A	N/A	154
New Sequence If Count	0005	Seq nbr	MSB	LSB	155
TTLTRG Control	0003	Bit field	N/A	N/A	153
TTLTRG Arm	0006	Bit field	N/A	N/A	156
TTLTRG Wait	0007	Bit field	N/A	N/A	157
IRQ Arm	0008	Bit field	N/A	N/A	158
IRQ Wait	0009	Bit field	N/A	N/A	159
Test shared RAM and Skip	7000	N/A	RAM address	Skipped Sequence Op's	160
Local bus throughput operations					
LBUS Eavesdrop	1001	LBUS blocks	N/A	LBUS width-Bytes/block	163
LBUS Consume Pipe	1002	Blks pass Blks consume	N/A	LBUS width-Bytes/block	164
LBUS Eavesdrop Pipe	1003	Blks pass Blks eaves	N/A	LBUS width-Bytes/block	165
LBUS Consume Continuous	1100	LBUS blocks	N/A	LBUS width-Bytes/block	166
LBUS Eavesdrop Continuous	1101	LBUS blocks	N/A	LBUS width-Bytes/block	167
LBUS Consume Pipe Continuous	1102	Blks pass Blks consume	N/A	LBUS width-Bytes/block	168
LBUS Eavesdrop Pipe Continuous	1103	Blks pass Blks eaves	N/A	LBUS width-Bytes/block	169

Operation	Code (hex)	Count	Address	Misc	Page
Local bus playback operations					
LBUS Generate	2000	LBUS blocks	N/A	LBUS width-Bytes/block	170
LBUS Append	2001	LBUS blocks	N/A	LBUS width-Bytes/block	171
VXI bus throughput operations					
Throughput A16 Buff 16	3000	Transfer bytes	A16 address	N/A	172
Throughput A16 Buff D32	3002	Transfer bytes	A16 address	N/A	172
Throughput A24 Buff 16	3003	Transfer bytes	A24 address	N/A	172
Throughput A24 Buff D32	3005	Transfer bytes	A24 address	N/A	172
Throughput A32 Buff 16	3006	Transfer bytes	A32 address	N/A	172
Throughput A32 Buff D32	3008	Transfer bytes	A32 address	N/A	172
Throughput A16 FIFO 16	3009	Transfer bytes	A16 address	N/A	172
Throughput A16 FIFO 32	300A	Transfer bytes	A16 address	N/A	172
Throughput A16 FIFO D32	300B	Transfer bytes	A16 address	N/A	172
Throughput A24 FIFO 16	300C	Transfer bytes	A24 address	N/A	172
Throughput A24 FIFO 32	300D	Transfer bytes	A24 address	N/A	172
Throughput A24 FIFO D32	300E	Transfer bytes	A24 address	N/A	172
Throughput A32 FIFO 16	300F	Transfer bytes	A32 address	N/A	172
Throughput A32 FIFO 32	3010	Transfer bytes	A32 address	N/A	172
Throughput A32 FIFO D32	3011	Transfer bytes	A32 address	N/A	172
Throughput Shared RAM	3012	Transfer bytes	RAM address	N/A	172
Throughput Dummy Bytes	3100	Pad bytes	N/A	N/A	173
VXI bus playback operations					
Playback A16 Buff 16	4000	Transfer bytes	A16 address	N/A	175
Playback A16 Buff D32	4002	Transfer bytes	A16 address	N/A	175
Playback A24 Buff 16	4003	Transfer bytes	A24 address	N/A	175
Playback A24 Buff D32	4005	Transfer bytes	A24 address	N/A	175
Playback A32 Buff 16	4006	Transfer bytes	A32 address	N/A	175
Playback A32 Buff D32	4008	Transfer bytes	A32 address	N/A	175
Playback A16 FIFO 16	4009	Transfer bytes	A16 address	N/A	175
Playback A16 FIFO 32	400A	Transfer bytes	A16 address	N/A	175
Playback A16 FIFO D32	400B	Transfer bytes	A16 address	N/A	175
Playback A24 FIFO 16	400C	Transfer bytes	A24 address	N/A	175
Playback A24 FIFO 32	400D	Transfer bytes	A24 address	N/A	175
Playback A24 FIFO D32	400E	Transfer bytes	A24 address	N/A	175

Operation	Code (hex)	Count	Address	Misc	Page
Playback A32 FIFO 16	400F	Transfer bytes	A32 address	N/A	175
Playback A32 FIFO 32	4010	Transfer bytes	A32 address	N/A	175
Playback A32 FIFO D32	4011	Transfer bytes	A32 address	N/A	175
Playback Shared RAM	4012	Transfer bytes	RAM address	N/A	175
Playback Bit Bucket	4100	Discard bytes	N/A	N/A	176

Local bus throughput operations with monitor

LBUS Consume Monitor Shared RAM	5000	LBUS blocks	RAM address	LBUS width-Bytes/block	177
LBUS Eavesdrop Monitor Shared RAM	5001	LBUS blocks	RAM address	LBUS width-Bytes/block	177
LBUS Consume Pipe Monitor Shared RAM	5002	Blks pass Blks consume	RAM address	LBUS width-Bytes/block	177
LBUS Eavesdrop Pipe Monitor Shared RAM	5003	Blks pass Blks eaves	RAM address	LBUS width-Bytes/block	177
LBUS Consume Monitor A24	5014	LBUS blocks	A24 address	LBUS width-Bytes/block	177
LBUS Eavesdrop Monitor A24	5015	LBUS blocks	A24 address	LBUS width-Bytes/block	177
LBUS Consume Pipe Monitor A24	5016	Blks pass Blks consume	A24 address	LBUS width-Bytes/block	177
LBUS Eavesdrop Pipe Monitor A24	5017	Blks pass Blks eaves	A24 address	LBUS width-Bytes/block	177

VXI throughput operations with monitor

Throughput Shared RAM Monitor Shared RAM	3812	Monitor bytes	RAM address	RAM address	174
Throughput Shared RAM Monitor A24 Buff D32	3912	Monitor bytes	RAM address	A24 address	174
Throughput Shared RAM Monitor A24 Buff	3a12	Monitor bytes	RAM address	A24 address	174
Throughput A16 FIFO D32 Monitor Shared RAM	380b	Monitor bytes	A16 address	RAM address	174
Throughput A16 FIFO D32 Monitor A24 Buff D32	390b	Monitor bytes	A16 address	A24 address	174
Throughput A16 FIFO D32 Monitor A24 Buff	3a0b	Monitor bytes	A16 address	A24 address	174
Throughput A16 FIFO16 Monitor Shared RAM	3809	Monitor bytes	A16 address	RAM address	174
Throughput A16 FIFO16 Monitor A24 Buff D32	3909	Monitor bytes	A16 address	A24 address	174
Throughput A16 FIFO16 Monitor A24 Buff	3a09	Monitor bytes	A16 address	A24 address	174
Throughput A16 Buff 16 Monitor Shared RAM	3800	Monitor bytes	A16 address	RAM address	174
Throughput A16 Buff 16 Monitor A24 BuffD32	3900	Monitor bytes	A16 address	A24 address	174
Throughput A16 Buff 16 Monitor A24 Buff	3a00	Monitor bytes	A16 address	A24 address	174
Throughput A16 Buff D32 Monitor Shared RAM	3802	Monitor bytes	A16 address	RAM address	174
Throughput A16 Buff D32 Monitor A24 Buff D32	3902	Monitor bytes	A16 address	A24 address	174
Throughput A16 Buff D32 Monitor A24 Buff	3a02	Monitor bytes	A16 address	A24 address	174

Operation	Code (hex)	Count	Address	Misc	Page
Throughput A24 FIFO D32 Monitor Shared RAM	380e	Monitor bytes	A24 address	RAM address	174
Throughput A24 FIFO D32 Monitor A24 Buff D32	390e	Monitor bytes	A24 address	A24 address	174
Throughput A24 FIFO D32 Monitor A24 Buff	3a0e	Monitor bytes	A24 address	A24 address	174
Throughput A24 FIFO 16 Monitor Shared RAM	380c	Monitor bytes	A24 address	RAM address	174
Throughput A24 FIFO 16 Monitor A24 Buff D32	390c	Monitor bytes	A24 address	A24 address	174
Throughput A24 FIFO 16 Monitor A24 Buff	3a0c	Monitor bytes	A24 address	A24 address	174
Throughput A24 Buff 16 Monitor Shared RAM	3803	Monitor bytes	A24 address	RAM address	174
Throughput A24 Buff 16 Monitor A24 Buff D32	3903	Monitor bytes	A24 address	A24 address	174
Throughput A24 Buff 16 Monitor A24 Buff	3a03	Monitor bytes	A24 address	A24 address	174
Throughput A24 Buff D32 Monitor Shared RAM	3805	Monitor bytes	A24 address	RAM address	174
Throughput A24 Buff D32 Monitor A24 Buff D32	3905	Monitor bytes	A24 address	A24 address	174
Throughput A24 Buff D32 Monitor A24 Buff	3a05	Monitor bytes	A24 address	A24 address	174
Synchronization operations					
Wait Bit Set A16	6000	Bit mask	A16 address	Loops	179
Wait Bit Clear A16	6001	Bit mask	A16 address	Loops	179
Wait Bit Set A24	6002	Bit mask	A24 address	Loops	179
Wait Bit Clear A24	6003	Bit mask	A24 address	Loops	179
Wait Bit Set A32	6004	Bit mask	A32 address	Loops	179
Wait Bit Clear A32	6005	Bit mask	A32 address	Loops	179
Wait Bit Set Shared RAM	6006	Bit mask	RAM address	Loops	179
Wait Bit Clear Shared RAM	6007	Bit mask	RAM address	Loops	179
Wait A16 Count 16	6008	16-bit value	A16 address	Loops	180
Wait A24 Count 16	6009	16-bit value	A24 address	Loops	180
Wait A32 Count 16	600A	16-bit value	A32 address	Loops	180
Wait Count Shared RAM 16	600B	16-bit value	RAM address	Loops	180
Wait A16 Count 32	600C	32-bit value	A16 address	Loops	180
Wait A24 Count 32	600D	32-bit value	A24 address	Loops	180
Wait A32 Count 32	600E	32-bit value	A32 address	Loops	180
Wait Count Shared RAM 32	600F	32-bit value	RAM address	Loops	180
Wait FIFO Empty	6010	N/A	N/A	N/A	181
Wait FIFO Half Empty	6011	N/A	N/A	N/A	181
Control A16 Reg16	6018	N/A	A16 address	Value	182
Control A24 Reg16	6019	N/A	A24 address	Value	182
Control A32 Reg16	601A	N/A	A32 address	Value	182
Control Reg Shared RAM 16	601B	N/A	RAM address	Value	182

Operation	Code (hex)	Count	Address	Misc	Page
Control A16 Reg 32	601C	N/A	A16 address	Value	182
Control A24 Reg 32	601D	N/A	A24 address	Value	182
Control A32 Reg 32	601E	N/A	A32 address	Value	182
Control Reg Shared RAM 32	601F	N/A	RAM address	Value	182
Dump A24 Seq Bytes	6020	N/A	A24 address	N/A	183
Dump A32 Seq Bytes	6021	N/A	A32 address	N/A	183
Dump Shared RAM Seq Bytes	6022	N/A	RAM address	N/A	183

VT2216A Sequence Operations

Do Nothing 0000

No Sequence operation is performed.

Sequence Syntax:

```
#h0000,<Count>,<Address>,<Misc>
<Count> ::= 0
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0000,0,0,0
```

Description: No fields are used.

Terminate Sequence 0001

The Sequence stops executing.

Sequence Syntax:

```
#h0001,<Count>,<Address>,<Misc>
<Count> ::= 0
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0001,0,0,0
```

Description: This operation terminates the Sequence even if the final count has not been met. This is useful only for creating a non-looping or one-time Sequence. No fields are used.

Pause N msec 0002

The Sequence stops executing for a designated period of time.

Sequence Syntax:

```
#h0002,<Count>,<Address>,<Misc>
<Count> ::= 10:4294967295
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0002,40,0,0
```

Description: This operation causes the Sequence to stop executing for the number of milliseconds designated by <Count>. The resolution of the clock is only 10 ms, therefore the specified count is rounded to the nearest 10 ms value. For a pause of shorter duration see “Pause N loops” on page 161.

<Address> and <Misc> are not used.

TTLTRG Control 0003

Controls the assertion of the TTLTRG lines.

Sequence Syntax:

```
#h0003,<Count>,<Address>,<Misc>
<Count> ::= 0:#b11111111
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0003,#b1010101,0,0
```

Description: All TTLTRG lines are controlled simultaneously. Therefore, one (or more) lines may be set while all others are cleared. Any bits set to 1 in bits 0-7 of <Count> represent corresponding TTLTRG lines that are asserted.

<Address> and <Misc> are not used.

See “TTLTRG Arm” on page 156 and “TTLTRG Wait” on page 157.

Execute New Sequence 0004

Begins executing a new logical Sequence.

Sequence Syntax:

```
#h0004,<Count>,<Address>,<Misc>
<Count> ::= 1:4
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0004,3,0,0
```

Description: This operation begins executing the new logical Sequence specified by <Count>. The new Sequence inherits the Sequence type and total bytes remaining from the currently executing Sequence. This operation is useful in situations that require a one-time set of operations at the beginning of a throughput followed by a looping set of data acquisition operations. An example of such a one-time action is writing a header at the beginning of a data stream.

<Address> and <Misc> are not used.

New Sequence If Count 0005

Begins executing a new logical Sequence if the remaining byte count is less than the value specified.

Sequence Syntax:

```
#h0005,<Count>,<Address>,<Misc>
<Count> ::= 1:4
<Address> ::= 0:#hFFFFFFFF
<Misc> ::= 0:#hFFFFFFFF
```

SCPI example:

```
SEQ:ADD #h0005,2,#hAEC,#h33E1F671
```

Description: This operation begins executing the new logical Sequence number specified by <Count> if the remaining byte count is less than that specified by <Address> and <Misc>. Since the byte count is a 64-bit value and the Sequence fields are only 32-bit values, both the <Address> and <Misc> are used to specify the byte count. The most significant 32 bits are specified in the <Address> field and the least significant 32 bits are specified in the <Misc> field. The new Sequence inherits the Sequence type and total bytes remaining from the currently executing Sequence.

TTLTRG Arm 0006

Clears a set of latched TTLTRG assertions.

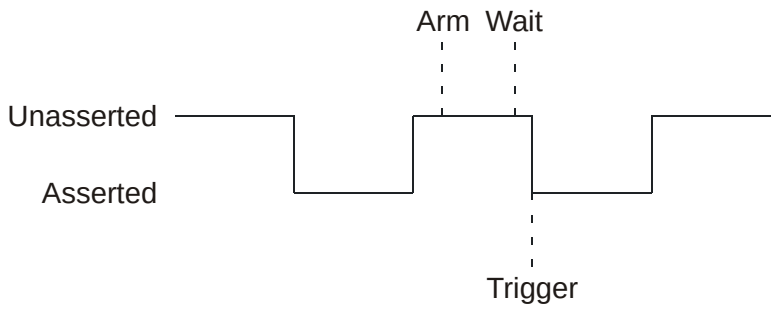
Sequence Syntax:

```
#h0006,<Count>,<Address>,<Misc>
<Count> ::= 0:#b11111111
<Address> ::= 0
<Misc> ::= 0
```

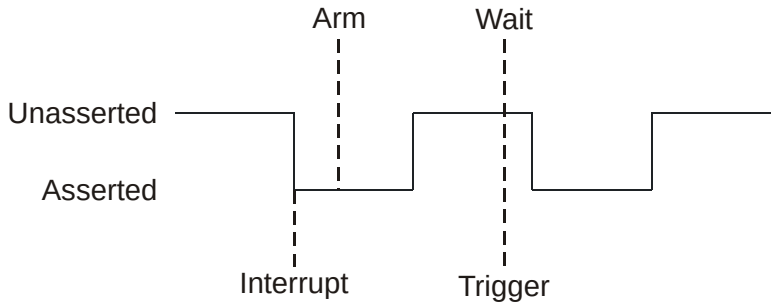
SCPI example:

```
SEQ:ADD #h0006,#b11011101,0,0
```

Description: Clearing latched TTLTRG assertions guarantees that any subsequent TTLTRG Wait will not be satisfied by an old latched TTLTRG assertion. Any bits set to 1 in bits 0-7 of <Count> clear assertions for corresponding to TTLTRG lines. The diagram below illustrates the effect of TTLTRG Arm and TTLTRG Wait on triggering in response to TTLTRG line assertion:



A perceived exception occurs if the trigger line is already asserted (set to the low voltage level) when the TTLTRG arm command is issued. In this case, a subsequent TTLTRG Wait will result in no delay because the assertion requirement was previously fulfilled by the interrupt generated prior to TTLTRG Arm:



<Address> and <Misc> are not used.

See “TTLTRG Control” on page 153 and “TTLTRG Wait” on page 157.

TTLTRG Wait 0007

Waits for a set of TTLTRG lines to be asserted.

Sequence Syntax:

```
#h0007,<Count>,<Address>,<Misc>
<Count> ::= 0:#b11111111
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0007,#b10101010,0,0
```

Description: Because TTLTRG assertions are latched, it is not necessary that all of the specified lines be set at the same time; only that each specified line undergo an unasserted-to-asserted transition since the last TTLTRG Wait or TTLTRG Arm operation. Any bits set to 1 in bits 0-7 of <Count> represent corresponding TTLTRG lines that await assertion.

<Address> and <Misc> are not used.

See "TTLTRG Control" on page 153 and "TTLTRG Arm" on page 156.

IRQ Arm 0008

Clears a set of latched IRQ assertions from a specified logical address.

Sequence Syntax:

```
#h0008,<Count>,<Address>,<Misc>
<Count> ::= 0:255
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0008,86,0,0
```

Description: Clearing latched IRQ assertions guarantees that any subsequent IRQ Wait will not be satisfied by an old latched IRQ assertion. The <Count> field indicates the logical address for which the latched IRQ should be cleared.

<Address> and <Misc> are not used.

IRQ Wait 0009

Waits for IRQ from a specific logical address.

Sequence Syntax:

```
#h0009,<Count>,<Address>,<Misc>
<Count> ::= 1:255
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h0009,222,0,0
```

Description: When a VXI system is configured, each IRQ line is assigned an IRQ Handler. The IRQ Handler may be any device that supports this capability. In order for the VT2216A to proceed after executing the IRQ Wait Sequence operation, it must receive an IRQ from the specific logical address on any IRQ line for which it has been assigned as IRQ Handler. See the VXI Resource Manager documentation to determine how to assign IRQ Handlers.

<Address> and <Misc> are not used.

Test shared RAM and Skip 7000

Execute the next sequence operation if a shared RAM location is non-zero.

Sequence Syntax:

```
#h7000,<Count>,<Address>,<Misc>
<Count> ::= 0
Shared RAM <Address> ::= 0:262142
<Misc> ::= 0:100
```

SCPI example:

```
SEQ:ADD #h7000,0,0,1
```

Description: Read a 16-bit value at the specified shared RAM <Address>. If the value read is zero, skip the next <Misc> number of sequence operations. If the value read is non-zero, set the 16-bit value in shared RAM to zero and execute the next sequence operation.

Pause N loops 000a

The Sequence stops executing for a designated number of loops.

Sequence Syntax:

```
#h0002,<Count>,<Address>,<Misc>
<Count> ::= 1:4294967295
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h000a,10,0,0
```

Description: This operation causes the Sequence to execute a delay loop for the number of repetitions designated by <Count>. This operation may be used to pause a Sequence for a shorter duration of time than may be achieved with the 'Pause N milliseconds' (0002) operation for which the minimum time is 10 milliseconds. A <Count> of 1560927 results in a delay of ≥ 1 second. The actual delay time may be longer due to the unpredictable nature of interrupts.

<Address> and <Misc> are not used.

LBUS Consume 1000

A throughput operation that reads blocks of data from the local bus and writes them to a SCSI Session.

Sequence Syntax: `#h1000,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256`

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1000,8,0,#h03000800`

Description: The LBUS Consume operation puts the local bus chip into a mode that acts as a sink for bytes on the local bus. In other words, no bytes are passed to the next module to the right.

<Count> indicates the number of local bus blocks to transfer.

<Address> is not used.

<Misc> contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

LBUS Eavesdrop 1001

A throughput operation that reads blocks of data from the local bus and writes them to a SCSI Session in addition to passing them along to the next local bus module to the right.

Sequence Syntax: `#h1001,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256`

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1001,2,0,#h3004000`

Description: The LBUS Eavesdrop operation puts the local bus chip into a mode in which each byte received from the module to the left is copied into the VT2216A and is also passed to the next module to the right.

<Count> indicates the number of local bus blocks to transfer.

<Address> is not used.

<Misc> contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

LBUS Consume Pipe 1002

A throughput operation that writes some blocks of local bus data to a SCSI Session while passing other blocks of local bus data to the next module to the right.

Sequence Syntax: `#h1002,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256 1:256` (see description below)

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1002,#h40002,0,#h3008000`

Description: The LBUS Consume Pipe operation reads N blocks of data from the local bus and writes them to a SCSI Session then passes M blocks to the next module to the right without copying those bytes to the VT2216A. This operation is used for high data rate applications that require multiple VT2216A modules.

<Count> has the dual purpose of specifying both the number of blocks to pass (M) as well as the number of blocks to consume (N). This is accomplished by placing M in the most significant 16 bits of the <Count> field and N in the least significant 16 bits of the <Count> field. M and N must have a greatest common denominator (cd) of ≤ 256 where the largest of M/cd and N/cd is ≤ 16.

<Address> is not used.

<Misc> also contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block. The upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

LBUS Eavesdrop Pipe 1003

A throughput operation that writes some blocks of local bus data to a SCSI Session while passing those blocks plus additional blocks of local bus data to the next module to the right.

Sequence Syntax: `#h1003,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256 1:256` (see description below)

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1003,#h20001,0,#h03004000`

Description: The LBUS Eavesdrop Pipe operation reads N blocks of data from the local bus, writes them to a SCSI Session and also passes them to the next module to the right. The operation then passes M blocks to the next module to the right without copying them to the VT2216A. This operation is used for high data rate applications that require multiple VT2216A modules.

<Count> has the dual purpose of specifying both the number of blocks to pass (M) as well as the number of blocks to Eavesdrop (N). This is accomplished by placing M in the most significant 16 bits of the <Count> field and N in the least significant 16 bits of the <Count> field. M and N must have a greatest common denominator (cd) of ≤ 256 where the largest of M/cd and N/cd is ≤ 16.

<Address> is not used.

<Misc> also contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

LBUS Consume Continuous 1100

A throughput operation that reads blocks of data from the local bus and writes them to a SCSI Session.

Sequence Syntax: `#h1100,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256`

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1100,8,0,#h03000800`

Description: The LBUS Consume operation puts the local bus chip into a mode that acts as a sink for bytes on the local bus. In other words, no bytes are passed to the next module to the right.

<Count> indicates the number of local bus blocks to transfer.

<Address> is not used.

<Misc> contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

Note This operation stops executing only when the sequence terminates.

LBUS Eavesdrop Continuous 1101

A throughput operation that reads blocks of data from the local bus and writes them to a SCSI Session in addition to passing them along to the next local bus module to the right.

Sequence Syntax: `#h1101,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256`

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1101,2,0,#h3004000`

Description: The LBUS Eavesdrop operation puts the local bus chip into a mode in which each byte received from the module to the left is copied into the VT2216A and is also passed to the next module to the right.

<Count> indicates the number of local bus blocks to transfer.

<Address> is not used.

<Misc> contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

Note This operation stops executing only when the sequence terminates.

LBUS Consume Pipe Continuous 1102

A throughput operation that writes some blocks of local bus data to a SCSI Session while passing other blocks of local bus data to the next module to the right.

Sequence Syntax: `#h1102,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256 1:256` (see description below)

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1102,#h40002,0,#h3008000`

Description: The LBUS Consume Pipe operation reads N blocks of data from the local bus and writes them to a SCSI Session then passes M blocks to the next module to the right without copying those bytes to the VT2216A. This operation is used for high data rate applications that require multiple VT2216A modules.

<Count> has the dual purpose of specifying both the number of blocks to pass (M) as well as the number of blocks to consume(N). This is accomplished by placing M in the most significant 16 bits of the <Count> field and N in the least significant 16 bits of the <Count> field. M and N must have a greatest common denominator (cd) of ≤ 256 where the largest of M/cd and N/cd is ≤ 16.

<Address> is not used.

<Misc> also contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block. The upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

Note This operation stops executing only when the sequence terminates.

LBUS Eavesdrop Pipe Continuous 1103

A throughput operation that writes some blocks of local bus data to a SCSI Session while passing those blocks plus additional blocks of local bus data to the next module to the right.

Sequence Syntax: `#h1103,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256 1:256` (see description below)

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFF` (see description below)

SCPI example: `SEQ:ADD #h1103,#h20001,0,#h03004000`

Description: The LBUS Eavesdrop Pipe operation reads N blocks of data from the local bus, writes them to a SCSI Session and also passes them to the next module to the right. The operation then passes M blocks to the next module to the right without copying them to the VT2216A. This operation is used for high data rate applications that require multiple VT2216A modules.

<Count> has the dual purpose of specifying both the number of blocks to pass (M) as well as the number of blocks to Eavesdrop (N). This is accomplished by placing M in the most significant 16 bits of the <Count> field and N in the least significant 16 bits of the <Count> field. M and N must have a greatest common denominator (cd) of ≤ 256 where the largest of M/cd and N/cd is ≤ 16.

<Address> is not used.

<Misc> also contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

The bytes-per-block value is used to decrement the bytes-remaining count, thus determining when the final Sequence count has been met. The number of bytes per block must be specified correctly for the Sequence to terminate properly.

Note This operation stops executing only when the sequence terminates.

LBUS Generate 2000

A playback operation that reads blocks of data from a SCSI Session then writes them to the local bus.

Sequence Syntax: `#h2000,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256`

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFC` (see description below)

SCPI example: `SEQ:ADD #h2000,16,0,#h03000c00`

Description: The LBUS Generate operation causes data to flow from the SCSI Session to the next module to the right of the VT2216A. This operation can only be used for local bus playback Sequences.

<Count> indicates the number of local bus blocks to transfer.

<Address> is not used.

<Misc> contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block and must be a multiple of four. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

A block marker is asserted on the local bus following every block-size number of bytes. A frame marker is placed following the last block written to the local bus by this Sequence operation.

LBUS Append 2001

A playback operation that reads blocks of data from a SCSI Session then appends them to the local bus stream of blocks.

Sequence Syntax: `#h2001,<Count>,<Address>,<Misc>`

`<Count> ::= 1:256`

`<Address> ::= 0`

`<Misc> ::= 0:3 #h10:#hFFFC` (see description below)

SCPI example: `SEQ:ADD #h2001,4,0,#h03000800`

Description: The LBUS Append operation causes data to flow from the SCSI Session and appends the data to the end of an LBUS frame as it passes to the next module to the right of the VT2216A. This operation can only be used for local bus playback Sequences.

<Count> indicates the number of local bus blocks to transfer.

<Address> is not used.

<Misc> contains two pieces of information: the lower 24 bits indicate the number of bytes in a local bus block; the upper 8 bits indicate the local bus width. The value indicating the local bus width is presented as the number of bytes minus 1:

Bits 24-31 of Misc parameter		Bits 0-23 of Misc parameter
A local bus width of:	Is represented by a parameter value of:	This value represents the number of bytes in a local bus block and must be a multiple of four. Every local bus block is assumed to be the same size and equal to the count specified here.
8	0	
16	1	
24	2	
32	3	

A block marker is asserted on the local bus following every block-size number of bytes. A frame marker is placed following the last block written to the local bus by this Sequence operation.

Throughput A16 Buff 16 - Throughput Shared RAM 3000-3012

Throughput operations that writes data from a memory buffer or FIFO to a SCSI Session.

Sequence Syntax:

```
#h3000,<Count>,<Address>,<Misc>
```

through

```
#h3012,<Count>,<Address>,<Misc>
<Count> ::= 4:#hFFFFFFFC
A16 <Address> ::= 0:#hFFFE
A24 <Address> ::= 0:#hFFFFFE
A32 <Address> ::= 0:#hFFFFFFFE
Shared RAM <Address> ::= 0:262144
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h300B,#h10000,#hD420,0
```

Notes: See “Sequence Quick Reference” on page 145 for a list of all sixteen operations included in this description.

Description: This description covers sixteen operations for which some essential properties are indicated in the operation name. Buff indicates a memory buffer whereas FIFO refers to reading from the same address as if reading from a FIFO. The buffer or FIFO corresponds to the address space specified in the name: A16, A24, A32, or Shared RAM. The last part of the operation name refers to an access type: a 16-bit access, a 32-bit access implemented as two 16-bit accesses or a D32 access. The D32 access applies only to devices that support D32. Shared RAM is always accessed as a 16-bit buffer.

<Count> designates the number of bytes to transfer and must be a multiple of four.

<Address> designates an offset in the specified memory space (A16, A24, A32, or Shared RAM) at which memory will be accessed. The value must be a multiple of 2.

<Misc> is not used.

Throughput Dummy Bytes 3100

A throughput operation that places dummy bytes in the data stream.

Sequence Syntax:

```
#h3100,<Count>,<Address>,<Misc>
<Count> ::= 0:#hFFFFFFFC
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h3100,#h10000,0,0
```

Description: This operation is used to add padding to certain data structures in the data stream to make it compatible with some post-processing programs that expect data in a certain location.

<Count> designates the number of dummy bytes to place in the data stream and must be a multiple of four.

<Address> and <Misc> fields are not used.

Throughput Shared RAM Monitor Shared RAM - Throughput A24 Buff D32 Monitor A24 Buff 3812-3a05

Throughput operations that perform a VXI bus throughput to a Session while providing a means for the host computer to monitor the data.

Sequence Syntax:

```
#h3812,<Count>,<Address>,<Misc>
```

through

```
#h3a05,<Count>,<Address>,<Misc>
<Count> ::= 4:#hFFFFFFFF
A16 <Address> ::= 0:#hFFFF
A24 <Address> ::= 0:#hFFFFFF
Shared RAM <Address> ::= 0:262144
A24 <Misc> ::= 0:#hFFFFFF
Shared RAM <Misc> ::= 0:262144
```

SCPI example:

```
SEQ:ADD #h3a00,#h200,#h400,#h8000
```

Note: See “Sequence Quick Reference” on page 145 for a list of all 27 operations included in this description.

Description: This description covers 27 operations for which some essential properties are indicated in the operation name:

The address location indicated in the operation name before the word ‘Monitor’ represents the memory location from which to draw data. The address location indicated in the operation name after the word ‘Monitor’ represents the memory location to which to monitor data.

Buff indicates a memory buffer whereas FIFO refers to reading from the same address as if reading from a FIFO. The buffer or FIFO corresponds to the address space specified in the name: A16, A24, A32, or Shared RAM.

<Count> is the number of bytes to monitor

<Address> is the VXI address from which to read data.

<Misc> is the VXI address to which to monitor data

Playback A16 Buff 16 - Playback Shared RAM 4000-4012

Playback operations that write data from a SCSI Session to a memory buffer or FIFO.

Sequence Syntax:

```
#h4000,<Count>,<Address>,<Misc>
```

through

```
#h4012,<Count>,<Address>,<Misc>
<Count> ::= 4:#hFFFFFFFC
A16 <Address> ::= 0:#hFFFE
A24 <Address> ::= 0:#hFFFFFE
A32 <Address> ::= 0:#hFFFFFFFE
Shared RAM <Address> ::= 0:262144
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h400B,#h10000,#hD420,0
```

Note: See “Sequence Quick Reference” on page 145 for a list of all sixteen operations included in this description.

Description: This description covers sixteen operations for which some essential properties are indicated in the operation name. Buff indicates a memory buffer whereas FIFO refers to writing to the same address as if writing to a FIFO. The buffer or FIFO corresponds to the address space specified in the name: A16, A24, A32, or Shared RAM. The last part of the operation name refers to an access type: a 16-bit access, a 32-bit access implemented as two 16-bit accesses or a D32 access. Shared RAM is always accessed as a 16-bit buffer.

<Count> designates the number of bytes to transfer and must be a multiple of four.

<Address> indicates an offset in the specified memory space (A16, A24, A32, or Shared RAM) at which memory will be accessed. This value must be a multiple of 2.

<Misc> is not used.

Playback Bit Bucket 4100

A playback operation that discards bytes from the data stream.

Sequence Syntax:

```
#h4100,<Count>,<Address>,<Misc>
<Count> ::= 4:#hFFFFFFFC
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h4100,#h10000,0,0
```

Description: This operation can be used to playback a single channel from a multiple-channel throughput.

<Count> designates the number bytes to discard and must be a multiple of four.

<Address> and <Misc> are not used.

LBUS Consume Monitor Shared RAM - LBUS Eavesdrop Pipe Monitor A24 5000-5017

Throughput operations that perform a local bus throughput to a Session while providing a means for the host computer to monitor the data via the VXI system bus.

Sequence Syntax:

```
#h5000,<Count>,<Address>,<Misc>
```

through

```
#h5017,<Count>,<Address>,<Misc>
Monitor <Count> ::= 1:256
Pipe Monitor <Count> ::= 1:256 1:256
```
(see description below)

```
A24 <Address> ::= 0:#hFFFFFF
Shared RAM <Address> ::= 0:262144
<Misc> ::= 0:3 #h10:#hFFFF
```
(see description below)

SCPI example:

```
SEQ:ADD #h5016,#h100004,#h400000,#h3008000
```

Note: See “Sequence Quick Reference” on page 145 for a list of all eight operations included in this description.

Description: This description covers eight operations for which some essential properties are indicated in the operation name:

- The part of the name preceding ‘Monitor’ indicates the type of throughput operation and the description corresponds to the description for the same type of operation described earlier in local bus throughput operations.
- In addition, the part of the name following ‘Monitor’ indicates the address space to which the data will be monitored (Monitor Shared RAM or Monitor A24).

The following considerations apply to throughput operations with a Monitor:

- All monitored data is passed into a memory buffer, not into a FIFO.
- All monitoring to Shared RAM is performed via D16 accesses.
- The A16 and A32 address spaces are not supported for monitoring.
- D32 monitoring is not available.

<Count> and <Misc> for monitoring are the same as the <Count> and <Misc> fields for the corresponding local bus throughput operations (1000-1003) described earlier.

<Address> indicates an offset in the specified memory space (A24 or Shared RAM). The memory block at the offset specified in the address space is four bytes larger than the local bus block size multiplied by the number of local bus blocks.

The following tips are applicable to running Sequences using Monitor:

- The act of monitoring a local bus transfer slows down the overall throughput rate because data must be copied to RAM, which would not otherwise be done. The more data monitored, the slower the maximum throughput rate.
- All local bus blocks must be specified that are to be monitored before running the Sequence. Once the Sequence is running, it is not possible to change which blocks to monitor.

- Flags are used to synchronize the host and the VT2216A for monitor operations. The flags are represented by the first four bytes of the memory to which Monitor data is being written, beginning at the address specified in that memory (A24 or Shared RAM). All the flag values must be initialized before running the Sequence. The flag is used to indicate the presence of data in the Monitor block. When the flag is 0, the VT2216A will write data into the block and set the flag to 1. It is expected that the host (or controller) will read the data and then set the flag to 0. If the monitor operation is executed with the flag non-zero, the memory copy will not be done, but the data will flow through the normal data stream to the SCSI Session. This allows the host to read data at a different rate than the actual acquisition of data without affecting the throughput rate. In fact, throughputs will be faster when the flag is set because the memory copy will not need to be done.

  The intention of monitoring many channels is that there will be a Sequence operation to monitor one block for each of the many channels. The flag values will initially be set to non-zero which means that no data will be copied to memory. When monitoring other local bus blocks is desired, the flag can be cleared, allowing data to be written to that monitor block. Upon seeing the 0 flag, the VT2216A will write data to that block and then set the flag indicating that a block of data is available. This scheme allows for changing which local bus blocks are being monitored during the throughput.

Wait Bit Set A16 - Wait Bit Clear Shared RAM 6000-6007

Synchronization operations that can be used to wait for data to be available from a device that generates data slower than the VT2216A can transfer it.

Sequence Syntax:

```
#h6000,<Count>,<Address>,<Misc>
```

through

```
#h6007,<Count>,<Address>,<Misc>
<Count> ::= 0:#b1111111111111111
A16 <Address> ::= 0:#hFFFF
A24 <Address> ::= 0:#hFFFFFF
A32 <Address> ::= 0:#hFFFFFFFF
Shared RAM <Address> ::= 0:262144
<Misc> ::= 0:#hFFFFFFFF
```

SCPI example:

```
SEQ:ADD #h6002,0,#h380024,#b1000
```

Note: See “Sequence Quick Reference” on page 145 for a list of all eight operations included in this description.

Description: These operations wait for a single bit or a group of bits to be set or cleared in another device. This description covers eight operations for which some essential properties are indicated in the operation name. Each reference is to a 16-bit value and performs a D16 access to the memory location specified. Both the memory space referenced and whether to wait for the bit(s) to be set or cleared are indicated in the name of the operation.

<Count> specifies a bit mask that is AND’ed with the 16-bit register specified by the memory offset and memory space. For the Set operations, all bits in the mask must be set. For the Clear operations, all bits in the mask must be clear. The VT2216A reads this register and checks the bits until the condition is met. If the condition is never met the Sequence will not be completed.

<Address> indicates the offset into the memory space indicated in the operation name.

<Misc> represents a user-programmable delay prior to the next VXI access. The number of loops specified here is performed before another VXI access. This frees the VXI bus to perform additional activities, rather than having Sequence operations completely dominate VXI bus usage. Loop time is approximately 3 μs.

Wait A16 Count16 - Wait Count Shared RAM 32 6008-600f

Synchronization operations that can be used to wait for data to be available from a device that generates data slower than the VT2216A can transfer it.

Sequence Syntax:

```
#h6008,<Count>,<Address>,<Misc>
```

through

```
#h600f,<Count>,<Address>,<Misc>
16 bit <Count> ::= 1:#hFFFF
32 bit <Count> ::= 1:#hFFFFFFFF
A16 <Address> ::= 0:#hFFFF
A24 <Address> ::= 0:#hFFFFFF
A32 <Address> ::= 0:#hFFFFFFFF
Shared RAM <Address> ::= 0:262144
<Misc> ::= 0:#hFFFFFFFF
```

SCPI example:

```
SEQ:ADD #h600B,#h4000,128000,0
```

Note: See “Sequence Quick Reference” on page 145 for a list of all eight operations included in this description.

Description: These operations wait for the count register in another device to be greater than the value specified by <Count>. This description covers eight operations for which some essential properties are indicated in the operation name. The memory space referenced is indicated in the name of the operation (A16, A24, A32, or Shared RAM). The count register may be a 16-bit or a 32-bit value as indicated by the name of the operation, but all accesses are done using D16 (a 32-bit count will be performed by using two 16-bit accesses).

<Count> specifies the number that must be met or exceeded before proceeding. The VT2216A reads this register and checks the count until the condition is met. If the condition is never met, the Sequence will not be completed.

<Address> indicates the offset into the memory space indicated in the operation name.

<Misc> represents a user-programmable delay prior to the next VXI access. The number of loops specified here is performed before another VXI access. This frees the VXI bus to perform additional activities, rather than having Sequence operations completely dominate VXI bus usage. Loop time is approximately 3 μs.

Wait FIFO Empty
Wait FIFO Half Empty 6010-6011

The Sequence stops executing until data in the VT2216A FIFO has been depleted.

Sequence Syntax:

```
#h6010,<Count>,<Address>,<Misc>
```

and

```
#h6011,<Count>,<Address>,<Misc>
<Count> ::= 0
<Address> ::= 0
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h6010,0,0,0
```

Description: These operations wait for the VT2216A FIFO (131072 bytes in size) to be either half or completely empty. These operations may be used to synchronize with a device that does not have a FIFO but is able to burst large amounts of data very quickly. In this case, it is necessary to wait for the VT2216A FIFO to be (half) empty before reading data from the device.

As an example, these operations may be necessary to synchronize with the Agilent/HP E1485C in conjunction with a Control Register operation.

Control A16 Reg 16 - Control Reg Shared RAM 32 6018-601f

Synchronization operations that allow the VT2216A to write directly to a memory location.

Sequence Syntax:

```
#h6018,<Count>,<Address>,<Misc>
```

through

```
#h601F,<Count>,<Address>,<Misc>
<Count> ::= 0
A16 <Address> ::= 0:#hFFFF
A24 <Address> ::= 0:#hFFFFFF
A32 <Address> ::= 0:#hFFFFFFFF
Shared RAM <Address> ::= 0:262144
<Misc> ::= #h0:FFFFFFFF
```

SCPI example:

```
SEQ:ADD #h601A,0,#h2D860860,#hF020
```

Note: See “Sequence Quick Reference” on page 145 for a list of all eight operations included in this description.

Description: These operations allow the VT2216A to write to a memory location, usually for the purpose of controlling another device on the bus. The register can be either 16-bits wide or 32-bits wide (a 32-bit register is written as two D16-bit writes). The memory space and register width are indicated in the Sequence name.

<Count> is not used.

<Address> indicates the offset into the memory space indicated in the operation name.

<Misc> contains the value that is to be written to the memory location specified by the memory space and the memory offset in the <Address> field.

Dump A24 Seq Bytes - Dump Shared RAM Seq Bytes 6020-6022

Writes to a memory space the number of bytes that have been transferred.

Sequence Syntax:

```
#h6020,<Count>,<Address>,<Misc>
```

through

```
#h6022,<Count>,<Address>,<Misc>
<Count> ::= 0
A24 <Address> ::= 0:#hFFFFFF
A32 <Address> ::= 0:#hFFFFFFFF
Shared RAM <Address> ::= 0:262144
<Misc> ::= 0
```

SCPI example:

```
SEQ:ADD #h6022,0,#h100,0
```

Note: See “Sequence Quick Reference” on page 145 for a list of all three operations included in this description.

Description: During both throughput and playback Sequences an internal counter keeps a count of how many bytes have been transferred. The contents of this counter may be written to a memory space in order to monitor progress of a Sequence.

<Address> indicates the offset into the memory space indicated in the operation name.

<Count> and <Misc> are not used.

Programming Using SCPI

Getting Started

SCPI (Standard Commands for Programmable Instruments) is an industry-standard instrument control language. SCPI builds on the IEEE 488.1 and 488.2 standards.

Message-based VXI devices

SCPI Command Structure and Format

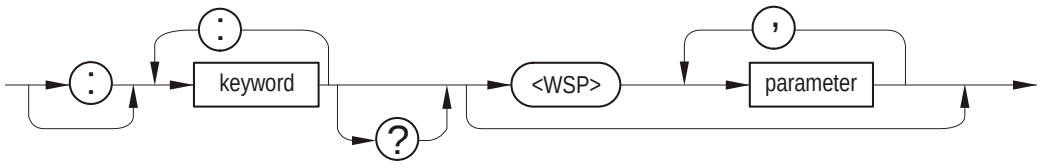
SCPI organizes related functions by grouping them together on a common branch of a command tree. Each branch is assigned a keyword to indicate the nature of the related functions. For example, the functions that control and monitor the status registers are grouped under the STATUS branch of the command tree. The STATUS branch is only one of the major SCPI branches that are called subsystems.

Colons indicate branching points on the command tree. A parameter is separated from the rest of the command by a space.

Multiple commands can be sent within a single message by separating commands with semicolons. One of the main functions of the command parser is to keep track of a program message's position in the command tree. If a program message contains two commands separated by a semicolon, the command parser assumes that the keywords of the second command come from the same branch of the tree as the final keyword of the preceding command. In this manner, multiple command program messages can be simplified.

Another way to simplify program messages is to delete implied mnemonics. Some keywords can be omitted from the command without changing the effect of the command. Implied mnemonics are identified by brackets [ ] in SCPI syntax diagrams.

The illustration below describes the basic syntax of SCPI commands.



NOTE:
WSP = whitespace. ASCII character (Decimal 0-9 or 11-32)

Parameter Settings

As the illustration shows, there must be a <WSP>, whitespace or <space>, between the last command keyword and the first parameter in a command. This is one of the few places in SCPI where <space> is required. If more than one parameter is sent with a single command, a comma must separate the adjacent parameters.

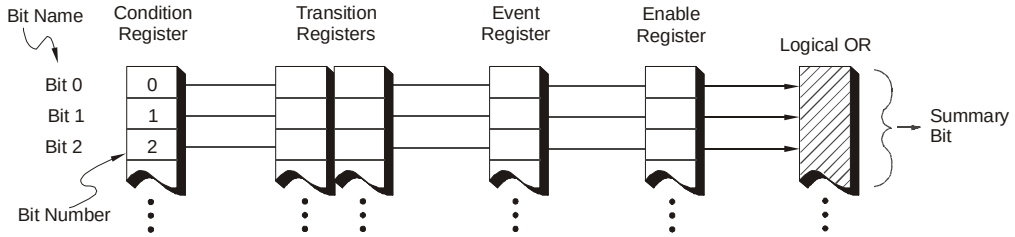
Each parameter format has one or more corresponding response-data formats. For example, a setting programed using a numeric parameter would return either floating point or integer response data when queried. Whether floating point or integer response data is returned, depends on the particular VXI module being used. However, response data is clearly defined for the module and query. The next chapter, “SCPI Command Reference ” specifies the data format for individual commands.

Using the Status Registers

The VT2216A's status registers contain information about various module conditions. The following sections describe the registers and explains how to use them in programs.

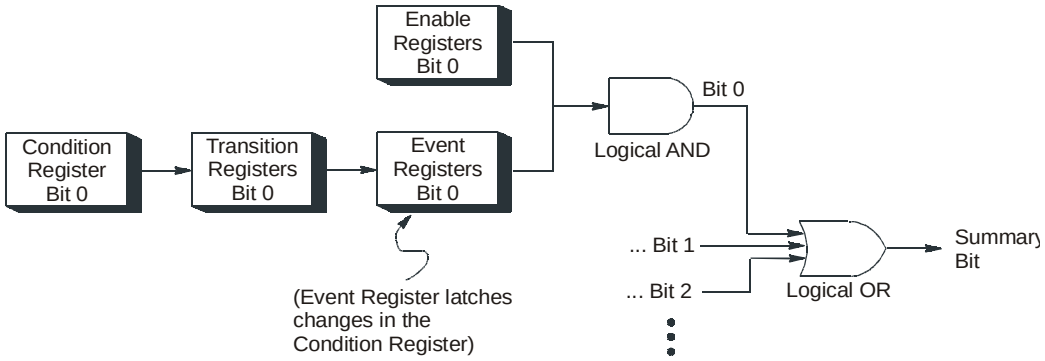
The General Status Register Model

The general status register model, shown below, is the building block of the VT2216A's status system. Most register sets in the module include all of the registers shown in the general model, although commands are not always available for reading or writing a particular register. The model consists of a condition register, two transition registers, an event register and an enable register.



The flow within a status group starts at the condition register and ends at the register summary bit. (See the illustration below.) Flow is controller by altering bits in the enable and transition registers.

The Operation Status and Questionable Status groups are 16 bits wide, while the Status Byte and Standard Event groups are 8 bits wide. In the 16-bit groups, the most significant bit (bit 15) is not used. Bit 15 is always set to 0.



Condition Register

The condition register continuously monitors hardware and firmware status. It represents the current state of the module. It is updated in real time. When the condition monitored by a particular bit becomes true, the bit is set to 1. When the condition becomes false, the bit is reset to 0. Condition registers are read-only.

If there is no command to read a particular condition register, it is simply transparent.

The Transition Registers

The positive and negative transition registers specify which type of bit transition in the Condition register will set corresponding bits in the Event register. Transition register bits may be set for positive transitions (0 to 1) or negative transitions (1 to 0).

Each bit set in the negative transition register indicates that a 1 to 0 transition of that bit in the Condition register sets the associated bit in the Event register. Each bit set in the positive transition register indicates that a 0 to 1 transition of that bit in the Condition register sets the associated bit in the Event register. Setting the same bits in both the positive and negative transition registers indicates that any transition of those bits in the Condition register sets corresponding bits in the Event register.

Event Register

The event register records condition changes. When a change occurs in the condition register, the corresponding event bit is set to 1 in accordance with the transition register settings. Once set, an event bit is no longer affected by condition changes and subsequent events corresponding to that bit are ignored. The event bit remains set until the event register is cleared— either when the register is read or when the *CLS (clear status) command is sent. Event registers are read-only.

Note

Reading the Event Register, clears the Event Register.

Enable Register

The enable register specifies which bits in the event register set a summary bit to 1. The module logically ANDs corresponding bits in the event and enable registers and OR's all the resulting bits to determine the state of a summary bit. Summary bits are in turn recorded in the Status Byte. (The summary bit is only set to 1 if one or more enabled event bits are set to 1.) Enable registers are read-write.

Enable registers are cleared 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 register set.

How to Use Registers

There are two methods which can be used to access the information in register sets:

- The polling method
- The service request (SRQ) method

Use the polling method when:

- The language/development environment does not support SRQ interrupts.

- A simple, single-purpose program is desured and the added complexity of setting up an SRQ handler is not.

Use the SRQ method when:

- Time-critical notification of changes are needed.
- Monitoring more than one device that supports SRQ.
- Having the controller do something else while it is waiting is needed.
- The performance penalty inherent to polling cannot be afforded.

The Polling Method

In the polling method, the module has a passive role. It only tells the controller that conditions have changed when the controller asks the right question. In the SRQ method, the module notifies the controller of a condition change without the controller asking. Either method allows one or more conditions to be monitored.

When monitoring a condition with the polling method, one must:

1. Determine which register contains the bit that monitors the condition.
2. Send the unique SCPI query that reads that register.
3. Examine the bit to see if the condition has changed.

The polling method works well if it is not necessary to know about changes the moment they occur. The SRQ method is more effective if knowing when a condition changes immediately is a must. To detect a change in a condition using the polling method, a program would need to continuously read the registers at very short intervals, which makes the program less efficient. In this case, it is better to use the SRQ method.

The SRQ Method

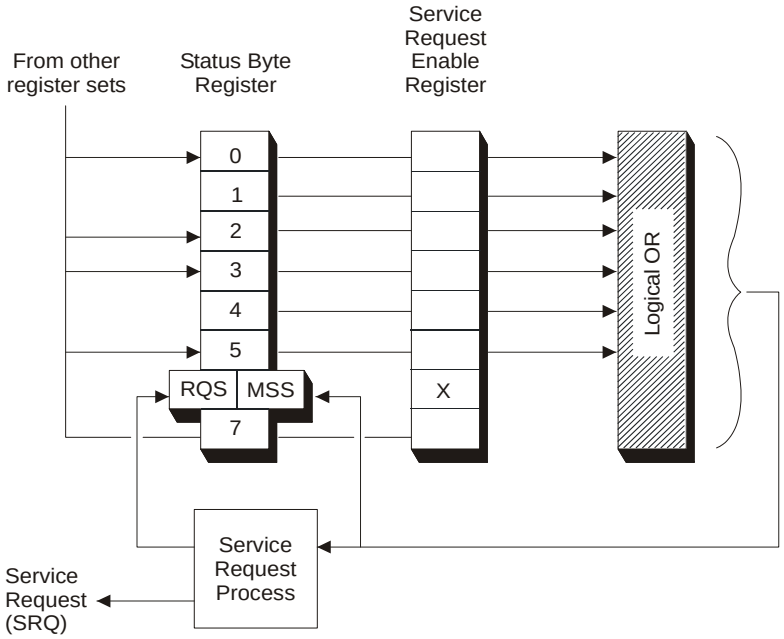
When monitoring a condition with the SRQ method, one must:

1. Determine which bit monitors the condition.
2. Determine how that bit reports to the request service (RQS) bit of the Status Byte.
3. Send SCPI commands to enable the bit that monitors the condition and to enable the summary bits that report the condition to the RQS bit.
4. Enable the controller to respond to service requests.

When the condition changes, the module sets its RQS bit and generates an SRQ. The controller is informed of the change as soon as it occurs. The time the controller would otherwise have used to monitor the condition can now be used to perform other tasks. The program determines how the controller responds to the SRQ.

Generating a Service Request

To use the SRQ method, it is necessary to understand how service requests are generated. As shown below, other register sets in the module report to the Status Byte. Many of them report directly, but some may report indirectly.



Bit 6 of the Status Byte serves two functions: the request service function (RQS) and the master summary status function (MSS). The RQS bit changes whenever something changes that it is configured to report. The RQS bit is cleared when it is read with a serial poll. The MSS bit is set in the same way as the RQS bit. However, the MSS bit is cleared only when the condition that set it is cleared. The MSS bit is read with *STB?.

When a register set causes its summary bit in the Status Byte to change from 0 to 1, the module can initiate the service request (SRQ) process. However, the process is only initiated if both of the following conditions are true:

- The corresponding bit of the Service Request enable register is also set to 1.
- The module does not have a service request pending. (A service request is considered to be pending between the time the module's SRQ process is initiated and the time the controller reads the Status Byte register with a serial poll.)

The SRQ process generates an SRQ. It also sets the Status Byte's request service (RQS) bit to 1. Both actions are necessary to inform the controller that the module requires service. Generating an SRQ only informs the controller that some device on the bus requires service. Setting the RQS bit allows the controller to determine which device requires service. That is, it tells the controller that this particular device requires service.

If the program enables the controller to detect and respond to service requests, it should instruct the controller to perform a serial poll of all modules when an SRQ is generated. Each device on the bus returns the contents of its Status Byte register in response to this poll. The device whose RQS bit is set to 1 is the device that requested service.

Note

When reading the module's Status Byte with a serial poll, the RQS bit is reset to 0. Other bits in the register are not affected.

The VT2216A Registers Sets

The VT2216A uses four register sets to keep track of the module's status:

- Status Byte
- Questionable Status
- Standard Event
- Operational Status

Their reporting structure is summarized in the illustration below. They are described in greater detail in the following sections.

Register bits not explicitly presented in the following sections are not used in the VT2216A. A query to one of these bits returns a value of 0.

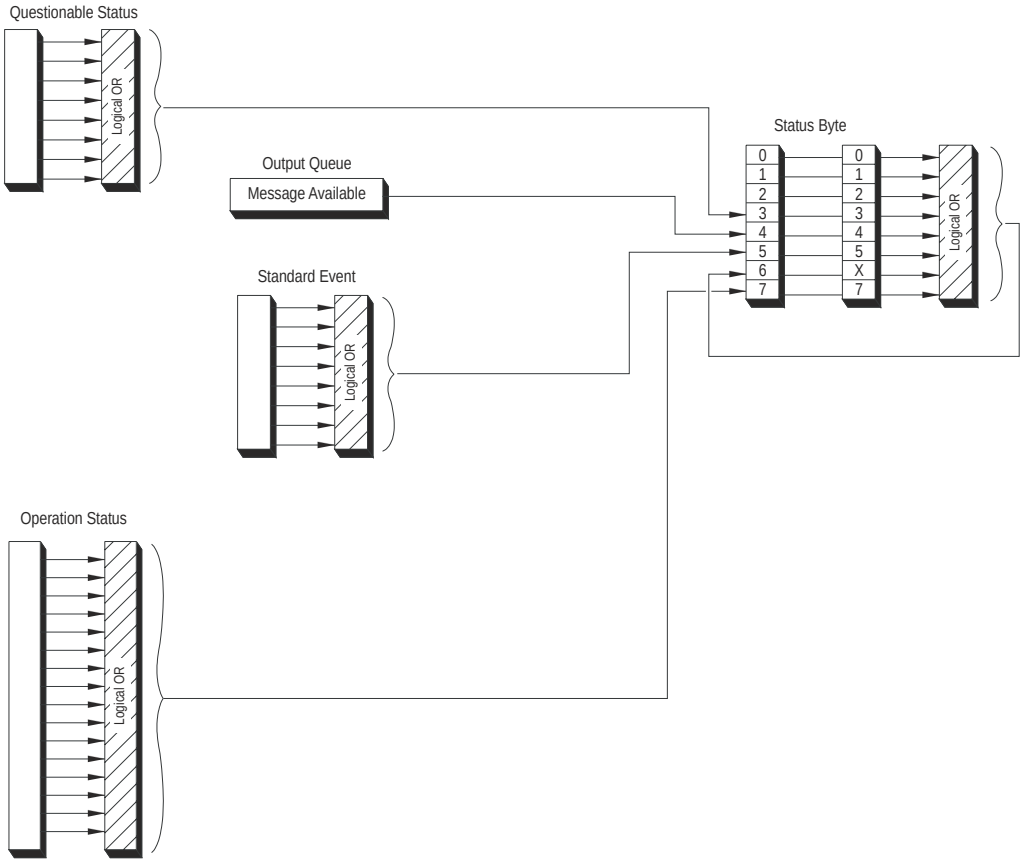
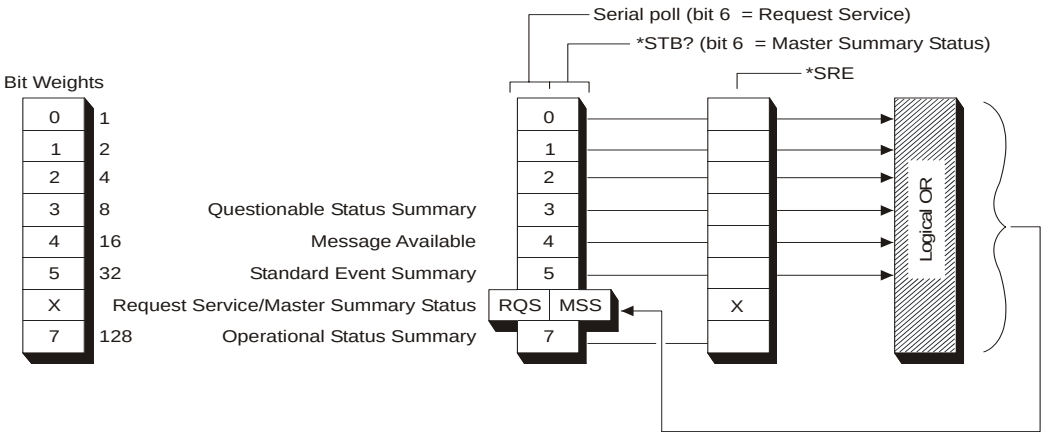


Figure 29 VT2216A Register Sets

Status Byte

The Status Byte summarizes the states of the other register sets and monitors the VT2216A's output queue. It is also responsible for generating service requests (see Generating a Service Request on page 190).



The Status Byte is unique because it does not exactly conform to the general status model presented earlier. It contains only two registers: the Status Byte register and the Service Request Enable register. The Status Byte registers behaves like a condition register for all bits except bit 6. The Service Request enable behaves like a standard enable register except that bit 6 is always set to 0.

Bits in the Status Byte register are set to 1 under the following conditions:

- Questionable Status Summary (bit 3) is set to 1 when one or more enabled bits in the Questionable Status event register are set to 1.
- Message Available (bit 4) is set to 1 when the output queue contains a response message.
- Standard Event Summary (bit 5) is set to 1 when one or more enabled bits in the Standard Event event register are set to 1.
- Master Summary Status (bit 6, when read by *STB?) is set to 1 when one or more enabled bits in the Status Byte register are set to 1.
- Request Service (bit 6, when read by serial poll) is set to 1 by the service request process (see Generating a Service Request on page 190).
- Operation Status Summary (bit 7) is set to 1 when one or more enabled bits in the Operation Status event register are set to 1.

The illustration also shows the commands used to read and write to the Status Byte registers. The following statements are example commands using the Status Byte and Status Byte enable register.

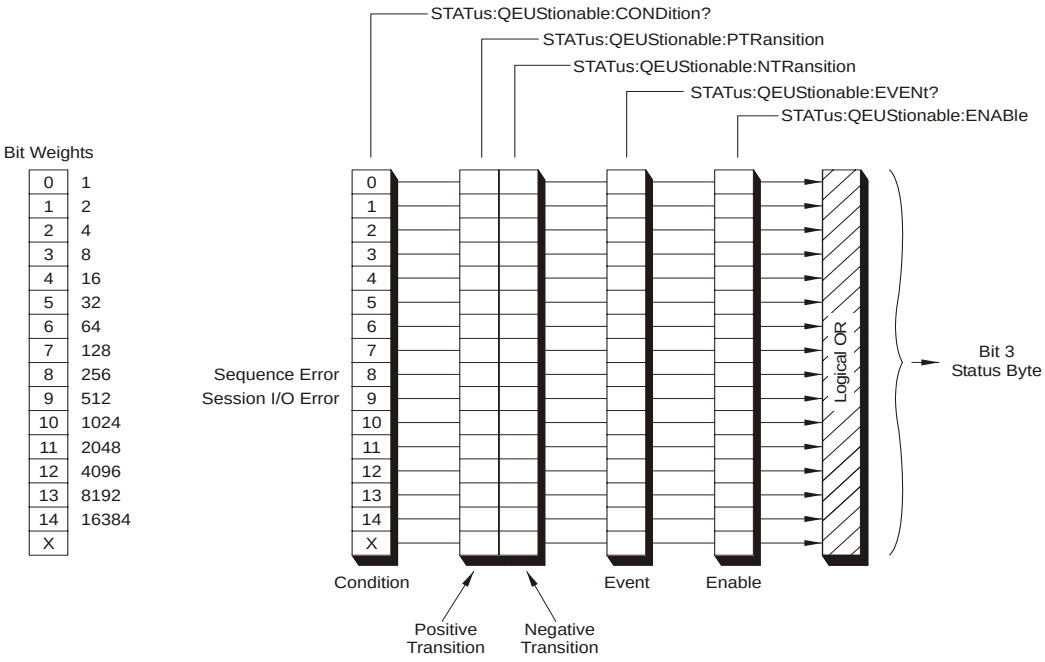
*SRE 16	Generate an SRQ interrupt when messages are available in the output queue.
*SRE?	Find out what events are enabled to generated SRQ interrupts.
*STB?	Read the Status Byte event register.

See Setting and Querying Registers on page 196 for more information about these commands.

Questionable Status Register Set

The Questionable Status register monitors conditions that affect the quality of the data transfer.

This register set includes a condition register, two transition registers, an event register and an enable register. It is accessed through the STATUS subsystem. See Setting and Querying Registers on page 196 for more information about using these commands.



The Condition Register

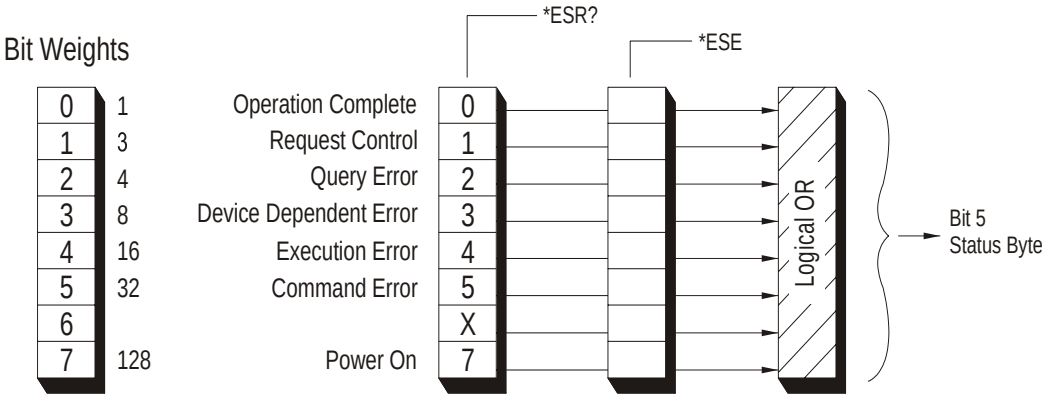
Bits in the Questionable Status condition register are set to 1 under the following conditions:

- Sequence Error (bit 8) is set to 1 when an error is detected during Sequence execution
- Session I/O Error (bit 9) is set to 1 when an error is detected during Session I/O operation

The illustration shows the commands used to read and write to the Questionable Status registers.

Standard Event Status Register Set

The Standard Event Status register set monitors module errors as shown below. It is one of the simplest and most frequently used. The unique aspect of this group is that it is programed by using common commands, while other register sets are programed through the STATUS subsystem.



The Standard Event Status Register set does not conform to the general status register model described at the beginning of this chapter. It contains only two registers: the Standard Event Status Event register and the Standard Event Status Enable register.

Bits in the Standard Event Status event register are set to 1 under the following conditions:

- Operation Complete (bit 0) is set to 1 when the following two events occur (in the order listed):
  - The *OPC command is sent to the module.
  - The module completes all pending overlapped commands.
- Query Error (bit 2) is set to 1 when the module detects a query error.
- Device Dependent Error (bit 3) is set to 1 when the command parser or execution routines detect a device-dependent error.
- Execution Error (bit 4) is set to 1 when the command parser or execution routines detect an execution error.
- Command Error (bit 5) is set to 1 when the command parser detects a command or syntax error.
- Power On (bit 7) is set to 1 when the module is turned on.

The illustration also shows the commands used to read and write to the Standard Event Status register sets. Example commands using Standard Event Status registers:

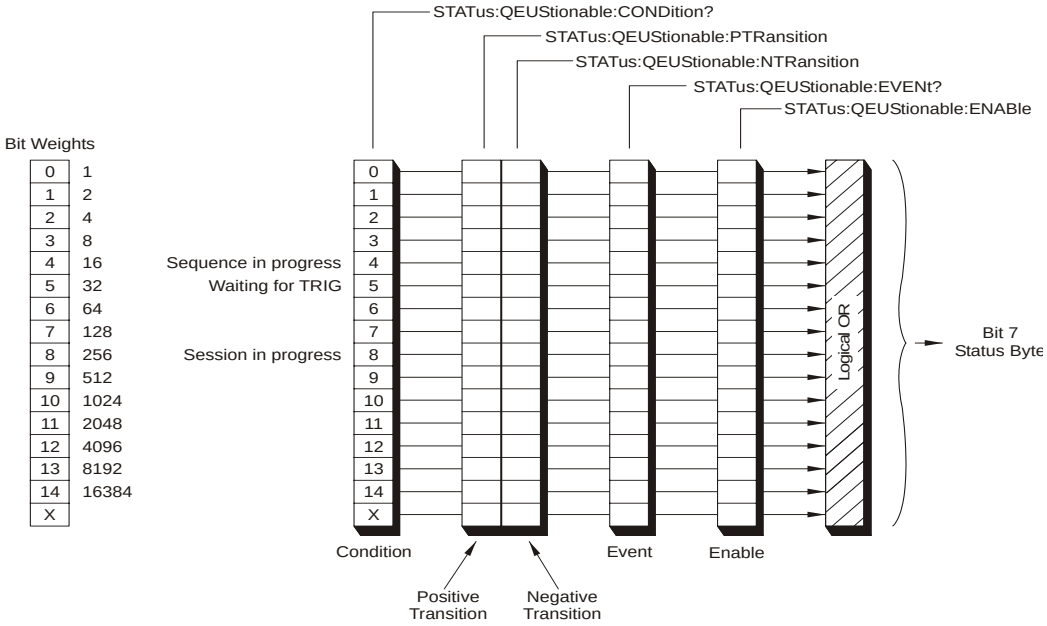
*ESE 48	Generate a summary bit whenever there is an execution or command error
*ESE?	Query the state of the Standard Event Status enable register?
*ESR?	Query the state of the Standard Event Status event register.

See Setting and Querying Registers on page 196 for more information about using these commands.

Operation Status Register Set

The Operation Status register set monitors conditions in the module's data transfer process.

This register set includes a condition register, two transition registers, an event register and an enable register. It is accessed through the STATUS subsystem. See Setting and Querying Registers on page 196 for more information about using these commands.



Bits in the Operation Status condition register are set to 1 under the following conditions:

- Sequence in Progress (bit 4) is set to 1 while a Sequence is in progress and to 0 when the Sequence has finished.
- Waiting for TRIG (bit 5) is set to 1 when the module is ready to accept a trigger signal from one of the trigger sources. (If a trigger signal is sent before this bit is set, the signal is ignored.)
- Session in Progress (bit 8) is set to 1 while a Session is in progress and to 0 when a Session is has finished.

The illustration shows the commands used to read and write to the Operation Status registers.

Setting and Querying Registers

The previous register set illustrations include the commands used to read from and to write the registers. Most commands have a *set form* and a *query form*.

Use the set form of the command to write to a register. The set form is shown in the illustrations. The set form of a command takes an extended numeric parameter.

Use the query form of the command to read a register. Add a "?" to the set form to create the query form of the command. Commands ending with a "?" in the illustrations are query-only commands. These commands cannot set the bits in the register, they can only query or read the register.

The register set illustrations also include the bit weights used to specify each bit in the register. For example, to get the Waiting for Trigger condition register (bit 5 in Operation Status register set) to generate a service request, send the following commands:

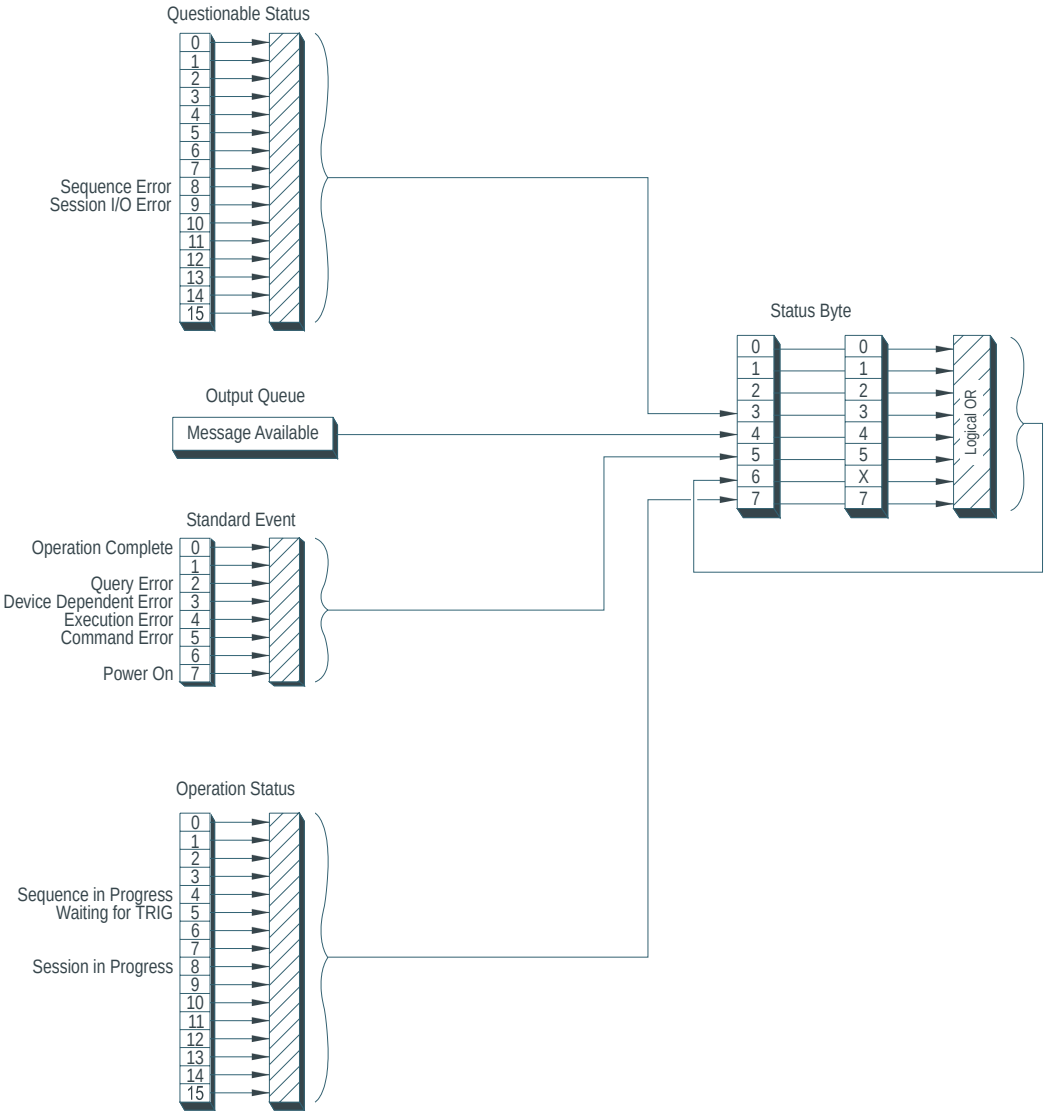
STATUS:PRESET Sets the Enable register bits in the Operational Status and the Questionable Status register sets to 0.

STATUS:OPERATION:ENABLE 32 Sets the Waiting for Trigger Enable register (bit 5) to 1.

***SRE 128** Sets bit 7 of the Service Request Enable register to 1.

See the next chapter for more information about these commands.

VT2216A Register Set Summary



Addressing the VT2216A

The VT2216A address in a SCPI environment consist of three parts; an interface select code, the primary address and the secondary address.

The interface select code specifies the interface. Seven is a typical number for the GPIB interface.

The primary address, typically 09, indicates which GPIB port in the system controller is used to communicate with the Slot 0 Control Module, for example the Agilent/HP E1406A.

The secondary address indicates the device-specific address. In this case, it represents the VXI logical address.

The VXI logical address ranges in value from 1 to 255. For use with a command module, the logical address to the GPIB cannot be used directly, but must be encoded into the GPIB secondary address. In addition, the logical address of the VT2216A must be a multiple of eight, not including 0.

If the logical address is 8, the GPIB secondary address is encoded to 1. If the logical address is 40, the secondary address is encoded to 5. In these examples, the GPIB address is 70901 and 70905.

Software running in a computer writing to a Slot 0 Control Module needs to use all three addresses: the select code, the primary GPIB address and the secondary address.

If a command module is not used, but rather another type of controller such as MXI, V382, or V743, the divide by 8 restriction for the VXI logical address does not apply and any logical address from 1 to 255 may be used.

SCPI Command Reference

Message-based VXI devices

The Command Reference chapter describes all of the VT2216A's SCPI commands. Each command has the following:

1. The heading. This includes two fields. The field to the left shows the command name. The field to the right indicates whether the command has a command form, a query form, or both.
2. A brief description of the command. This one- or two-line description appears just below the heading.
3. A syntax description. This may consist of one or two parts: only a command syntax, only a query syntax or both. The syntax description shows the syntax expected by the command parser. A detailed description for the elements appearing in the syntax description follows.
4. Example statements. This field appears at the end of the syntax description. It contains two examples of BASIC output statements that use the command.
5. A return format description. This field is only used if the command has a query form. It describes how data is returned in response to the query.
6. An attribute summary. This field defines the command's preset state, identifies overlapped commands requiring synchronization and specifies compliance with SCPI. A "confirmed" command complies with SCPI 1994.
7. A detailed description. This field contains additional information about the command.

Finding the Right Command

- If a command cannot be found where it is expected, try scanning the VT2216A SCPI Quick Reference tables that begin on page 204 for the equivalent command that contains the implied mnemonic.

  Each command has a brief description. After equivalent command is located, a more detailed description in the command reference can be found.

- If searching for a command that accesses a particular function, use the index.

  For example, to find the commands that open or close a Transfer Unit, look for “transfer unit” in the index. It lists the pages that describe the MMEMory:TUNit[1|2|...|15]:OPEN and MMEMory:TUNit[1|2|...|15]:CLOSe commands.

Command Syntax

This section describes the syntax elements used in the SCPI command reference. It also describes the general syntax rules for both kinds of command and query messages.

Special Syntactic Elements

Several syntactic elements have special meanings:

- colon (:) — When a command or query contains a series of keywords, the keywords are separated by colons. A colon immediately following a keyword tells the command parser that the program message is proceeding to the next level of the command tree. A colon immediately following a semicolon tells the command parser that the program message is returning to the base of the command tree.
- semicolon (;) — When a program message contains more than one command or query, a semicolon is used to separate them from each other. For example, to set up Session devices to begin at a specified block number and then start a measurement using one program message, the message would be:

  ```
  MMEMory:SESSion2:SEEK 8191;:SEQuence:BEGin VPL,262144,1
  ```

- comma (,) — A comma separates the data sent with a command or returned with a response. For example, the SEQuence:BEGin command requires three values to determine the destination, size and data source of a Sequence that is to be executed. For example, a message to begin a playback to the local bus of 8388608 bytes from Session 3 would be:

  ```
  SEQuence:BEGin LPL,8388608,3
  ```

- <WSP> — One white space is required to separate a program message (the command or query) from its parameters. For example, the command “SEQuence:BEGin VPL,262144,1” contains a space between the program header (SEQuence:BEGin) and its program data (VPL,262144,1). White space characters are not allowed within a program header.

Conventions

Syntax and return format descriptions use the following conventions:

- < > Angle brackets enclose the names of items that need further definition. The definition will be included in accompanying text. In addition, detailed descriptions of these elements appear at the end of this section.
- ::= “is defined as” When two items are separated by this symbol, the second item replaces the first in any statement that contains the first item. For example, A::=B indicates that B replaces A in any statement that contains A.
- | “or” When items in a list are separated by this symbol, one and only one of the items can be chosen from the list. For example, A|B indicates that A or B can be chosen, but not both.
- ... An ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more times.
- [ ] Square brackets indicate that the enclosed items are optional.
- { } Braces are used to group items into a single syntactic element. They are most often used to enclose lists and to enclose elements that are followed by an ellipsis.

Although the command interpreter is not case sensitive, the case of letters in the command keyword is significant in the Command Reference. Keywords that are longer than four characters can have a short form or a long form. SCPI accepts either form. Upper-case letters show the short form of a command keyword.

SCPI is sensitive to white space characters. White space characters are not allowed within command keywords. They are only allowed when they are used to separate a command and a parameter.

A message terminator is required at the end of a program message or a response message. Use <NL>, <^END> or <NL> <^<END> as the *program message terminator.* The word <NL> is an ASCII new line (line feed) character. The word <^END> means that End or Identify (EOI) is asserted on the GPIB interface at the same time the preceding data byte is sent. Most programming languages send these terminators automatically. For example, if using the BASIC OUTPUT statement, <NL> is automatically sent after the last data byte. If using a PC, one can usually configure the system to send whatever terminator is specified.

Syntax Descriptions

Syntax descriptions in the SCPI command reference chapter use the following elements:

<CHAR> This item designates a string of ASCII characters. There are no delimiters. Usually, the string is from an explicit set of responses. Maximum length is twelve characters.

<STRING> This item specifies any 8-bit characters delimited by single quotes or double quotes. The beginning and ending delimiter must be the same. If the delimiter character is within the string, it must be entered twice. (For example, to get “EXAMPLE”, enter ""EXAMPLE"").

VT2216A SCPI Quick Reference

Command	Description	Page
MMEMory:SCSI[1|2|...|30]:ERASe	Erases blocks on certain magneto-optical disks	234
MMEMory:SCSI[1|2|...|30]:OPEN	Opens a SCSI device	235
MMEMory:SESSion[1|2|...|12]:ADD	Adds a Transfer Unit to a SCSI Session	238
MMEMory:SESSion[1|2|...|12]:COPY	Copies data from one SCSI Session to another	239
MMEMory:SESSion[1|2|...|12]:DELete:ALL	Deletes all Transfer Units from the specified SCSI Session	240
MMEMory:SESSion[1|2|...|12]:READ:BUFFer	Reads data from a Session into a memory buffer	241
MMEMory:SESSion[1|2|...|12]:READ:FIFO	Reads data from a SCSI Session into a FIFO	242
MMEMory:SESSion[1|2|...|12]:SEEK	Locates a specific logical block in a Session	243
MMEMory:SESSion[1|2|...|12]:SIZE?	Returns the number of Transfer Units in the Session	244
MMEMory:SESSion[1|2|...|12]:WRITe:BUFFer	Writes data to a SCSI Session from a memory buffer	245
MMEMory:SESSion[1|2|...|12]:WRITe:FIFO	Writes data to a SCSI Session from a FIFO	246
MMEMory:TUNit[1|2|...|15]:CLOSe	Closes an open Transfer Unit	247
MMEMory:TUNit[1|2|...|15]:OPEN	Opens a Transfer Unit	248
Sequence Operations		
SEQuence[1|2|3|4]:ADD	Appends an operation to the specified Sequence	249
SEQuence[1|2|3|4]:BEGin	Begins a Sequence for data transfer	250
SEQuence[1|2|3|4]:DELete:ALL	Removes all operations from the specified Sequence list	251
SEQuence[1|2|3|4]:SIZE?	Returns the number of elements in the Sequence	252
SEQuence[1|2|3|4]:TRANsferred?	Returns the number of bytes transferred in the Sequence	253
Status Reporting		
STATus:OPERation:CONDition?	Reads the Operation Status condition register	254
STATus:OPERation:ENABle	Sets and queries bits in the Operation Status enable register	255
STATus:OPERation[:EVENt]?	Reads and clears the Operation Status event register	256
STATus:OPERation:NTRansition	Sets and queries bits in the Operation Status negative transition register	257
STATus:OPERation:PTRansition	Sets and queries bits in the Operation Status positive transition register	258
STATus:PRESet	Sets bits in most enable and transition registers to the default state	259
STATus:QUEStionable:CONDition?	Reads the Questionable Status condition register	260
STATus:QUEStionable:ENABle	Sets and queries bits in the Questionable Status enable register	261

Command	Description	Page
STATus:QUEStionable[:EVENt]?	Reads and clears the Questionable Status event register	262
STATus:QUEStionable:NTRansition	Sets and queries bits in the Questionable Status negative transition register	263
STATus:QUEStionable:PTRansition	Sets and queries bits in the Questionable Status positive transition register	264

System Control

SYSTem:ABORt	Aborts a data transfer Session and/or Sequence	265
SYSTem:COMMunicate:SCSI[:SELF]:ADDRess	Changes the module's SCSI bus controller address	266
SYSTem:ERRor?	Returns one error message from the module's queue	267
SYSTem:VERSion?	Returns the SCPI version to which the module complies	268

Diagnostics

DIAGnostic:BOARd:MAIN?	Tests the Main PC board	217
DIAGnostic:BOARd:SCSI?	Tests the SCSI PC board	218
DIAGnostic:LBUS:CONSume?	Tests the local bus data transfer to module	219
DIAGnostic:LBUS:GENerate?	Tests the local bus data transfer from module	220
DIAGnostic:SCSI:DAT?	Tests DAT at specific SCSI bus/address	221
DIAGnostic:SCSI:DEVices?	Tests the interface for a specific SCSI controller	222
DIAGnostic:SCSI:DISK?	Tests disk at specific SCSI bus/address	223

The following commands are provided for backward compatibility with models Agilent/HP E1562A/B/C (which use HP-manufactured disk drives) and application software designed to support them. Their behavior for the non-HP-manufactured disk drives used in the VT2216A or for non-HP- manufactured disk drives supplied by the customer are described in the command descriptions on the following pages.

MMEMory:SCSI[1|2|...|30]:CALibrate:AUTO	Sets or queries the Auto Head Calibration Mode of an open SCSI device	227
MMEMory:SCSI[1|2|...|30]:CALibrate[:IMMediate]	Performs head calibration on an open SCSI device	229
MMEMory:SCSI[1|2|...|30]:CALibrate:TIME?	Returns the time until the next head calibration	230
MMEMory:SCSI[1|2|...|30]:TEMPerature?	Returns drive temperature	237

VT2216A SCPI Commands

*CLS command

Clears the Status Byte by emptying the error queue and clearing all event registers.

Command Syntax: `*CLS`

Example Statements:
```
OUTPUT 70918;":*CLS"
OUTPUT 70918;"*cls"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This command clears the Status Byte register. It does so by emptying the error queue and clearing (setting to 0) all bits in the event registers of the following register sets:

- Questionable Status
- Standard Event
- Operation Status

In addition, *CLS cancels any preceding *OPC command or query. This ensures that bit 0 of the Standard Event register will not be set to 1 and that a response will not be placed in the instrument's output queue when pending overlapped commands are completed.

*CLS does not change the current state of enable registers or transition filters.

Note

To guarantee that the Status Byte's Message Available and Master Summary Status bits are cleared, send *CLS immediately following a Program Message Terminator.

For more information on the Status Byte register, see "The VT2216A Registers Sets" on page 192.

*ESE command/query

Sets or queries bits in the Standard Event Status enable register.

Command Syntax:

```
*ESE <Mask>

<Mask>::=number
         limits: 0:255
```

Example Statements:

```
OUTPUT 70918;"*ese 1"
OUTPUT 70918;"*ESE 60"
```

Query Syntax:

```
*ESE?
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This command allows one to set bits in the Standard Event Status enable register. Assign a decimal weight to each bit that is to be set (to 1) according to the following formula:

$2^{(bit\_number)}$

with acceptable values for bit_number being 0 through 7. Add the weights and then send the sum with this command.

When an enable register bit is set to 1, the corresponding bit of the Standard Event Status event register is enabled. All enabled bits are logically ORed to create the Standard Event Status summary, which reports to bit 5 of the Status Byte. Bit 5 is only set to 1 if both of the following are true:

- One or more bits in the Standard Event Status event register are set to 1.
- At least one set bit is enabled by a corresponding bit in the Standard Event Status enable register.

The query returns the current state of the Standard Event Status enable register. The state is returned as a sum of the decimal weights of all set bits.

For more information on the Standard Event Status register set, see the “VT2216A Register Set Summary” on page 197.

*ESR? query

Reads and clears the Standard Event Status event register.

Query Syntax: `*ESR?`

Example Statements:
```
OUTPUT 70918;":*ESR?"
OUTPUT 70918;"*esr?"
```

Return Format: Integer

Attribute Summary: Preset State: +0
Synchronization Required: no
SCPI Compliance: confirmed

Description: This query returns the current state of the Standard Event Status event register. The state is returned as a sum of the decimal weights of all set bits. The decimal weight for each bit is assigned according to the following formula:

$2^{(bit\_number)}$

with acceptable values for bit_number being 0 through 7.

The query clears the register after it reads the register.

A bit in this register is set to 1 when the condition it monitors becomes true. A set bit remains set, regardless of further changes in the condition it monitors, until one of the following occurs:

- The register is read with this query.
- All event registers are cleared with the *CLS command.

For more information on the Standard Event Status enable register set, see the “VT2216A Register Set Summary” on page 197.

*IDN? query

Returns a string that uniquely identifies the module.

Query Syntax: `*IDN?`

Example Statements:
```
OUTPUT 70918;":*IDN?"
OUTPUT 70918;"*idn?"
```

Return Format: Hewlett-Packard,N2216A (E1562E),<serial_number><software_revision>

Note

This response will vary with the revision and date of the firmware used. A VXI Technology response will yield a response with a "VT" prefix (e.g. "VT2216A") or an Agilent/HP response will be returned with an "N" prefix (e.g. "N2216"). Both are acceptible.

Attribute Summary:
Preset State: instrument dependent
Synchronization Required: no
SCPI Compliance: confirmed

Description: This query returns:

- The name of the manufacturer.
- The product number, N2216A (E1562E)
- The serial number
- The version of the software

*OPC command/query

Enable status bit or query completion of all pending overlapped commands.

Command Syntax: `*OPC`

Example Statements:
```
OUTPUT 70918;":*OPC"
OUTPUT 70918;"*opc"
```

Query Syntax: `*OPC?`

Return Format: Integer

Attribute Summary:
Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description:

Some commands are processed sequentially. A sequential command holds off the processing of subsequent commands until it has been completely processed. However, some commands do not hold off the processing of subsequent commands. These commands are called overlapped commands. At times, overlapped commands require synchronization. The Attribute Summary for each command indicates whether it requires synchronization.

The module uses the No Pending Operation (NPO) flag to keep track of overlapped commands that are still pending (that is, not completed). The NPO flag is reset to 0 when an overlapped command is pending. It is set to 1 when no overlapped commands are pending. The NPO flag cannot be read directly, but *OPC and *OPC? can be used to tell when the flag is set to 1.

If *OPC is used, bit 0 of the Standard Event Status event register is set to 1 when the NPO flag is set to 1. This allows the instrument to generate a service request when all pending overlapped commands are completed (assuming that bit 0 of the Standard Event Status register and bit 5 of the Status Byte register have been enabled).

If *OPC? is used, +1 is placed in the output queue when the NPO flag is set to 1. This effectively allows the controller to be paused until all pending overlapped commands are completed. It must wait until the response is placed in the queue before it can continue.

Note

The *CLS and *RST commands cancel any preceding *OPC command or query. Pending overlapped commands are still completed, but one can no longer determine when.

*RST command

Executes a device reset.

Command Syntax: `*RST`

Example Statements:

```
OUTPUT 70918;":*RST"
OUTPUT 70918;"*rst"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This command returns the instrument to a reset state. In addition, *RST cancels any pending *OPC command or query.

The reset state is the same as the preset state. The preset state of each command is listed in the Attribute Summary.

The following are *not* affected by this command:

- The error queue
- The state of all enable registers
- The state of all transition registers

*SRE command/query

Sets or queries bits in the Service Request enable register.

Command Syntax:

```
*SRE <Mask>

<Mask>::=number
         limits: 0:255
```

Example Statements:

```
OUTPUT 70918;":*SRE 128"
OUTPUT 70918;"*sre 32"
```

Query Syntax:

```
*SRE?
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This command allows bits in the Service Request enable register to be set. Assign a decimal weight for each bit to be set (to 1) according to the following formula:

$2^{(bit\_number)}$

with acceptable values for bit_number being 0 through 7. Add the weights and then send the sum with this command.

Note

The module ignores the setting specified for bit 6 of the Service Request enable register. This is because the corresponding bit of the Status Byte register is always enabled.

The module requests service from the active controller when one of the following occurs:

- A bit in the Status Byte register changes from 0 to 1 while the corresponding bit of the Service Request enable register is set to 1.
- A bit in the Service Request enable register changes from 0 to 1 while the corresponding bit of the Status Byte register is set to 1.

The query returns the current state of the Service Request enable register. The state is returned as a sum of the decimal weights of all set bits.

*STB? query

Reads the Status Byte register.

Query Syntax: `*STB?`

Example Statements:
```
OUTPUT 70918;":*STB?"
OUTPUT 70918;"*stb?"
```

Return Format: Integer

Attribute Summary: Preset State: variable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This command allows bits in the Status Byte register to be set. The state is returned as a sum of the decimal weights of all set bits. The decimal weight for each bit is assigned according to the following formula:

$2^{(bit\_number)}$

with acceptable values for bit_number being 0 through 7.

The register is not cleared by this query. To clear the Status Byte register, the *CLS command must be sent.

For more information on the Status Byte register, see the “VT2216A Register Set Summary” on page 197.

*TST? query

Performs a selftest on the instrument hardware and returns the results.

Query Syntax: `*TST?`

Example Statements:

```
OUTPUT 70918;":*TST?"
OUTPUT 70918;"*tst?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This command performs tests on both the internal main board and the internal SCSI board by sending the commands DIAGnostic:BOARd:MAIN? and DIAGnostic:BOARd:SCSI?

The following errors indicate that DIAG:BOAR:MAIN failed:

1: diagErr_versionOrSwitch
2: diagErr_fitsReset
3: diagErr_lbusStatic
4: diagErr_fitsStatic
5: diagErr_vxiStatic
6: diagErr_sharedRam
7: diagErr_fifo
8: diagErr_sramMagicRead
9: diagErr_sramMagicWrite
10: diagErr_a24MagicRead
11: diagErr_a24MagicWrite

The following errors indicate that DIAG:BOAR:SCSI failed:

12: diagErr_staticScsi
13: diagErr_inScsi
14: diagErr_outScsi

See the DIAGnostic commands for additional diagnostic tests.

*WAI command

Holds off processing of subsequent commands until all preceding commands have been processed.

Command Syntax: `*WAI`

Example Statements:
```
OUTPUT 70918;":*WAI"
OUTPUT 70918;"*wai"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: Use *WAI to hold off the processing of subsequent commands until all pending overlapped commands have been completed.

Some commands are processed sequentially by the instrument. A sequential command holds off the processing of any subsequent commands until it has been completely processed. However, some commands do not hold off the processing of subsequent commands; they are referred to as overlapped commands. *WAI ensures that overlapped commands are completely processed before subsequent commands (those sent after *WAI) are processed.

DIAGnostic:BOARd:MAIN? query

Tests the Main internal PC board.

Query Syntax: `DIAGnostic:BOARd:MAIN?`

Example Statements:

```
OUTPUT 70918;":DIAGNOSTIC:BOARD:MAIN?"
OUTPUT 70918;"diag:boar:main?"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Failures return a string describing the error.

See “Troubleshooting the VT2216A” starting on page 29 for usage information.

DIAGnostic:BOARd:SCSI? query

Tests the internal SCSI PC board.

Query Syntax: `DIAGnostic:BOARd:SCSI?`

Example Statements:
```
OUTPUT 70918;":DIAGNOSTIC:BOARD:SCSI?"
OUTPUT 70918;"diag:boar:scsi?"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Failures return a string describing the error.

See “Troubleshooting the VT2216A” starting on page 29 for usage information.

DIAGnostic:LBUS:CONSume? query

Tests the a local bus data transfer to the module.

Query Syntax:

```
DIAGnostic:LBUS:CONSume? <Logical Address>
<Logical Address> ::= number
                      limits 0-255
```

Example Statements:

```
OUTPUT 70918;":DIAGNOSTICLBUS:CONSUME? 32"
OUTPUT 70918;"diag:lbus:cons? 96"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:. This test requires more than one VT2216A module. This command is sent to the VT2216A on the right of two adjacent VT2216As and tests the ability to transfer data from the local bus to the module.

<Logical Address> specifies the VXI logical address of the VT2216A to the left of this module.

Failures return a string describing the error.

DIAGnostic:LBUS:GENerate? query

Tests the local bus data transfer from the module.

Query Syntax: `DIAGnostic:LBUS:GENerate? <Logical Address>`

Example Statements:

```
OUTPUT 70918;":DIAGNOSTIC:BUS:GENERATE? 64"
OUTPUT 70918;"diag:bus:gen? 136"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: This test requires more than one VT2216A module to perform. This command is sent to the VT2216A on the left of two adjacent VT2216As and tests the ability to transfer data from the module to the local bus.

<Logical Address> specifies the VXI logical address of the VT2216A to the left of this module.

Failures return a string describing the error.

DIAGnostic:SCSI:DAT? query

Performs tests on a SCSI DAT.

Query Syntax:

```
DIAGnostic:SCSI:DAT? <Controller>,<Bus Address>
<Controller>::=A|B
<Bus Address>::=number
                limits: 0:15
```

Example Statements:

```
OUTPUT 70918;":DIAGNOSTIC:SCSI:DAT? A,0"
OUTPUT 70918;"diag:scsi:dat? a,7"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Tests the DAT specified at the given SCSI bus address. If the device at this address is not a DAT an error will be returned.

Note

A DAT tape must be inserted in the DAT drive in order to perform this test. All data on the tape will be destroyed during this test.

Failures return a string describing the error.

DIAGnostic:SCSI:DEVices? query

Verifies the interface for a specific SCSI controller.

Query Syntax:

```
DIAGnostic:SCSI:DEVices? <Controller>
<Controller>::=A|B
```

Example Statements:

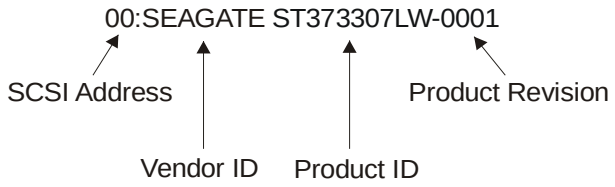
```
OUTPUT 70918;":DIAGNOSTIC:SCSI:DEVICES? A"
OUTPUT 70918;"diag:scsi:dev? b"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: This command verifies correct operation of a single SCSI controller chip and also its interface to external devices. The command must be sent to each controller to verify that both of the SCSI controller chips are functioning. If the test is successful the command returns a string which contains the SCSI address, vendor id, product id and product revision in a "dd:aaaaaaa aadddddddaa-dddd" format. A null string is returned if the test fails.

Example of the string returned when the SCSI bus contains a Seagate disk at SCSI address 0:



This command does not verify that the devices connected to the SCSI bus are operating correctly. The command DIAG:BOARd:SCSI? verifies that the VT2216A side of the controller chip is functioning correctly. The command DIAG:SCSI:DISK? and DIAG:SCSI:DAT? verify that individual devices are functioning correctly.

See “Troubleshooting the VT2216A” starting on page 29 for usage information.

DIAGnostic:SCSI:DISK? query

Performs tests on a SCSI disk drive.

Query Syntax:

```
DIAGnostic:SCSI:DISK? <Controller>,<Bus Address>
<Controller>::=A|B
<Bus Address>::=number
                limits 0:15
```

Example Statements:

```
OUTPUT 70918;":DIAGNOSTIC:SCSI:DISK? A,12"
OUTPUT 70918;"diag:scsi:disk? b,2"
```

Return Format: String

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Tests the disk drive specified at the given SCSI bus address. If the device at this address is not a disk, an error will be returned.

Failures return a string describing the error.

See “Troubleshooting the VT2216A” starting on page 29 for usage information.

LBUS:READ:BUFFer command

Reads data from the module to the left of the VT2216A and writes it to a memory buffer.

Command Syntax:

```
LBUS:READ :BUFFer<Count>,<Blocksize>,<Offset>,<Memory Space>

<Count>::=number
          limits: 1:256

<Blocksize>::=number
              limits: 16:65536

<Offset>::=number
           limits: 0:4294967295

<Memory Space>::=A24|A32|SRAM
```

Example Statements:

```
OUTPUT 70918;":LBUS:READ:BUFFER 16,#H3000D000,A32"
OUTPUT 70918;"lbus:read:buff 64,#h20a080,a24"
```

Attribute Summary:
Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

This command may be used in conjunction with MMEM:SESS:WRIT:BUFF to transfer data from the local bus to a session. In most cases, users find it easier and faster to use Sequence throughput operations.

This command reads data from the module to the left of the VT2216A on local bus and writes it using D16 to a designated memory location using LBUS CONSUME mode and four bytes per local bus element. The VINStrument:LBUS command must have been previously sent with the NORMal parameter.

<Count> specifies the number of LBUS blocks to write to the memory buffer from the local bus.

<Blocksize> specifies the number of bytes per local bus block. A block marker will be received after this many bytes have been read. The maximum number of local bus blocks read is 256.

<Offset> indicates where in the designated memory buffer space the data will be written. The value is an offset from the beginning of the address space. All address spaces start at offset 0. A24 has an upper limit of 16777215, A32 has an upper limit of 4294967295, SRAM has an upper limit of 262143. The value of offset must be divisible by two.

<Memory space> specifies into which memory to write the data blocks. The usable memory spaces for this command are A24, A32 and SRAM. SRAM indicates the VT2216A shared RAM.

Note

Be sure enough memory space is available for the data to be transferred.

LBUS:WRITe:BUFFer command

Reads data from a memory buffer and writes it to the module to the right of the VT2216A.

Command Syntax:

```
LBUS:WRITe:BUFFer <Count>,<Blocksize>,<Offset>,<Memory Space>

<Count>::=number
          limits: 1:256

<Blocksize>::=number
              limits: 16:65536

<Offset>::=number
           limits: 0:4294967295

<Memory Space>::=A24|A32|SRAM
```

Example Statements:

```
OUTPUT 70918;"LBUS:WRITE:BUFFER 128,16,#H20000000,A32"
OUTPUT 70918;"lbus:writ:buff 16,#h10,#h128,sram"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

This command may be used in conjunction with MMEM:SESS:READ:BUFF to transfer data from a session to the local bus. In most cases, users find it easier and faster to use Sequence playback operations.

This command uses D16 to read blocks of data from a designated offset in a memory buffer then writes them to the module to the right of the VT2216A on the local bus using LBUS GENERATE mode and four bytes per local bus element. The VINStrument:LBUS command must have been previously sent with the NORMal parameter.

<Count> specifies the number of local bus blocks to read from the memory buffer and copy to the local bus.

<Blocksize> specifies the number of bytes per local bus block. A block marker will be asserted after this many bytes have been transferred on the local bus. A frame marker will also be asserted periodically, but there should be no particular meaning associated with the frame marker. The maximum number of local bus blocks written is 256.

<Offset> indicates from where in the designated memory buffer space the data will be read. The value is an offset from the beginning of the address space. All address spaces start at offset 0. A24 has an upper limit of 16777215, A32 has an upper limit of 4294967295, SRAM has an upper limit of 262143. The value of offset must be divisible by two.

MMEMory:SCSI[1|2|...|30]:BSIZe? query

Returns the number of bytes in a logical block for an open SCSI device.

Query Syntax: `MMEMory:SCSI[1|2|...|30]:BSIZe?`

Example Statements:
```
OUTPUT 70918;":MMEMory:SCSI4:BSIZe?"
OUTPUT 70918;"mmemory:scsi23:bsize?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: The returned value is the number of bytes in a logical block for any SCSI device that has been opened by the MMEM:SCSI:OPEN command. An error will be generated if the device is not currently open. To find the number of logical blocks on a device send MMEM:SCSI:CAPacity?

Some of the MMEM commands require addresses and counts specified in terms of logical blocks. The return value of this query specifies the size of a logical block.

MMEMory:SCSI[1|2|...|30]:CALibrate:AUTO command/query

Note

This command is provided for backward compatability with models Agilent/HP E1562A/B/C (which use HP-manufactured disk drives) and application software designed to support them. Using this command with the non-HP-manufactured disk drives (including those in the VT2216A) results in no action. The query returns the current head calibration mode, but the number does not mean anything for a non-HP drive.

Sets or queries the Auto Head Calibration Mode of an open HP disk drive.

Command Syntax:

```
MMEMory:SCSI[1|2|...|30]:CALibrate:AUTO <Auto Cal Mode>
<Auto Cal Mode>::=ON|OFF
```

Example Statements:

```
OUTPUT 70918;":MMEM:SCSI14:CAL:AUTO OFF"
OUTPUT 70918;"mmemory:scsi18:calibrate:auto ON"
```

Query Syntax:

```
MMEMory:SCSI[1|2|...|30]:CALibrate:AUTO?
```

Return Format: Integer

Attribute Summary:
Preset State: ON
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Auto head calibration mode is supported only for certain Hewlett-Packard SCSI Direct-Access devices (hard disks). Many disks periodically perform a head recalibration to assure that the positioning capability is within specification through temperature changes and over time. This command is useful for disabling head calibration during medium to high speed data transfers.

An example of the head calibration schedule of a C2490 disk drive follows:

Time since spin up (minutes)	Delta (minutes)
0	
2	2
4	2
6	2
8	2
10	2
12	2
15	3
18	3
22	4
27	5
33	6
42	9
60	18
100	40
160	60
220	60
etc	etc

Caution

If auto calibration is disabled for a period of time longer than the device finds acceptable (in the instance of the C2490 this is twice the time in the table), writes to the device may be disabled or the device may force a head calibration to be done without regard to the state of the auto calibration flag. This may result in an overflow condition due to an interruption of real-time data flow.

When any SCSI device is opened, the auto-calibration mode will be enabled.

<Auto Cal Mode> indicates whether the automatic head recalibration mode should be enabled or disabled.

This command generates an error if the device is not currently open.

This command is not available for SCSI devices that are not Hewlett-Packard disks and will generate an error.

The query returns the current state of the automatic head calibration mode for the device: 0=OFF, 1=ON.

MMEMory:SCSI[1|2|...|30]:CALibrate[:IMMediate] command

Note

This command is provided for backward compatability with models Agilent/HP E1562A/B/C (which use HP-manufactured disk drives) and application software designed to support them. Using this command with the non-HP-manufactured disk drives (including those in the VT2216A) results in no action.

Performs head calibration on an HP disk drive.

Command Syntax:

```
MMEMory:SCSI[1|2|...|30]:CALibrate[:IMMediate]
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SCSI8:CALIBRATE:IMMEDIATE"
OUTPUT 70918;"mmem:scsi:cal"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

This command is intended only for certain Hewlett-Packard SCSI Direct-Access devices (hard disks). Many disks periodically perform a head recalibration to assure that the positioning capability is in specification through temperature changes and over time. This command will force certain Hewlett-Packard supported disks to perform a head calibration immediately. This command is useful before starting a high speed data transfer. It is not useful when the data transfer rate will be significantly less than the disk's sustained media transfer rate.

This command generates an error if the device is not currently open.

MMEMory:SCSI[1|2|...|30]:CALibrate:TIME? query

Note

This command is provided for backward compatability with models Agilent/HP E1562A/B/C (which use HP-manufactured disk drives) and application software designed to support them. Using this command with the non-HP-manufactured disk drives (including those in the VT2216A) returns a value of 9,999,999.

Returns the time until the next head calibration.

Query Syntax:

```
MMEMory:SCSI[1|2|...|30]:CALibrate:TIME?
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SCSI7:CALibrate:TIME?"
OUTPUT 70918;"mmem:scsi7:cal:time?"
```

Return Format: Integer

Attribute Summary:
Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: This command will work on HP disk drives only. It is a good idea to check the time remaining until the next calibration before starting a long throughput. If the time is short, send the MMEM:SCSI:CAL command to each of the devices in the Session and send this query again. If the time is still short, wait for the disks to be spun up for a longer length of time. See the MMEM:SCSI:CAL:AUTO command for a schedule of calibration times.

This query generates an error if the device is not currently open.

MMEMory:SCSI[1|2|...|30]:CAPacity? query

Returns the number of logical blocks on an open SCSI device.

Query Syntax: `MMEMory:SCSI[1|2|...|30]:CAPacity?`

Example Statements:
```
OUTPUT 70918;":MMEMORY:SCSI7:CAPACITY?"
OUTPUT 70918;"mmem:scsi7:cap?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: The returned value is the number of available logical blocks for any SCSI device that has been opened by the MMEM:SCSI:OPEN command. All Direct-Access SCSI devices (disks) support this capability and will return a meaningful result. Other types of devices may not be able to provide any meaningful information and will return 4294967295.

The size, in bytes, of each logical block can be determined by sending the MMEM:SCSI:BSIZe? query. By using the combination of both commands and multiplying their results, the total byte capacity of the device can be determined.

This command generates an error if the device is not currently open.

MMEMory:SCSI[1|2|...|30]:CLOSe command

Closes an open SCSI device.

Command Syntax: `MMEMory:SCSI[1|2|...|30]:CLOSe`

Example Statements:

```
OUTPUT 70918;":MMEM:SCSI2:CLOS"
OUTPUT 70918;"mmemory:scsi20:close"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Use of this command will make the specified logical device descriptor unavailable for use with other commands. If a SCSI device has been opened more than once using different logical device descriptors, it will need to be closed more than once. An error will be returned if the device is not currently open.

If the MMEM:SCSI:CLOS command is sent while the logical descriptor is in use in a TUNIT or SESSION; indeterminate and undesirable results will occur.

MMEMory:SCSI[1|2|...|30]:EBYPass [:STATe] command/query

Sets or queries the Erase Bypass Mode of an open SCSI device.

Command Syntax:
```
MMEMory:SCSI[1|2|...|30]:EBYPass[:STATe] <Erase Mode>
<Erase Mode>::=ON|OFF
```

Example Statements:
```
OUTPUT 70918;":MMEMORY:SCSI16:EBYPASS ON"
OUTPUT 70918;"mmem:scsi22:ebyp:stat OFF"
```

Query Syntax:
```
MMEMory:SCSI[1|2|...|30]:EBYPass[:STATe]?
```

Return Format: Integer

Attribute Summary:
Preset State: OFF
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Erase bypass mode is supported only by certain magneto-optical disks that require an erase before write. This mode must only be enabled when the media has never been written to or if an erase has already been performed in the area to be written.

Caution

When erase bypass mode is active, a write command to a non-erased block on a device will fail and the data will be lost. In addition, the error correction bits on the device will be corrupted making the block unreadable.

This command generates an error if the device is not currently open or if the device does not support erasing.

<Erase Mode> indicates whether the erase bypass mode should be enabled or disabled. The value upon opening a device is OFF or disabled.

The query returns the current state of the erase bypass mode for the device: 0=OFF, 1=ON.

MMEMory:SCSI[1|2|...|30]:ERASe command

Erase blocks on an open SCSI device.

Command Syntax:

```
MMEMory:SCSI[1|2|...|30]:ERASe <Address>,<Length>

<Address>::=number
            limits: 0:4294967295

<Length>::=number
           limits: 0:4294967295
```

Example Statements:

```
OUTPUT 70918;":MMEM:SCSI15:ERAS #H1CC00,#H200"
OUTPUT 70918;"mmemory:scsi3:erase 117760,512"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: This command should be sent only for certain magneto-optical devices. It is used to speed up writes to these types of devices that require an erase before write. This command should be used in conjunction with the command MMEM:SCSI:EBYP.

Caution

Any block that has been erased cannot be read. The erasure removes data as well as the error correction bits on the media. A read of media without correction bits will fail.

<Address> is the logical block number at which to start erasing.

<Length> is the number of logical blocks to erase.

This command generates an error if the device is not currently open or if the device does not support erasing.

This command does not have a query form.

MMEMory:SCSI[1|2|...|30]:OPEN command/query

Opens a SCSI device.

Command Syntax:

```
MMEMory:SCSI[1|2|...|30]:OPEN <Controller>,<Bus address>,<Device unit>,
<Mode>

<Controller>::=A|B

<Bus address>::=number
                limits: 0:15

<Device unit>::=number
                limits: 0:7

<Mode>::=number
         limits: 0:4294967295
```

Example Statements:

```
OUTPUT 70918;":MMEM:SCSI5:OPEN A,5,0,#H40"
OUTPUT 70918;"mmemory:scsi2:open b,0,0,#he8"
```

Query Syntax:

```
MMEMory:SCSI[1|2|...|30]:OPEN?
```

Return Format: Integer

Attribute Summary:
Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: This command opens the logical device descriptor specified by the subopcode [1|2|...|30] on the SCSI node.

The following conditions generate an error:

- This logical descriptor is already open.
- No device responds at the address specified by <Controller>, <Bus address>, <Device unit>.
- The device at the designated address is not a block-oriented, mass storage device.

With no other errors, the device is opened and ready for use in other commands.

The query version of this command requires no parameters. It returns 0 if the logical device descriptor is not currently open and 1 if the logical descriptor is already open.

<Controller> is A or B indicating to which SCSI controller the device is connected. The A or B corresponds to the A or B on the front-panel of the module.

<Bus address> is 0-15 indicating the SCSI logical address.

<Device unit> is 0-7 indicating the logical unit number. The logical unit number for all VT2216A devices is 0. Some external devices may have other logical unit numbers.

<Mode> is an unsigned integer representing a bitfield that indicates various options for the device. Bits in the <Mode> field are:

Mode Bitfield	Description	Bits	
		Decimal	Hexa-decimal
TM_verifyAfterWrite	Performs a medium verification after every write. *Not* supported for sequential devices.	1	0x001
TM_reserve	Do not allow other SCSI initiators to access the device while it is open.	2	0x002
TM_preventRemoval	For devices with removable media, do not allow the media to be removed while the device is open.	4	0x004
TM_ejectOnClose	For devices with removable media, eject the media when the device is closed. For fixed-media devices, spin down the drive when it is closed.	8	0x008
TM_writeWithoutErase	For optical memory devices, disable the erase-before-write to increase speed of writes. See SCPI commands MMEM:SCSI:ERAS and MMEM:SCSI:EBYP for warnings about this mode.	16	0x010
TM_asynchronousTransfer	Use asynchronous instead of synchronous transfers to the device. This cuts the max transfer rate in half.	32	0x020
TM_preventDisconnect	Do not allow the device to disconnect while doing certain operations. This is slightly faster than allowing disconnect.	64	0x040
TM_dontStartUnit	Do not spin up a disk. Reads and writes will cause errors under these conditions but the disks may be configured to spin up on powerup. This bit is good for a quick check for which devices are present without waiting for a spin up.	256	0x100
TM_readOnly	Return an error if any write to the device is attempted.	512	0x200

In order to specify more than one of the above modes, add the desired bit values together to obtain the mode value to send to the VT2216A.

SCSI devices may be opened more than once via different logical device descriptors (represented by the subopcode in the command), but the mode field specified the first time a device is opened is used for all successive times it is opened even if the specified value is different. Any device that is opened multiple times, must also be closed multiple times.

MMEMory:SCSI[1|2|...|30]:TEMPerature? query

Note

This command is provided for backward compatability with models Agilent/HP E1562A/B/C (which use HP-manufactured disk drives) and application software designed to support them. Using this command with the non-HP-manufactured disk drives (including those in the VT2216A) returns a value 1.0°C (first integer = 1, second integer = 0).

Returns the temperature of an HP disk drive

Query Syntax:

```
MMEMory:SCSI[1|2|...|30]:TEMPerature?
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SCSI4:TEMPERATURE?"
OUTPUT 70918;"mmemory:scsi23:temp?"
```

Return Format:

Integer
Integer

Attribute Summary:

Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

This command is supported for certain Hewlett-Packard disk drives only. Two integer values are returned. The first integer represents the temperature in degrees Centigrade. The second integer represents the fractional part of an additional degree and is designated as the number of 256ths of a degree.

Absolute accuracy of temperate is ±5°C.

MMEMory:SESSion[1|2|...|12]:ADD command

Adds a Transfer Unit to a Session.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:ADD <Transfer Unit>,<Count>

<Transfer Unit>::=number
                  limits: 1:15

<Count>::=number
          limits: 1:33554430
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SESSION12:ADD 5,26974"
OUTPUT 70918;"mmemory:session:add 12,#h7fff"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

All data transfers, including Sequences, are performed using Sessions. A Session may contain no more than fifteeh Transfer Units. This command adds a Transfer Unit to the specified Session. The Session can be cleared using the MMEM:SESS:DEL:ALL command.

Every Transfer Unit that is added to a Session must contain the same number of SCSI devices. If each Transfer Unit in a Session is made up of a single device, each device must be on the same SCSI controller. This command generates an error if these constraints are not met.

<Transfer Unit> determines which logical Transfer Unit is to be added to the SESSION. Transfer units are always added to the end of the current list of Transfer Units. The command generates an error if the Transfer Unit is not currently open. A Transfer Unit may be included in more than one Session. It is also permissible to include the same Transfer Unit more than once in a single Session, although there is no good reason to do so.

<Count> is the number of logical blocks to transfer to or from this Transfer Unit before switching to the next Transfer Unit in the Session. The size of a logical block can be determined by sending the MMEM:SCSI:BSIZ? query. When there are multiple Transfer Units in the Session, <Count> should be chosen such that the total number of bytes written at one time is less than or equal to the number of bytes of cache contained on the device(s). If the Transfer Unit contains two devices, <Count> specifies the number of logical blocks to be written to or read from the pair of devices, half of which will be written to or read from an individual device. The largest value of <Count> for disk drives is 65535 per disk drive or 131070 for a pair of disk drives.

MMEMory:SESSion[1|2|...|12]:COPY command

Copies data from one Session to another.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:COPY <Destination Session>,<Count>

<Destination Session>::=number
                        limits: 1:12

<Count>::=number
          limits: 1:4294967295
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SESSION2:COPY 10,#H40000"
OUTPUT 70918;"mmemory:session:copy 1,262144"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

This command copies the contents of a Session (which may be split width-wise across two devices and/or length-wise across N Transfer Units) to another Session. The Session may consist of a single Transfer Unit or even a single SCSI device. It is even possible to copy data from one part of a device to another part of the same device as long as the copied data does not overlap the original data. Both Sessions must have been properly initialized using open Transfer Units and open SCSI devices. This command provides a convenient means to backup a previously acquired throughput Session, or to recombine a throughput Session that was split lengthwise and/or widthwise into a linear file.

Data is read from one Session into the VT2216A FIFO; then data is written to the destination Session.

<Destination Session> is the destination Session of the copy. An error will be generated if the Session does not contain any Transfer Units.

<Count> is the number of logical disk blocks to copy. It refers to the number of logical blocks on the source Session since it is possible to have different sized logical blocks between the source Session and the destination Session. The size of a logical block for a given device can be obtained by sending the MMEM:SCSI:BSIZ query.

The expected backup performance is greater than the total time required for source plus destination device access.

MMEMory:SESSion[1|2|...|12]:DELete:ALL command

Deletes all Transfer Units from the specified Session.

Command Syntax: `MMEMory:SESSion[1|2|...|12]:DELete:ALL`

Example Statements:
```
OUTPUT 70918;":MMEMORY:SESSION:DELETE:ALL"
OUTPUT 70918;"mmem:sess3:del:all"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: All Transfer Units are removed from the Session definition. The Transfer Units are NOT closed. To close the Transfer Units send the command MMEM:TUN:CLOS.

MMEMory:SESSion[1|2|...|12]:READ:BUFFer command

Reads data from a Session into a memory buffer.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:READ:BUFFer <Count>,<Offset>,<Memory Space>

<Count>::=number
          limits: 1:4294967295

<Offset>::=number
           limits: 0:4294967295

<Memory Space>::=A24|A32|SRAM
```

Example Statements:

```
OUTPUT 70918;":MMEM:SESS6:READ:BUFF 256,131072,SRAM"
OUTPUT 70918;"mmem:sess1:read:buff #H200,3354430,A32"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

The memory buffer may be in the A24, A32, or VT2216A shared RAM memory spaces. The A16 memory space is not usable, due to the limited addressable area available in that memory space. MMEM:SESS:SEEK may be used to position the Session to where the read will take place. All data transfers to the buffer are done using D16.

<Count> specifies the number of logical blocks to read from the Session and copy to the specified memory space. If the Session contains Transfer Units that consist of two SCSI devices, count must be an even number or the data transfer will be indeterminate.

<Offset> specifies where in the memory space, the data will be written. The value is an offset in bytes from the beginning of the address space. It is up to the user to make sure that there is enough space available starting at this offset in which to write the data. All address spaces start at offset 0. A24 has an upper limit of 16777215, A32 has an upper limit of 4294967295, SRAM has an upper limit of 262143. The value of offset must be divisible by two.

<Memory Space> specifies the memory space into which to copy the data. The memory spaces that make sense for this command are A24, A32 and SRAM. SRAM indicates the VT2216A shared RAM.

MMEMory:SESSion[1|2|...|12]:READ:FIFO command

Reads data from a SCSI Session into a FIFO.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:READ:FIFO <Count>,<Offset>,<Memory Space>,
<FIFO width>

<Count>::=number
          limits: 1:4294967295

<Offset>::=number
           limits: 0:4294967295

<Memory Space>::=A16|A24|A32|SRAM

<FIFO width>::=number
               limits: 16|32
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SESSION8:READ:FIFO 32768,#HDA2A,A16,16"
OUTPUT 70918;"mmemory:session:read:fifo 128,#h280040,A24,32"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

The FIFO may be in the A16, A24, A32, or VT2216A shared RAM memory spaces. MMEM:SESS:SEEK may be used to position the Session to where the read will take place. This command differs from MMEM:SESS:READ:BUFF in that the offset is not incremented – every write to the FIFO is at the same offset in the memory space.

<Count> specifies the number of SCSI logical blocks to read from the Session and copy to the specified memory space. If the Session contains Transfer Units that consist of two SCSI devices, count must be an even number or the data transfer will be indeterminate.

<Offset> specifies where in the memory space, the data will be written. The value is an offset in bytes from the beginning of the address space. All address spaces start at offset 0. A16 has an upper limit of 65535, A24 has an upper limit of 16777215, A32 has an upper limit of 4294967295, SRAM has an upper limit of 262143. The value of offset must be divisible by two.

<Memory Space> specifies into which memory space to copy the data. The memory spaces are A16, A24, A32 and SRAM. SRAM indicates the VT2216A shared RAM.

<FIFO width> indicates the number of bits in each element of the FIFO. The only acceptable values are 16 and 32. All writes to the FIFO are done using D16, thus a 32 bit FIFO consists of two writes.

MMEMory:SESSion[1|2|...|12]:SEEK command

Sets up all devices in a Session to allow the next data transfer to begin at the specified block number.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:SEEK <Block>

<Block>::=number
          limits: 0:4294967295
```

Example Statements:

```
OUTPUT 70918;":MMEM:SESS11:SEEK 8191"
OUTPUT 70918;"mmemory:session4:seek #h1FFF"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

This command sets the current position of the Session to be offset to the specified logical block from the beginning of the Session. If Transfer Units in the Session contain two SCSI devices, the specified logical block must be even. All Transfer Units in the Session will be positioned such that a MMEM:SESS:READ:*, MMEM:SESS:WRIT:*, or SEQ:BEG will start at the specified logical block.

The block number specified here does not correspond to a specific logical block number on a particular device. Rather the block number corresponds to an offset from the logical block number specified as the beginning logical block when each Transfer Unit was opened with MMEM:TUN:OPEN. In other words, when a Transfer Unit is opened, it can specify to begin at some non-zero logical block number. Therefore, specifying a seek to block zero means that the next data transfer will begin at the first block of the Session. This corresponds to the non-zero block number specified when each Transfer Unit was opened. This is also true if a Transfer Unit consists of a single device, such that a single-device Session may also start at a non-zero block. Furthermore, for Transfer Units that consist of a pair of SCSI devices, the data is two logical blocks in size and is spread across the pair of devices alternating two bytes to each device.

MMEMory:SESSion[1|2|...|12]:SIZE? query

Returns the number of Transfer Units in the Session.

Query Syntax: `MMEMory:SESSion[1|2|...|12]:SIZE?`

Example Statements:
```
OUTPUT 70918;":MMEMORY:SESSION1:SIZE?"
OUTPUT 70918;"mmemory:session7:size?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

MMEMory:SESSion[1|2|...|12]:WRITe:BUFFer command

Writes data to a SCSI Session from a memory buffer.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:WRITe:BUFFer <Count>,<Offset>,<Memory Space>

<Count>::=number
          limits: 1:4294967295

<Offset>::=number
           limits: 0:4294967295

<Memory Space>::=A24|A32|SRAM
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SESSION11:WRITE:BUFFER 16384,#H300AD0,A24"
OUTPUT 70918;"mmemory:session2:write:buffer 64,4608,SRAM"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

The memory buffer may be in the A24, A32, or VT2216A shared RAM memory spaces. The A16 memory space should not be specified, due to the limited addressable area available in that memory space. MMEM:SESS:SEEK may be used to position the Session to where the SCSI write will be done. All reads from the buffer are done using D16.

<Count> specifies the number of logical blocks to write to the Session from the specified memory space. If the Session contains Transfer Units that consist of two SCSI devices, count must be an even number or the data transfer will be indeterminate.

<Offset> specifies from where in the memory space the data will be read. The value is an offset in bytes from the beginning of the address space. One must make sure that there is enough space available starting at this offset from which to read data. All address spaces start at offset 0. A24 has an upper limit of 16777215, A32 has an upper limit of 4294967295, SRAM has an upper limit of 262143. The value of offset must be divisible by 2.

<Memory Space> specifies from which memory space to copy the data. The memory spaces that make sense for this command are A24, A32 and SRAM. SRAM indicates the VT2216A shared RAM.

MMEMory:SESSion[1|2|...|12]:WRITe:FIFO command

Writes data to a SCSI Session from a FIFO.

Command Syntax:

```
MMEMory:SESSion[1|2|...|12]:WRITe:FIFO <Count>,<Offset>,<Memory Space>,
<FIFO width>

<Count>::=number
          limits: 1:4294967295

<Offset>::=number
           limits: 0:4294967295

<Memory Space>::=A16|A24|A32|SRAM

<FIFO width>::=number
               limits: 16|32
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:SESSION:WRITE:FIFO 3840,#H20000A800,A32,16"
OUTPUT 70918;"mmemory:session2:write:fifo #F00,#hCC38,A16,32"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:

The FIFO may be in the A16, A24, A32, or VT2216A shared RAM memory spaces. MMEM:SESS:SEEK may be used to position the Session to where the write will be done. This command differs from MMEM:SESS:WRIT:BUFF in that the offset is not incremented – every FIFO read is at the same offset in the memory space.

<Count> specifies the number of SCSI logical blocks to write to the Session from the specified memory space. If the Session contains Transfer Units that consist of two SCSI devices, count must be an even number or the data transfer will be indeterminate.

<Offset> specifies from where in the memory space the data will be read. The value is an offset in bytes from the beginning of the address space. All address spaces start at offset 0. A16 has an upper limit of 65535, A24 has an upper limit of 16777215, A32 has an upper limit of 4294967295, SRAM has an upper limit of 262143. The value of offset must be divisible by two.

<Memory Space> specifies from which memory space to copy the data. The memory spaces that make sense for this command are A16, A24, A32 and SRAM. SRAM indicates the VT2216A shared RAM.

<FIFO width> indicates the number of bits in each element of the FIFO. The only acceptable values are 16 and 32. All reads from the FIFO are done using D16, thus a 32 bit FIFO consists of two reads.

MMEMory:TUNit[1|2|...|15]:CLOSe command

Closes an open Transfer Unit.

Command Syntax: `MMEMory:TUNit[1|2|...|15]:CLOSe`

Example Statements:

```
OUTPUT 70918;":MMEMORY:TUNIT12:CLOSE"
OUTPUT 70918;"mmemory:tunit:close"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: An error will be generated if the Transfer Unit is not currently open.

This command only disassociates individual SCSI devices from this Transfer Unit. The underlying SCSI devices will not be closed with this command. The SCSI devices must be closed later with the MMEM:SCSI:CLOS command.

MMEMory:TUNit[1|2|...|15]:OPEN command/query

Opens a Transfer Unit given the underlying SCSI devices and starting logical block numbers.

Command Syntax:

```
MMEMory:TUNit[1|2|...|15]:OPEN <Device1>,<Block1>[,<Device2>,<Block2>]

<Device1>::=number
            limits: 1:30

<Block1>::=number
           limits: 0:4294967295

<Device2>::=number
            limits: 0:30

<Block2>::=number
           limits: 0:4294967295
```

Example Statements:

```
OUTPUT 70918;":MMEMORY:TUNIT2:OPEN 20,0,0,0"
OUTPUT 70918;"mmem:tun13:open 3,2048,1,2048"
```

Query Syntax:

```
MMEMory:TUNit[1|2|...|15]:OPEN?
```

Return Format: Integer

Attribute Summary:
Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

<Device1> is the logical device number used in a previous call to MMEM:SCSIx:OPEN, where x is the value to use here. An error will be generated if the logical device MMEM:SCSIx is not currently open.

<Block1> is the SCSI logical block number specifying the beginning position on the device specified by <Device1>. It will be impossible for MMEM:TUN:* commands or MMEM:SESS:* commands using this MMEM:TUN to reference SCSI logical blocks smaller than the number specified here.

<Device2> is the logical device number used in a previous call to MMEM:SCSIx:OPEN. If the Transfer Unit is to be opened using only a single device this field must be set to 0. An error will be generated if the logical device MMEM:SCSIx is not currently open or if <Device1> and <Device2> are on the same SCSI bus. For a split pair <Device1> must be on the SCSI A bus and <Device2> must be on the SCSI B bus.

<Block2> is the SCSI logical block number specifying the beginning position on the device specified by <Device2>. For a single-device Transfer Unit, this field should be set to 0.

The query version of this command requires no parameters. It returns 0 if the logical Transfer Unit is not currently open and 1 if the logical Transfer Unit is open.

It is possible to use a single, open SCSI device in more than one Transfer Unit, but it is important that all Transfer Units in a Session include an individual SCSI device only once. By including two SCSI units in a Transfer Unit, 32-bit data may be transferred by splitting the data across two devices in such a way as to make the upper 16 bits go to one device and the lower 16 bits of the quantity go to another device. This type of data splitting requires that the two devices be on different SCSI controllers. If two SCSI devices are used in the Transfer Unit, they must both have the same blocksize. The blocksize can be determined by sending the MMEM:SCSI:BSIZ? command.

SEQuence[1|2|3|4]:ADD command

Append an operation to the specified Sequence.

Command Syntax:

```
SEQuence[1|2|3|4]:ADD <Operation>, <Count>, <Address>, <Misc>

<Operation>::=number
              limits: 0:65535

<Count>::=number
          limits: 0:4294967295

<Address>::=number
            limits: 0:4294967295

<Misc>::=number
         limits: 0:4294967295
```

Example Statements:

```
OUTPUT 70918;"SEQUENCE:ADD #h1000,#h10,0,#h03000800"
OUTPUT 70918;"seq3:add #h3012,65536,2048,0"
```

Attribute Summary:

Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

Add the specified operation to the end of the specified Sequence. The maximum number of operations in a single Sequence is 100. The number of operations currently in the Sequence may be determined by sending the SEQ:SIZE? command.

The list of Sequence operations should be cleared by sending the SEQ:DEL:ALL command before adding new Sequence operations using this command.

Note

For a list and detailed description of all Sequence operations that may be added using this command and an explanation of the above parameters see “Sequence Operations Reference” starting on page 141.

SEQuence[1|2|3|4]:BEGin command

Begin a Sequence for throughput or playback data transfer.

Command Syntax:

```
SEQuence[1|2|3|4]:BEGin <Type>, <Byte Count>, <Session>
<Type>::=THRoughput|VPLayback|LPLayback|MTHRoughput|MPLayback
<Byte Count>::=number
              limits: 1:9223372036854775807
<Session>::=number
           limits: 1:12
```

Example Statements:

```
OUTPUT 70918;":SEQ:BEG VPL,262144,1"
OUTPUT 70918;"sequence3:begin throughput,#h200000000,2"
```

Attribute Summary:
Preset State: not applicable
Synchronization Required: yes
SCPI Compliance: instrument-specific

Description:
Begin execution of the specified Sequence. A Session must already be set up before sending this command. See the commands MMEM:SESS:*.

<Type> indicates whether the Sequence will be a throughput or a playback. The internal software needs to be told this in order to get data flowing in the right direction. It also needs to be told whether a playback will be to the VXI bus, or to the LBUS and whether block and frame markers will be saved with the data on a throughput or restored on a playback.

Transfer type name	Transfer action performed
THRoughput	Throughput without block and frame markers. Data may subsequently be played back with either VPLayback or LPLayback.
VPLayback	VXI playback only. For use with data previously transferred with THRoughput.
LPLayback	Local bus playback only without block and frame markers. For data previously transferred with THRoughput.
MTHRoughput	Throughput in which all local bus data will have the block and frame markers embedded in the data stream. Data acquired via MTHROUGHPUT may only be played back using MPLAYBACK.
MPLayback	Local bus playback that will reconstruct the block and frame markers from information in the data stream. For use with data previously transferred with MTHRoughput.

<Byte Count> is a 64-bit integer that indicates the total number of bytes that will be transferred by the Sequence. Once this byte count is reached, the Sequence will terminate. This byte count is used to determine how many SCSI logical blocks will be transferred to/from the devices. The software will round up to the next even number of logical blocks

<Session> indicates which Session will be used for this Sequence. The commands to initialize the Session must already have been successfully sent to the module. See MMEM:SESS:*.

SEQuence[1|2|3|4]:DELete:ALL command

Remove all elements from the specified Sequence list.

Command Syntax: `SEQuence[1|2|3|4]:DELete:ALL`

Example Statements:

```
OUTPUT 70918;":SEQUENCE4:DELETE:ALL"
OUTPUT 70918;"seq2:del:all"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Delete all elements of the Sequence. This is the only command that removes elements from a Sequence. This command should be sent before adding elements to a Sequence.

SEQuence[1|2|3|4]:SIZE? query

Return the number of elements in the Sequence.

Query Syntax: `SEQuence[1|2|3|4]:SIZE?`

Example Statements:
```
OUTPUT 70918;":SEQ2:SIZE?"
OUTPUT 70918;"sequence:size?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: The number returned should equal the number of times SEQ:ADD has been sent since the last SEQ:DEL:ALL, *RST or powerup. The maximum number of elements in a Sequence is 100.

SEQuence[1|2|3|4]:TRANsferred? query

Return the number of bytes transferred in the Sequence.

Query Syntax: `SEQuence[1|2|3|4]:TRANsferred?`

Example Statements:

```
OUTPUT 70918;":SEQUENCE2:TRANSFERRED?"
OUTPUT 70918;"seq4:tran?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: If the Sequence completed without errors and was not aborted, the returned value will be almost equal to the number of bytes specified with the SEQ:BEG command. Otherwise, the returned value will indicate the number of bytes that were successfully transferred.

STATus:OPERation:CONDition? query

Reads the Operation Status condition register.

Query Syntax: `STATus:OPERation:CONDition?`

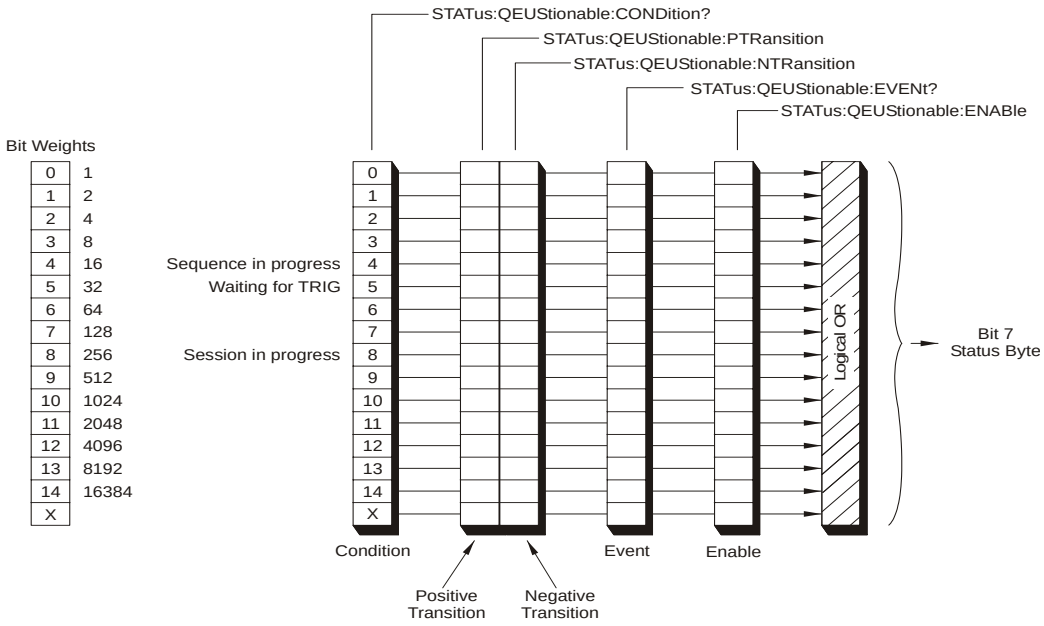
Example Statements:
```
OUTPUT 70918;":STATUS:OPERATION:CONDITION?"
OUTPUT 70918;"status:operation:condition?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This query returns the sum of the decimal weights of all bits currently set to 1 in the Operation Status condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See Operation Status Register Set on page 196 for a definition of bits in the register set.



STATus:OPERation:ENABle command/query

Sets and queries bits in the Operation Status enable register.

Command Syntax:

```
STATus:OPERation:ENABle <Bit Mask>

<Bit Mask>::=number
             limits: 0:32767
```

Example Statements:

```
OUTPUT 70918;":STATUS:OPER:ENAB 304"
OUTPUT 70918;"status:operation:enable 32"
```

Query Syntax:

```
STATus:OPERation:ENABle?
```

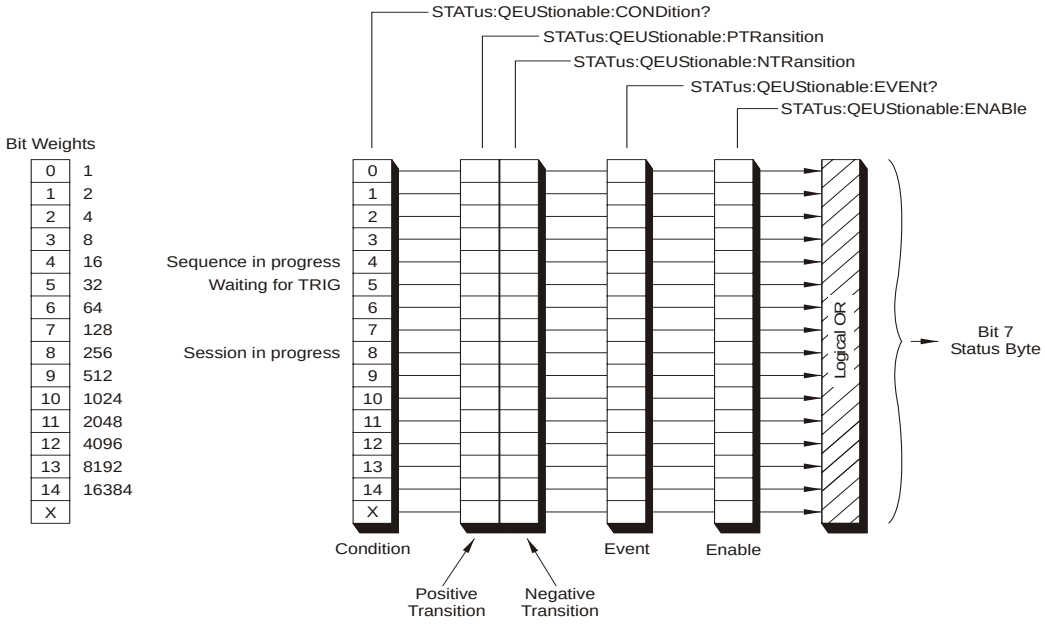
Return Format: Integer

Attribute Summary: Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: To set a single bit in the Operation Status enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the sum of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

All bits are initialized to 0 on powerup or when the STAT:PRES command is sent. However, the current setting of bits is *not* modified when the *RST command is sent.

See Operation Status Register Set on page 196 for a definition of bits in the register set.



STATus:OPERation[:EVENt]? query

Reads and clears the Operation Status event register.

Query Syntax: `STATus:OPERation[:EVENt]?`

Example Statements:
```
OUTPUT 70918;":STATUS:OPERATION?"
OUTPUT 70918;"stat:oper:even?"
```

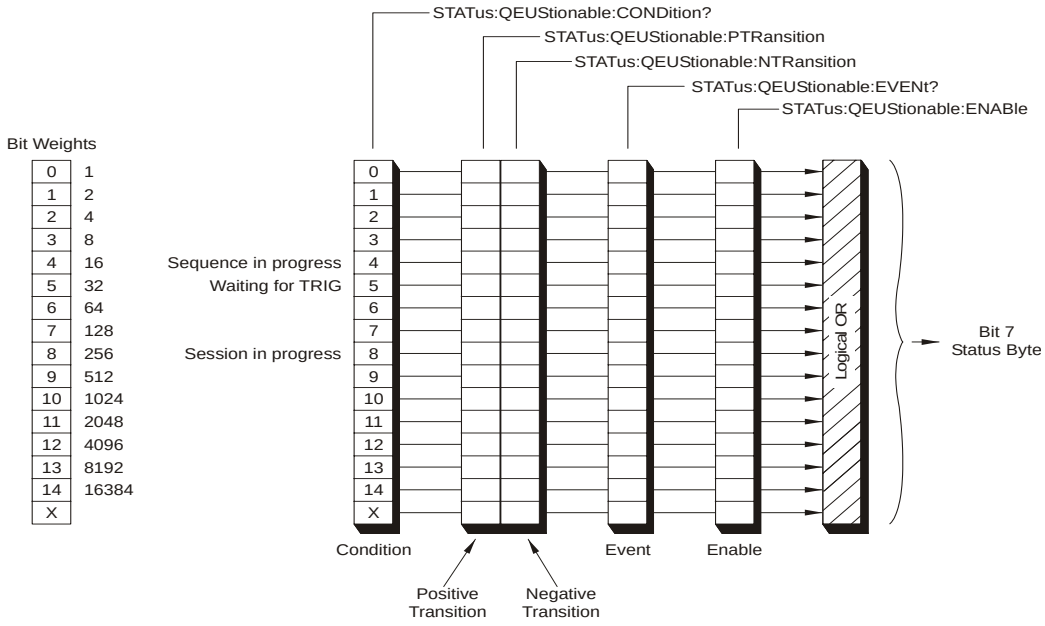
Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This query returns the sum of the decimal weights of all bits currently set to 1 in the Operation Status event register. (The decimal weight of a bit is 2^n, where n is the bit number.)

Note The Operation Status event register is automatically cleared after it is read by this query.

See Operation Status Register Set on page 196 for a definition of bits in the register set.



STATus:OPERation:NTRansition command/query

Sets and queries bits in the Operation Status negative transition register.

Command Syntax:

```
STATus:OPERation:NTRansition <Bit mask>

<Bit mask>::=number
             limits: 0:32767
```

Example Statements:

```
OUTPUT 70918;":STAT:OPER:NTR 256"
OUTPUT 70918;"status:operation:ntransition 48"
```

Query Syntax:

```
STATus:OPERation:NTRansition?
```

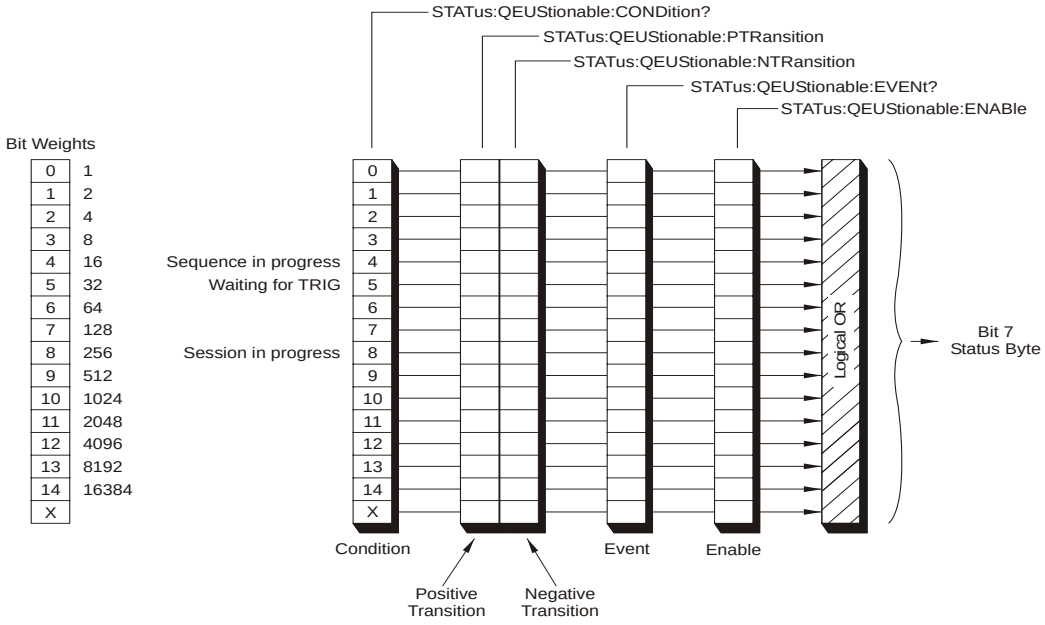
Return Format: Integer

Attribute Summary:
Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: To set a single bit in the Operation Status negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the sum of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

All bits are initialized to 0 on powerup or when the STAT:PRES command is sent. However, the current setting of bits is *not* modified when the *RST command is sent.

See Operation Status Register Set on page 196 for a definition of bits in the register set.



STATus:OPERation:PTRansition command/query

Sets and queries bits in the Operation Status positive transition register.

Command Syntax:

```
STATus:OPERation:PTRansition <Bit mask>

<Bit mask>::=number
             limits: 0:32767
```

Example Statements:

```
OUTPUT 70918;":STAT:OPER:PTR 304"
OUTPUT 70918;"status:operation:ptransition 32"
```

Query Syntax:

```
STATus:OPERation:PTRansition?
```

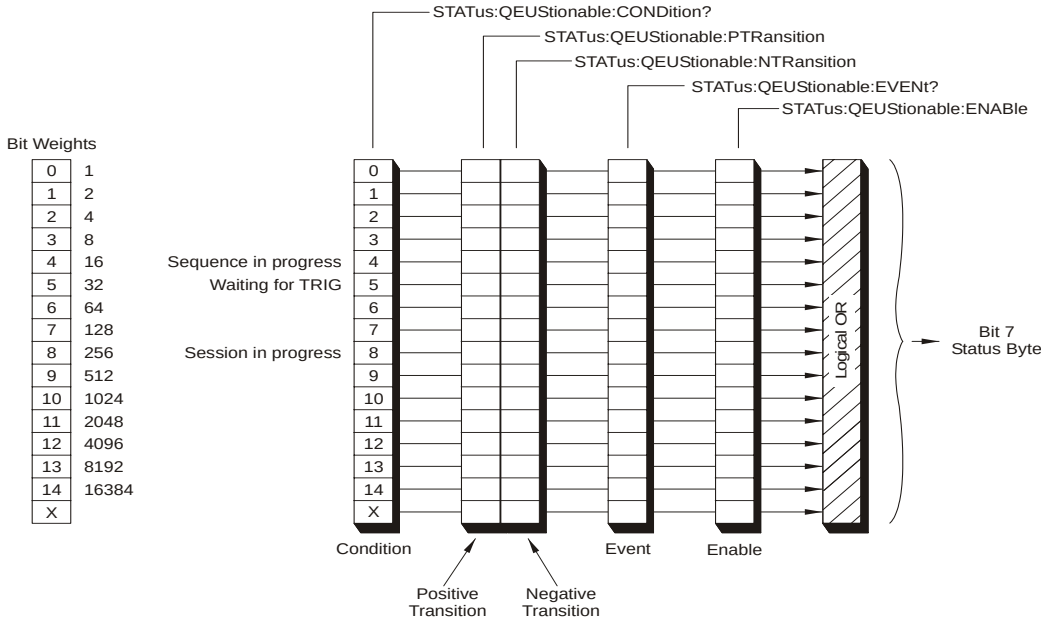
Return Format: Integer

Attribute Summary: Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: To set a single bit in the Operation Status positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the sum of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

All bits are initialized to 1 on powerup or when the STAT:PRES command is sent. However, the current setting of bits is *not* modified when the *RST command is sent.

See Operation Status Register Set on page 196 for a definition of bits in the register set.



STATus:PRESet command

Sets bits in most enable and transition registers to their default state.

Command Syntax: `STATus:PRESet`

Example Statements:

```
OUTPUT 70918;":STATUS:PRESET"
OUTPUT 70918;"stat:pres"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: STATUS:PRESet has the effect of bringing all events to the second level register sets (Questionable Status and Operation Status) *without* creating an SRQ or reflecting events in a serial poll.

It also affects these register sets (Questionable Status and Operation Status) as follows:

- Sets all enable register bits to 0.
- Sets all positive transition register bits to 1.
- Sets all negative transition register bits to 0.

STATus:QUEStionable:CONDition? query

Reads the Questionable Status condition register.

Query Syntax: `STATus:QUEStionable:CONDition?`

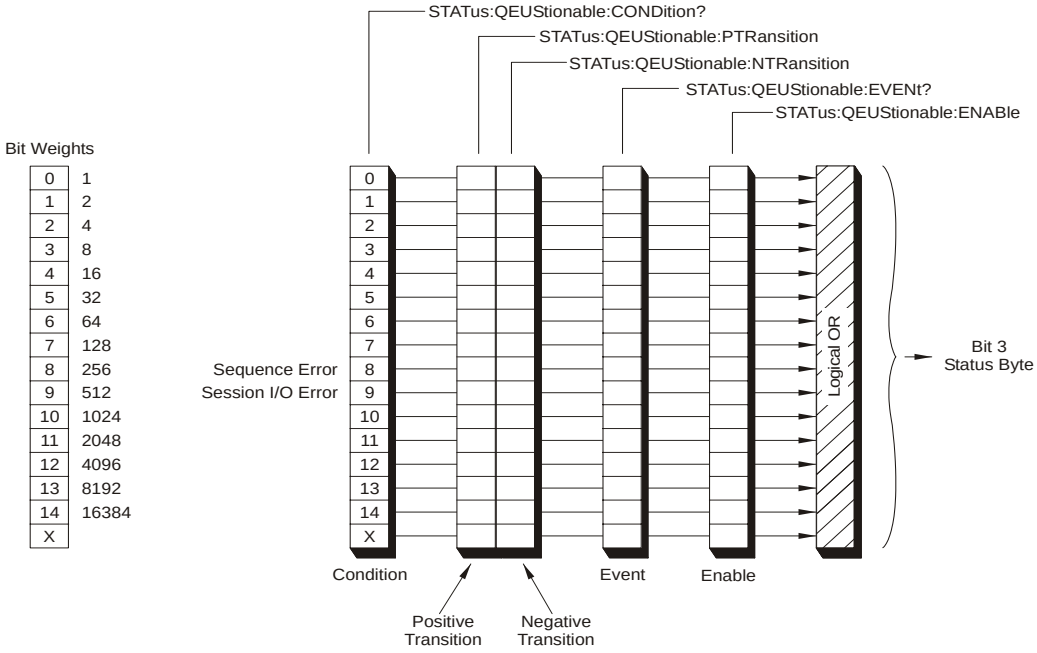
Example Statements:
```
OUTPUT 70918;":STAT:QUES:COND?"
OUTPUT 70918;"status:questionable:condition?"
```

Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Status condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See Questionable Status Register Set on page 194 for a definition of bits in the register set.



STATus:QUEStionable:ENABle command/query

Sets and queries bits in the Questionable Status enable register.

Command Syntax:
```
STATus:QUEStionable:ENABle <Bit Mask>

<Bit Mask>::=number
             limits: 0:32767
```

Example Statements:
```
OUTPUT 70918;":STAT:QUES:ENAB 256"
OUTPUT 70918;"status:questionable:enable 512"
```

Query Syntax:
```
STATus:QUEStionable:ENABle?
```

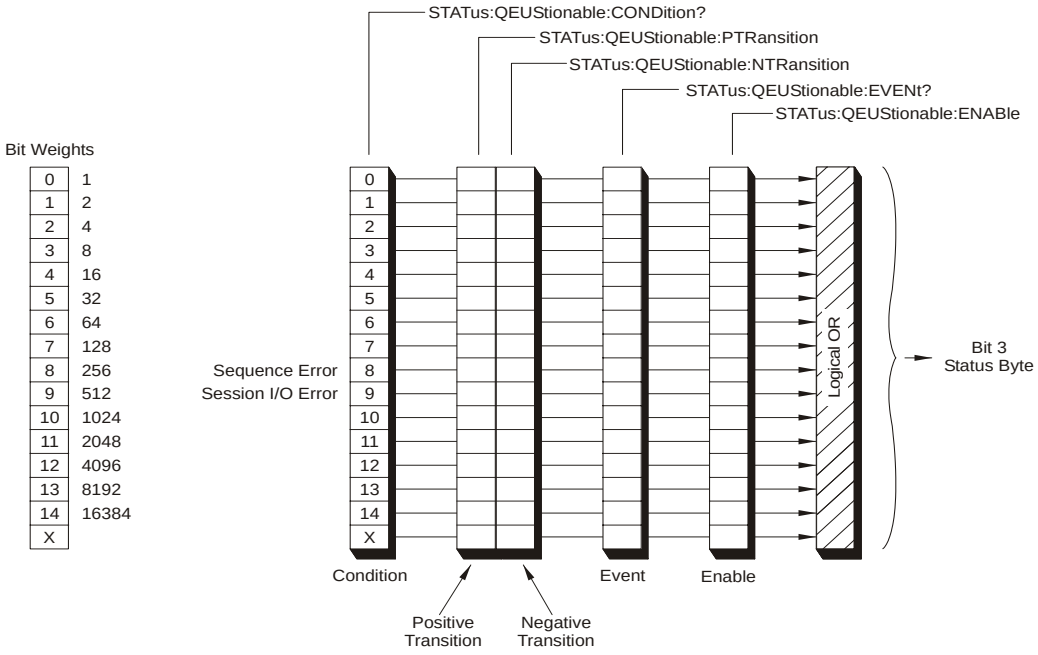
Return Format: Integer

Attribute Summary: Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: To set a single bit in the Questionable Status enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the sum of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

All bits are initialized to 0 on powerup or when the STAT:PRES command is sent. However, the current setting of bits is *not* modified when the *RST command is sent.

See Questionable Status Register Set on page 194 for a definition of bits in the register set.



STATus:QUEStionable[:EVENt]? query

Reads and clears the Questionable Status event register.

Query Syntax: `STATus:QUEStionable[:EVENt]?`

Example Statements:
```
OUTPUT 70918;":STATUS:QUESTIONABLE:EVENT?"
OUTPUT 70918;"status:questionable?"
```

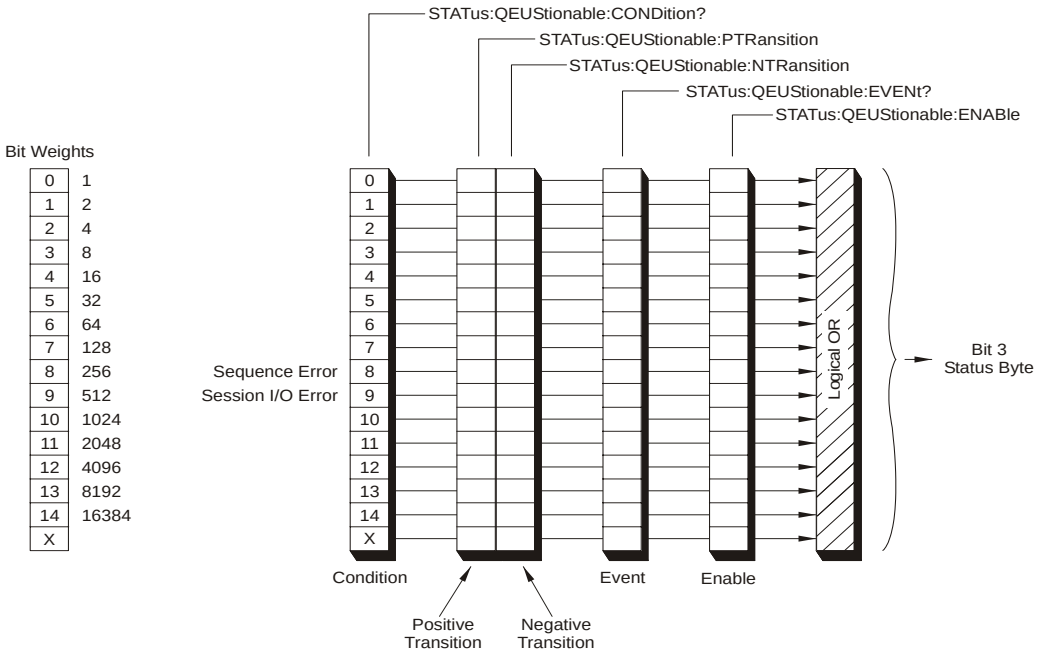
Return Format: Integer

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Status event register. (The decimal weight of a bit is 2^n, where n is the bit number.)

Note The Questionable Status event register is automatically cleared after it is read by this query.

See Questionable Status Register Set on page 194 for a definition of bits in the register set.



STATus:QUEStionable:NTRansition command/query

Sets and queries bits in the Questionable Status negative transition register.

Command Syntax:
```
STATus:QUEStionable:NTRansition <Bit mask>

<Bit mask>::=number
             limits: 0:32767
```

Example Statements:
```
OUTPUT 70918;":STAT:QUES:NTR 768"
OUTPUT 70918;"Status:Questionable:Ntransition 256"
```

Query Syntax:
```
STATus:QUEStionable:NTRansition?
```

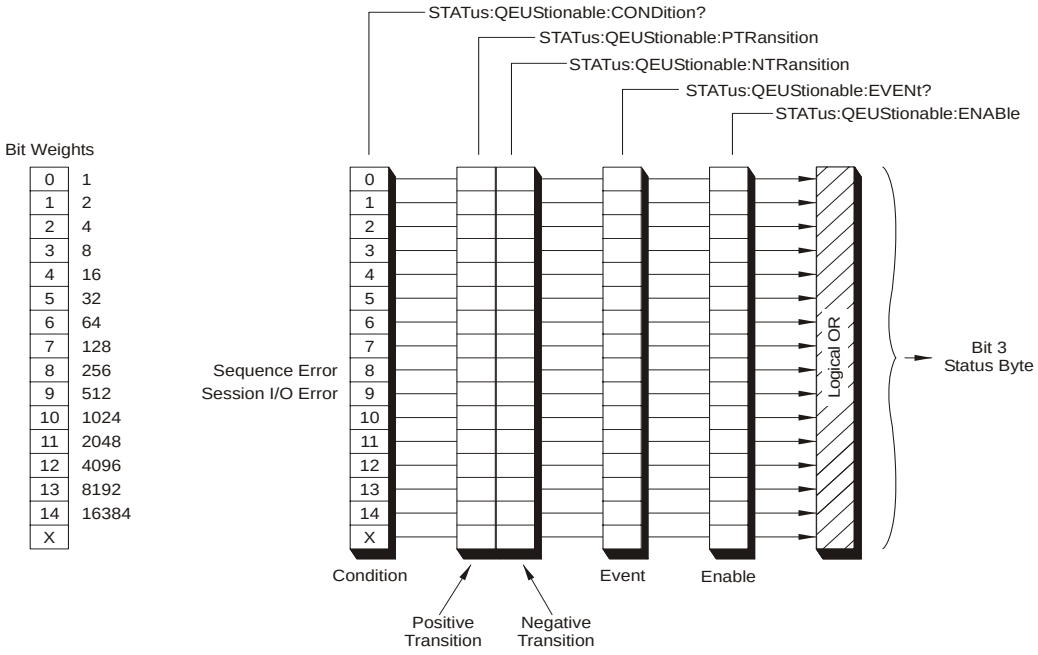
Return Format: Integer

Attribute Summary: Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: To set a single bit in the Questionable Status negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the sum of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

All bits are initialized to 0 on powerup or when the STAT:PRES command is sent. However, the current setting of bits is *not* modified when the *RST command is sent.

See Questionable Status Register Set on page 194 for a definition of bits in the register set.



STATus:QUEStionable:PTRansition command/query

Sets and queries bits in the Questionable Status positive transition register.

Command Syntax:
```
STATus:QUEStionable:PTRansition <Bit mask>

<Bit mask>::=number
            limits: 0:32767
```

Example Statements:
```
OUTPUT 70918;":STATUS:QUESTIONABLE:PTRANSITION 256"
OUTPUT 70918;"stat:ques:ptr 512"
```

Query Syntax:
```
STATus:QUEStionable:PTRansition?
```

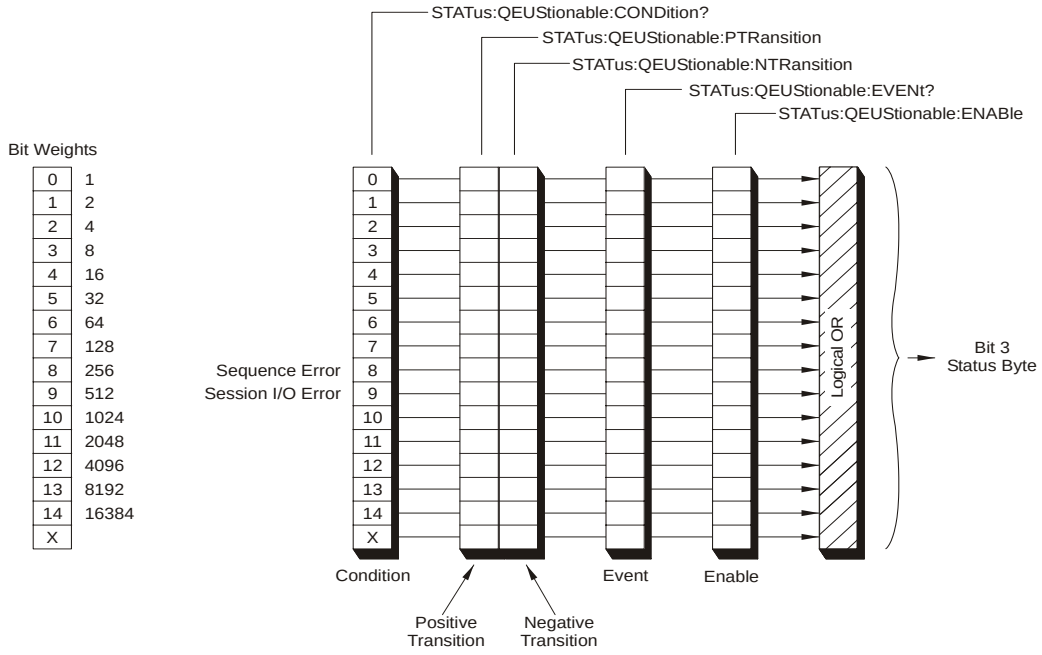
Return Format: Integer

Attribute Summary: Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: To set a single bit in the Questionable Status positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the sum of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

All bits are initialized to 1 on powerup or when the STAT:PRES command is sent. However, the current setting of bits is *not* modified when the *RST command is sent.

See Questionable Status Register Set on page 194 for a definition of bits in the register set.



SYSTem:ABORt command

Aborts a data transfer Session and/or Sequence.

Command Syntax: `SYSTem:ABORt`

Example Statements:

```
OUTPUT 70918;":SYSTEM:ABORT"
OUTPUT 70918;"syst:abor"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: Any data transfer Session in progress is aborted. The Session data structures will not be altered and all Transfer Units and SCSI devices will remain open. The local bus is placed into the reset state. Any Sequence in progress is aborted. Sequence data structures are updated such that a SEQ:TRAN? query will correctly indicate the number of bytes actually transferred during the Sequence.

SYSTem:COMMunicate:SCSI[:SELF]:ADDRess command/query

Changes the SCSI address on a VT2216A individual SCSI bus controller.

Command Syntax:

```
SYSTem:COMMunicate:SCSI[:SELF]:ADDRess <Controller>,<Bus address>
<Controller>::=A|B
<SCSI address>::=number
                limits 0:15
```

Example Statements:

```
OUTPUT 70918;":SYSTEM:COMMUNICATE:SCSI:SELF:ADDRESS A,5"
OUTPUT 70918;"syst:comm:scsi:addr b,13"
```

Query Syntax:

```
SYSTem:COMMunicate:SCSI:SELF:ADDRess? <Controller>
```

Return Format: Integer

Attribute Summary:
Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: The VT2216A SCSI bus address is an internal address that is set on powerup by switches CA0 and CA1. When multiple VT2216As are on the same SCSI bus, it must be ensured that no two SCSI controllers share the same SCSI bus address before any SCSI accesses can be performed. This command allows one to query and change the SCSI address values of the VT2216A, overriding the switch settings. (See “Installing the VT2216A” starting on page 19 for information on setting the switches manually.) It is also necessary to change the VT2216A SCSI address if an external host or other SCSI device is at the same address as any SCSI bus controller.

The query returns the current address.

Note

This address should not be confused with the SCSI logical address of a device that is designated by MMEM:SCSI:OPEN. The address set by SYST:COMM:SCSI:ADDR is only used internally by the VT2216A and will not be used by any SCPI commands.

SYSTem:ERRor? query

Returns one error message from the module's error queue.

Query Syntax: `SYSTem:ERRor?`

Example Statements:
```
OUTPUT 70918;":SYSTEM:ERROR?"
OUTPUT 70918;"syst:err?"
```

Return Format: Integer
STRING

Attribute Summary: Preset State: not affected by Preset
Synchronization Required: no
SCPI Compliance: confirmed

Description: The error queue temporarily stores up to ten error messages. When the SYST:ERR query is sent, one message is moved from the error queue to the output queue so the controller can read the message. The error queue delivers messages to the output queue in the order received.

If more than ten error messages are reported before any are read from the queue, the oldest error messages are saved. The last error message indicates that too many error messages were received for the queue.

Note: The error queue is cleared when the VXI system is turned on and when the *CLS command is sent.

SYSTem:VERSion? query

Returns the SCPI version to which the module complies.

Query Syntax: `SYSTem:VERSion?`

Example Statements:
```
OUTPUT 70918;":SYSTEM:VERSION?"
OUTPUT 70918;"syst:vers?"
```

Return Format: YYYY.V

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: confirmed

Description: The Ys represent the SCPI year-version and the V represents the revision number for that year.

The VT2216A will return 1994.0.

VINStrument[:CONFigure]:LBUS [:MODE] RESet|NORMal|PIPE — command/query

Configures the local bus.

Command Syntax:

```
VINStrument[:CONFigure]:LBUS[:MODE] <Lbus Mode>
<Lbus Mode>::=RESet|PIPE|NORMal
```

Example Statements:

```
OUTPUT 70918;":VINSTRUMENT:CONFIGURE:LBUS:MODE RESET"
OUTPUT 70918;"vins:lbus norm"
```

Query Syntax:

```
VINStrument[:CONFigure]:LBUS[:MODE]?
```

Return Format: CHAR

Attribute Summary:
Preset State: RESet
Synchronization Required: no
SCPI Compliance: instrument-specific

Description:

The local bus interface has strict requirements as to the order in which modules in a VXI mainframe have their local bus interface reset. On powerup or whenever any module in the mainframe is put in the reset state, all modules should be placed into the reset state from left to right. Then all modules can be put into the un-reset state from left to right.

Sending VINStrument:CONfigure:LBUS:MODE RESet, places the VT2216A local bus interface into the LBUS RESet state. Sending either PIPE or NORMAL takes the local bus interface out of the LBUS RESet state. In order for the VT2216A to use the local bus, the mode must be set to NORMal.

Sending VINStrument:LBUS PIPE puts the local bus interface into a state such that all local bus data from the module to the left is automatically routed to the module on the right. This is useful if one desires to route local bus data past the VT2216A to other modules rather than have the VT2216A participate in any local bus throughput or playback. In this mode, the local bus cannot be used in a Sequence operation or via the LBUS:READ:BUFFer or LBUS:WRITe:BUFFer commands. The behavior is undefined if this value is set to PIPE then a Sequence operation is executed that either reads from or writes to the local bus.

When transitioning from PIPE to NORMal mode, one additional block will be piped after the change. In order to make this transition easier to coordinate it is best to reset the local bus on all modules then go to NORMal.

This command may be used instead of VINStrument:LBUS:RESet to place the VT2216A local bus in the un-reset state.

VINStrument:LBUS:RESet command

Resets the local bus.

Command Syntax: `VINStrument:LBUS:RESet`

Example Statements:

```
OUTPUT 70918;":VINSTRUMENT:LBUS:RESET "
OUTPUT 70918;"vins:lbus:res"
```

Attribute Summary: Preset State: not applicable
Synchronization Required: no
SCPI Compliance: instrument-specific

Description: The local bus interface has strict requirements as to the order in which modules in a VXI mainframe have their local bus interface reset. On powerup or whenever any module in the mainframe is put in the reset state, all modules should be placed into the reset state from left to right. Then all modules can be put into the un-reset state from left to right.

This command toggles the local bus reset state for the VT2216A; first going into the reset state, then back out. Once this is completed the local bus mode is NORMal.

This command is *not* required if the VINStrument:CONFigure:LBUS:MODE command used to configure the local bus.

Errors

SCPI Command Errors

Error Number	Description
-100	Command error. This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that a Command Error as defined in *IEEE 488.2*, 11.5.1.1.4 has occurred.
-101	Invalid character. A syntactic element contains a character that is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of errors -114, -121, -141 and perhaps others.
-102	Syntax error. An unrecognized command or data type was encountered; for example, a string was received when the device does not accept strings.
-103	Invalid separator. The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit, *EMC 1 :CH1:VOLTS 5.
-104	Data type error. The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.
-105	GET not allowed. A Group Execute Trigger was received within a program message (see *IEEE 488.2*, 7.7).
-108	Parameter not allowed. More parameters were received than expected for the header; for example, the *EMC common command only accepts one parameter, so receiving *EMC 0,,1 is not allowed.
-109	Missing parameter. Fewer parameters were received than required for the header; for example, the *EMC common command requires one parameter, so receiving *EMC is not allowed.
-110	Command header error. An error was detected in the header. This error message is used when the device cannot detect the more specific errors described for errors -111 through -119.
-111	Header separator error. A character that is not a legal header separator was encountered while parsing the header; for example, no white space followed the header, thus *GMC"MACRO" is an error.
-112	Program mnemonic too long. The header contains more than twelve characters (see *IEE488.2*, 7.6.1.4.1).
-113	Undefined header. The header is syntactically correct, but it is undefined for this specific device; for example, *XYZ is not defined for any device.
-114	Header suffix out of range. The value of a numeric suffix attached to a program mnemonic, see Syntax and Style section 6.2.5.2, makes the header invalid.
-120	Numeric data error. This error, as well as errors -121 through -129, are generated when parsing a data element that appears to be numeric, including the nondecimal numeric types. This particular error message is used if the device cannot detect a more specific error.

SCPI Command Errors

Error Number	Description
-121	Invalid character in number. An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric or a “9” in octal data.
-123	Exponent too large. The magnitude of the exponent was larger than 32000 (see *IEEE 488.2*, 7.7.2.4.1).
-124	Too many digits. The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see *IEEE 488.2*, 7.7.2.4.1).
-128	Numeric data not allowed. A legal numeric data element was received, but the device does not accept one in this position for the header.
-130	Suffix error. This error, as well as errors -131 through -139, are generated when parsing a suffix. This particular error message is used if the device cannot detect a more specific error.
-131	Invalid suffix. The suffix does not follow the syntax described in *IEEE 488.2*, 7.7.3.2 or the suffix is inappropriate for this device.
-134	Suffix too long. The suffix contained more than 12 characters (see IEEE 488.2, 7.7.3.4).
-138	Suffix not allowed. A suffix was encountered after a numeric element which does not allow suffixes.
-140	Character data error. This error, as well as errors -141 through -149, are generated when parsing a character data element. This particular error message is used if the device cannot detect a more specific error.
-141	Invalid character data. Either the character data element contains an invalid character or the particular element received is not valid for the header.
-144	Character data too long. The character data element contains more than twelve characters (see IEEE 488.2, 7.7.1.4).
-148	Character data not allowed. A legal character data element was encountered where prohibited by the device.
-150	String data error. This error, as well as errors -151 through -159, are generated when parsing a string data element. This particular error message is used if the device cannot detect a more specific error.
-151	Invalid string data. A string data element was expected, but was invalid for some reason (see IEEE 488.2, 7.7.5.2); for example, an END message was received before the terminal quote character.
-158	String data not allowed. A string data element was encountered but was not allowed by the device at this point in parsing.
-160	Block data error. This error, as well as errors -161 through -169, are generated when parsing a block data element. This particular error message is used if the device cannot detect a more specific error.
-161	Invalid block data. A block data element was expected, but was invalid for some reason (see IEEE 488.2, 7.7.6.2); for example, an END message was received before the length was satisfied.
-168	Block data not allowed. A legal block data element was encountered but was not allowed by the device at this point in parsing.

SCPI Command Errors

Error Number	Description
-170	Expression error. This error, as well as errors -171 through -179, are generated when parsing an expression data element. This particular error message is used if the device cannot detect a more specific error.
-171	Invalid expression. The expression data element was invalid (see IEEE 488.2, 7.7.7.2); for example, unmatched parentheses or an illegal character.
-178	Expression data not allowed. A legal expression data was encountered but was not allowed by the device at this point in parsing.
-181	Invalid outside macro definition. Indicates that a macro parameter placeholder ($<number) was encountered outside of a macro definition.
-183	Invalid inside macro definition. Indicates that the program message unit sequence, sent with a *DDT or *DMC command, is syntactically invalid (see IEEE 488.2, 10.7.6.3).

SCPI Execution Errors

Error Number	Description
-200	Execution error. This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that an Execution Error as defined in *IEEE 488.2*, 11.5.1.1.5 has occurred.
-220	Parameter error. Indicates that a program data element related error occurred. This error message is used when the device cannot detect the more specific errors described for errors -221 through -229.
-221	Settings conflict. Indicates that a legal program data element was parsed but could not be executed due to the current device state (see *IEEE 488.2*, 6.4.5.3 and 11.5.1.1.5.)
-222	Data out of range. Indicates that a legal program data element was parsed but could not be executed because the interpreted value was outside the legal range as defined by the device (see *IEEE 488.2*, 11.5.1.1.5.)
-223	Too much data. Indicates that a legal program data element of block, expression, or string type was received that contained more data than the device could handle due to memory or related device-specific requirements.
-224	Illegal parameter value. Used where exact value, from a list of possibles, was expected.
-240	Hardware error. Indicates that a legal program command or query could not be executed because of a hardware problem in the device. Definition of what constitutes a hardware problem is completely device-specific. This error message is used when the device cannot detect the more specific errors described for errors -241 through -249.
-241	Hardware missing. Indicates that a legal program command or query could not be executed because of missing device hardware; for example, an option was not installed. Definition of what constitutes missing hardware is completely device-specific.
-250	Mass storage error. Indicates that a mass storage error occurred. This error message is used when the device cannot detect the more specific errors described for errors -251 through -259.
-251	Missing mass storage. Indicates that a legal program command or query could not be executed because of missing mass storage; for example, an option that was not installed. Definition of what constitutes missing mass storage is device-specific.

SCPI Execution Errors

Error Number	Description
-252	Missing media. Indicates that a legal program command or query could not be executed because of a missing media; for example, no disk. The definition of what constitutes missing media is device-specific.
-253	Corrupt media. Indicates that a legal program command or query could not be executed because of corrupt media; for example, bad disk or wrong format. The definition of what constitutes corrupt media is device-specific.
-254	Media full. Indicates that a legal program command or query could not be executed because the media was full; for example, there is no room on the disk. The definition of what constitutes a full media is device-specific.
-258	Media protected. Indicates that a legal program command or query could not be executed because the media was protected; for example, the write-protect tab on a disk was present. The definition of what constitutes protected media is device-specific.
-272	Macro execution error. Indicates that a syntactically legal macro program data sequence could not be executed due to some error in the macro definition (see *IEEE 488.2*, 10.7.6.3.)
-273	Illegal macro label. Indicates that the macro label defined in the *DMC command was a legal string syntax, but could not be accepted by the device (see *IEEE 488.2*, 10.7.3 and 10.7.6.2); for example, the label was too long, the same as a common command header or contained invalid header syntax.
-276	Macro recursion error. Indicates that a syntactically legal macro program data sequence could not be executed because the device found it to be recursive (see *IEEE 488.2*, 10.7.6.6).
-277	Macro redefinition not allowed. Indicates that a syntactically legal macro label in the *DMC command could not be executed because the macro label was already defined (see *IEEE 488.2*, 10.7.6.4).
-278	Macro header not found. Indicates that a syntactically legal macro label in the *GMC? query could not be executed because the header was not previously defined.

SCPI Device-Specific Errors

Error Number	Description
-310	System error. Indicates that some error termed “system error” by the device, has occurred. This code is device-dependent.
-311	Memory error. Indicates that an error was detected in the device’s memory. The scope of this error is device-dependent.
-315	Configuration memory lost. Indicates that nonvolatile configuration data saved by the device has been lost. The meaning of this error is device-specific.
-321	Out of memory.
-330	Self-test failed.
-350	Queue overflow. A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.

SCPI Query Errors

Error Number	Description
-400	Query error. This is the generic query error for devices that cannot detect more specific errors. This code indicates only that a Query Error as defined in *IEEE 488.2*, 11.5.1.1.7 and 6.3 has occurred.
-410	Query INTERRUPTED. Indicates that a condition causing an INTERRUPTED Query error occurred (see *IEE 448.2*, 6.3.2.3); for example, a query followed by DAB or GET before a response was completely sent.
-420	Query UNTERMINATED. Indicates that a condition causing an UNTERMINATED Query error occurred (see *IEEE 488.2*, 6.3.2.2); for example, the device was addressed to talk and an incomplete program message was received.
-430	Query DEADLOCKED. Indicates that a condition causing an DEADLOCKED Query error occurred (see *IEEE 488.2*, 6.3.1.7); for example, both input buffer and output buffer are full and the device cannot continue.
440	Query UNTERMINATED after indefinite response. Indicates that a query was received in the same program message after an query requesting an indefinite response was executed (see *IEEE 488.2*, 6.5.7.5).

VT2216A-Specific Errors

Error Number	Description
6201	Device not open. A read, write, or other command was used to access a device that was not already open. See MMEM:SCSI:OPEN.
6202	Device not ready. A command was used to access a device that had been opened with the dontStartUnit bit set. Try closing the device and re-opening it without that bit set in the mode word.
6203	Device already open. A second open was attempted on a device that was already open.
6204	Device incompatible. This error is returned if a MMEM:TUN:OPEN is sent with two devices on the same SCSI bus or with the devices swapped (specifying the SCSI B device first and then the SCSI A device). This error is returned from MMEM:SESS:ADD if a two device Transfer Unit is added to a Session that already contains one device Transfer Units or vice versa; or if a single device Transfer Unit on SCSI A is added to a Session containing single device Transfer Units on SCSI B or vice versa. Also, there are a few commands that will only execute on a certain type of device, MMEM:SCSI:ERAS will only execute on optical memory devices.
6205	Device error. A device returned an error after attempting to perform some operation for which there is no further information.
6206	Session full. A MMEM:SESS:ADD was attempted on a Session that already contained the maximum number of Transfer Units.
6207	Session busy. A SCPI command attempted to use a Session that was already performing some SCSI operation. Use of the Session Busy bit in the Operation Status register may help avoid this error.
6208	Session empty. A SCPI command attempted to use a Session that did not contain any Transfer Units.

VT2216A-Specific Errors

Error Number	Description
6209	Sequence full. A SEQ:ADD was attempted on a Sequence that already contained the maximum number of Sequence operations.
6210	Sequence busy. A SEQ:BEG command was attempted when a Sequence was already running.
6211	Sequence empty. A SEQ:BEG command was attempted on a Sequence containing no Sequence operations.
6212	Local bus busy. The local bus chip was not in the paused state when a SEQ:BEG was attempted.
6213	Require even block count. A SCPI command with an odd number of SCSI blocks attempted an opertion on a Session containing split Transfer Units. Only even block counts are accepted on split Sessions.
6214	Device timeout. A SCSI device timed out when a SCSI command was sent to it.
6215	Sequence bus error. The sequencer detected a bus error while running a Sequence. This could be due to a VXI device not existing at the expected logical address or due to memory not existing at an expected location. Check the operations in the Sequence for errors.
6216	(This error is only applicable to the Agilent/HP 1562A/B/C. Maximum safe disk temperature exceeded. This error indicates that the disk drive has exceeded 72°C. It has been spun down if it was spinning. The temperature check is done every 30 minutes after powerup. This check can only be done for HP disks due to the use of non-SCSI-defined commands. The SCSI controller and bus address will be included in the error message. For example, the results of SYST:ERR? might return: `16, "Maximum safe disk temperature exceeded; SCSI B, bus address 0"`
6217	Write to a read-only device. Either the device was opened with the read-only bit in the open mode set, or the device is write-protected (i.e. a tape in the DAT is write protected) and a write operation was attempted to this device.

LIF Library Reference

Getting Started

Why Use the LIF Library?

LIF (Logical Interchange Format) is a directory and file format used to exchange files among various computer systems and instruments. Any VT2216A Session, including one or more disks, may be formatted as a LIF volume.

LIF library functions provide a higher level of access to VT2216A data. For example, Sessions can be set up by using LIF library functions as an alternative to using MMEM:SCSI:OPEN, MMEM:TUN:OPEN and MMEM:SESS:ADD. Data can then be transferred using either LIF functions, Sequences, or SCPI commands.

An advantage of using the LIF library to access a VT2216A Session as a LIF volume is that multiple data acquisitions may easily be stored with each group of data identified by its own name and size in a directory. Furthermore, a single disk volume may be directly accessed by a host computer when connected via a SCSI cable.

Special Considerations for the LIF Library

The implementation of the LIF library for the VT2216A involves some special constraints and conventions:

- The number of open volumes is limited to the number of available Sessions on the VT2216A. A volume describes a single file-system which may exist on a single device or may cross several devices as do Sessions on the VT2216A. Many files may be accessed simultaneously on one volume.
- The LIF library assumes that every SCSI block in a Session is the same size. This implies that for Sessions involving striping each Transfer Unit must have the same SCSI block size. This is only an issue for striped Sessions since both devices within a Transfer Unit must already be the same size due to restrictions imposed by the VT2216A. The additional restriction imposed by the LIF libraries requires that all Transfer Units within a Session also have the same SCSI block size.
- This library assumes that every volume starts at the beginning of all devices which make up the volume. For example, if a Session is built from individual devices, the MMEM:TUN:OPEN command will always be sent with the starting SCSI block parameter set to 0.
- The LIF library can read and write only BDAT files. Files of other types may be written to volumes but they cannot be accessed by the LIF library.
- Each BDAT file on a volume begins with a 256-byte block of additional header information including, most importantly, file size. The library protects this header from reads and writes and designates that seeks to the beginning of the file go to the block following this BDAT header information.
- All LIF functions set ‘e1562_errno.’ Error codes are listed at the end of this chapter.

Naming Conventions

Several functions expect that a name be passed as a parameter. In some cases, the name refers to a single file (i.e., `e1562_fopen`) and in other cases the name refers to a volume only (i.e., `e1562_pack`).

Volume names and file names use a special naming convention to indicate which SCSI device is being referenced and also provides for a single file system consisting of many SCSI devices. The special conventions include:

- A device pair always starts with a capital ‘V’ followed by the A SCSI bus disk address then the B SCSI bus disk address.
- Addresses are designated by lower case hexadecimal numbers.
- Unused devices are designated by the placeholder ‘x’.
- File size designations (when required) are decimal ASCII.
- File names are made up of the volume name followed by a colon followed by a base LIF file name.
- A volume consisting of a striped Session has a name that includes several of the disk pairs described above, each followed by a burst block count. This is analogous to the <Count> parameter used in MMEM:SESS:ADD.

Volume name examples:

A single device at address 3 on controller B:
`"Vx3"` (SCSI A is absent; SCSI B at address 3)

A split pair of devices with the controller/address pairs of A/5 and B/11:
`"V5b"` (SCSI A at address 5; SCSI B at address 11)

A split and striped disk set using a pair of disks at A/1 and B/7 and a second pair at A/14 and B/12 with a burst block count of 512:
`"V17512Vec512"` (First pair: SCSI A at address 1; SCSI B at address 7;
Second pair: SCSI A at address 14; SCSI B at address 12)

File name example:

A file named "file1" on a volume with data split between a pair of devices:
`"Vaf:file1"`

Note

The most current LIF Library is available on-line at www.vxitech.com.

LIF Library Quick Reference

Function	Description	Page
Library Management		
e1562_closeLibrary	Release all dynamic data structures	286
e1562_initializeLibrary	Initialize the internal data structures	298
e1562_mapModule	Associate a module with an ID	299
Volume Management		
e1562_available	Return the number of LIF blocks available on the volume	283
e1562_copy	Copy a file or volume	285
e1562_defaultVolume	Define the default volume	287
e1562_dirFirst	Return information from the first valid directory entry on a volume	288
e1562_dirInit	Replace information on a volume with an empty LIF file system	289
e1562_dirNext	Return information from a subsequent valid directory entry on a volume	290
e1562_pack	Reorganize the file system to delete empty space	300
e1562_remove	Delete the specified file from the LIF directory	301
e1562_rename	Change the name of a file	302
File Management		
e1562_allocated	Return the number of 256-byte blocks allocated to the file	282
e1562_block	Return the current VT2216A Session block number	284
e1562_fclose	Flush the stream and close the associated file	291
e1562_fflush	Cause any unwritten data for a stream to be written to its associated file	292
e1562_fgetpos	Store the current file position indicator for the stream	293
e1562_fopen	Open the file specified by the string name	294
e1562_fread	Read data from a file into an array	295
e1562_fsetpos	Sets the file position indicator for the stream	296
e1562_fwrite	Write data to a file from an array	297
e1562_setEOF	Set the end-of-file marker for the file	303

LIF Commands Available from the Command Line

Command	Description	Page
e1562ls	List contents of current volume	307
e1562mv	Rename a file	308
e1562cp	Copy a file	305
e1562in	Initialize a volume	306
e1562pk	Pack a volume	309
e1562rm	Remove a file	310

VT2216A LIF Functions

e1562_allocated

Returns the physical file length.

Synopsis:

```
#include "e1562lif.h"

unsigned long e1562_allocated(e1562_FILE *stream);
```

Description: This function returns the number of 256-byte blocks allocated to the file. This may be different from the number of blocks actually occupied by data. This number is the physical length of the file, whereas the logical length of the file is the amount of data contained in the file.

Notes: To determine how much data is in the file, use `e1562_fsetpos` to go to the end of the file, then call `e1562_fgetpos` to determine the offset from the beginning of the file in bytes.

Example:

```
e1562_FILE *fp;
unsigned long LIF blocks;

fp = e1562_fopen(2,"Vx0:jan20temp","r");
LIF blocks = e1562_allocated(fp);
```

Return Value: Number of 256-byte blocks allocated to the file.

See Also: e1562_fopen on page 294

e1562_available

Returns available space and available contiguous space on a volume.

Synopsis:

```
#include "e1562lif.h"

e1562_errors e1562_available(e1562ID id, const char *volname,
                             unsigned long *totalBlocks,
                              unsigned long *largestBlock);
```

Description: This function returns the number of LIF blocks (256 bytes) available on the entire volume as well as the largest sequential number of blocks available. The value returned in ‘largestBlock’ is the size of the longest file which can be created on this volume. It is possible that a larger file might be created after calling `e1562_pack`.

Example:

```
e1562_errors error;
unsigned long Total, Large;

error = e1562_mapModule(1,"Vx1", 48);
error = e1562_available(1,"V24512Vc3512", &Total, &Large);
```

Return Value: This function returns zero if successful and an error number if it fails.

See Also:
e1562_pack on page 300
e1562_mapModule on page 299

e1562_block

Returns the next volume block number to be used.

Synopsis:

```
#include "e1562lif.h"

unsigned long e1562_block(e1562_FILE *stream,
                          unsigned long *blockSize,
                           unsigned long *byteOffset);
```

Description: This function returns the physical location on a VT2216A volume to or from which the next character would be transferred on a one byte read or write. The ‘blockSize’ argument will reflect the number of bytes in the volume’s block. For a split volume the ‘blockSize’ will be twice the size of a volume consisting of a single device.

The value pointed to by the ‘byteOffset’ argument will be set to the byte offset into the returned block number at which the next byte would be read or written on the current volume. The block returned is a SCSI block offset from the beginning of the VT2216A Session associated with the LIF volume on which the file resides.

Notes: The file must be opened with `e1562_fopen` before this function can be executed.

Example:

```
e1562_FILE *file;
unsigned long block, bytes;

file = e1562_fopen(0,"V0x:pressure", "r");
block = e1562_block(file, &bytes);
```

Return Value: If an error is detected, 0xffffffff will be returned, otherwise the Session block number is returned.

See Also: e1562_fopen on page 294

e1562_copy

Copy a file or volume.

Synopsis:

```
#include "e1562lif.h"

e1562_errors e1562_copy(e1562ID id, const char *filename,
                        const char *newfile);
```

Description: The file ‘filename’ is duplicated as ‘newfile’. This function copies a file or an entire volume.

If ‘filename’ is a file, ‘newfile’ may be either a filename or a volume name. If ‘filename’ is a volume name (must include the ‘:’) then ‘newfile’ must be a volume name.

If ‘filename’ and ‘newfile’ are on different volumes, this function requires that two VT2216A Sessions be available for use.

Notes: If ‘newfile’ already exists, an error is returned.

It is not possible to copy from one VT2216A to another with this function.

Example:

```
Copy a file to another file:
    error = e1562_copy(1,"V1x:pump3", "V1x:pump3_bak");

Copy a file to a volume:
    error = e1562_copy(0,"V55:frf4_5","V1x");

Copy a volume to a volume:
    error = e1562_copy(3,"Vab:","Vxc:");
```

Return Value: This function returns zero if successful and an error number if it fails.

See Also: e1562_mapModule on page 299

e1562_closeLibrary

Closes all volumes and files and deallocates all dynamic memory.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_closeLibrary(void);
```

Description: This function cleans up all data structures in preparation for program termination.

Example:

```
e1562_closeLibrary();
```

e1562_defaultVolume

Define the default volume.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_defaultVolume(e1562ID id, const char *volname);
```

Description: This function defines ‘volname’ as the default volume. If ‘volname’ does not specify a valid volume, an error will be returned. The default volume is initially null upon startup. Calling this function successfully allows the programmer to reference names on that volume without being required to specify the volume name as part of the filename. Any file specified without a volume name will be assumed to reside on the default volume. It is still possible to place a volume name in the filename to reference a specific volume which may not be the default volume.

Example:

```
e1562_errors err;
err = e1562_defaultVolume(1, "V05512V12512");
```

Return Value: This function returns zero if successful and an error code if it fails.

See Also: e1562_mapModule on page 299

e1562_dirFirst

Returns information from the first valid directory entry.

Synopsis:

```
#include "e1562lif.h"
e1562_dirEntry *e1562_dirFirst(e1562ID id, const char *volname,
                                e1562_dirEntry *buffer);
```

Description: This function returns information from the first valid directory entry on volume ‘volname.’ The received pointer to ‘e1562_dirEntry’ must point to an actual e1562_dirEntry structure (memory is allocated in the calling function). This function, in conjunction with `e1562_dirNext`, is used to traverse a LIF directory.

The following fields in the structure returned by this function provide information about a file in the directory.

Type	Definition
unsigned char name [12]	name of file
unsigned long date stamp	create date in BCD - YYMMDD
unsigned long time stamp	create time in BCD - HHMMSS
signed long type	type of file
unsigned long LIFstart	first 256-byte block of file
unsigned long LIFallocated	number of 256-byte blocks allocated
unsigned long sizeHigh	MS half of the byte count
unsigned long sizeLow	LS half of the byte count
unsigned long volume	volume number, MSB=1 is last volume
unsigned long reserved	not used
unsigned long entryNumber	index into directory
unsigned long session	session number
e1562 id	which VT2216A
void * vid	volume id

Notes: Deleted directory entries or otherwise invalid directory entries will never be returned from this function.

Example:

```
e1562_dirEntry entry;
e1562_dirEntry *entryp;

entryp = e1562_dirFirst(2, "V13", &entry);
```

Return Value: If an error is found or there are no files in the directory the return value will be zero, otherwise a pointer to that structure will be returned.

See Also:
e1562_dirNext on page 290
e1562_mapModule on page 299

e1562_dirInit

Replace information on a volume with an empty LIF file system.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_dirInit(e1562ID id, const char *volname);
```

Description: This function replaces all information on the designated volume with an empty LIF file system.

Note

Any information previously existing on the volume will be lost.

Example:

```
e1562_errors err;
err = e1562_dirInit(1, "Vxe");
```

Return Value: This function returns zero if successful and an error number if it fails.

See Also: e1562_mapModule on page 299

e1562_dirNext

Retrieve the next entry from a LIF directory.

Synopsis:

```
#include "e1562lif.h"

e1562_dirEntry *e1562_dirNext(e1562_dirEntry *previous);
```

Description: Assuming the received pointer to ‘e1562_dirEntry’ already contains valid directory information from a volume, this function replaces the structure with information about the next valid file in the directory.

The same structure type definitions apply to this function as listed in the description of the function `e1562_dirFirst`.

It is important that the contents of the structure returned by a previous call to `e1562_dirFirst` or `e1562_dirNext` NOT be modified before calling this function.

Notes: Deleted directory entries or otherwise invalid directory entries will never be returned from this function.

Example:

```
e1562_dirEntry entry;
e1562_dirEntry *entryp;

entryp = e1562_dirFirst(2, "V13", &entry);
entryp = e1562_dirNext(entryp);
```

Return Value: If there are no more files in the directory zero will be returned, otherwise the received pointer is returned.

See Also: e1562_dirFirst on page 288

e1562_fclose

Flush the stream and close the associated file.

Synopsis:

```
#include "e1562lif.h"

int e1562_fclose(e1562_FILE *stream);
```

Description: Any unwritten data is written to the file; any unread data is discarded. Any buffers are deallocated.

Notes: The file must be opened with `e1562_fopen` before this function can be executed.

Example:

```
e1562_FILE *f;

f = e1562_fopen (3, "Vx0:trace5,1048576", "w");
if (e1562_fclose(f) ! = 0)
    fprintf(stderr "Close failed\n");
```

Return Value: This function returns 0 if successful; returns -1 if any errors were detected

See Also: e1562_fopen on page 294

e1562_fflush

Causes any unwritten data for the stream to be written to its associated file.

Synopsis:

```
#include "e1562lif.h"
int e1562_fflush(e1562_FILE *stream);
```

Notes: The file must be opened with `e1562_fopen` before this function can be executed.

Example:

```
e1562_FILE *fp
fp = e1562_fopen (2, "V5x:rotor", "r+");
                .
                .
        read/write data
                .
                .
e1562_fflush(fp);
```

Return Value: This function returns 0 if successful and -1 if a write error occurs.

See Also: e1562_fopen on page 294

e1562_fgetpos

Stores the current value of the file position indicator.

Synopsis:

```
#include "e1562lif.h"

e1562_errors e1562_fgetpos(e1562_FILE *stream,
                           e1562_fpos_t *byteOffset);
```

Description: This function stores the current value of the file position indicator for the stream into the object pointed to by 'byteOffset.' The value stored contains information usable by the `e1562_fsetpos` function for repositioning the stream.

Notes: The file must be opened with `e1562_fopen` before this function can be executed.

This function differs from the ANSI C fgetpos in two ways:

- The returned information is defined to be the byte offset from the beginning of the file at which the file position indicator is currently located.
- Upon failure the value of errno is NOT modified.

Example:

```
e1562_FILE *data;
e1562_fpos_t position;

data = e1562_fopen(0, "Vbc:spl_5", "w+");
e1562_fgetpos(data, &position);
```

Return Value: This function returns 0 if successful and an error number if it fails.

See Also:
e1562_fsetpos on page 296
e1562_fopen on page 294

e1562_fopen

Open the file specified by the string name and designate the file size.

Synopsis:

```
#include "e1562lif.h"
e1562_FILE *e1562_fopen(e1562ID id, const char *name,
                        const char *mode);
```

Description: The filename may include the volume name as a prefix to the file name separated by a colon, or may rely on the default volume (see e1562_defaultVolume on page 287). The argument ‘mode’ points to a string with one of the following options:

r	Open file for reading.
w	Truncate to zero length or create file for writing.
a	Append; open or create file for writing at end-of-file.
r+	Open file for update (reading and writing).
w+	Truncate to zero length or create file for update.
a+	Append; open or create file for update; writing at end-of-file.

Opening a file with ‘r’ as the first character in mode fails if the file does not exist or cannot be read. Opening a file with ‘a’ as the first character causes all writes to be forced to the end-of-file, regardless of any intervening calls to the `e1562_fsetpos` function. When a file is opened for update (‘+’ contained in the mode string), both input and output may be performed on the file. However output may not be directly followed by input unless either the function `e1562_fsetpos` or `e1562_fflush` is called, and input may not be directly followed by output unless the input encountered the end-of-file or `e1562_fsetpos` is called.

Notes: This is the only LIF library function which may include file size in the file name.

A filename may include up to 54 characters: a prefix of up to 28 characters for the volume specifier, a ten character name conforming to the LIF restriction, a suffix of up to thriteen characters to specify the file length and three additional characters to accommodate a colon, a comma and the null terminator.

Up to ten files may be open simultaneously.

Example:

```
e1562_FILE *f;
```

Open file for reading only:

```
f = e1562_fopen(1, "Vfx:piston2", "r");
```

Create file for writing:

```
f = e1562_fopen(2, "Vac:chan7,1048576", "w");
```

Open file for reading and writing:

```
f = e1562_fopen(0, "Vx2:station4", "r+");
```

Return Value: If successful this function returns a pointer to the object controlling the stream and returns a null pointer if errors are detected.

See Also: e1562_defaultVolume on page 287
e1562_mapModule on page 299

e1562_fread

Reads data from the file associated with the stream into an array.

Synopsis:

```
#include "e1562lif.h"

size_t e1562_fread(void *buff, size_t bufelSize, size_t count,
                   e1562_FILE *stream);
```

Description: Data of a size up to ‘count’ elements is read from the file associated with the stream into the array pointed to by ‘buff’, whose size is specified by ‘bufelSize’. The file position indicator for the steam is advanced by the number of bytes successfully read.

Notes: If an error occurs, the resulting value of the file position indicator is indeterminate. If a partial element is read, its value is indeterminate.

The file must be opened with `e1562_fopen` before this function can be executed.

Example:

```
unsigned char data[8192]
e1562_FILE *file;
size_t count;

file = e1562_fopen("V3x:vib", "r");
count = e1562_fread(data, sizeof(data[0]), 8192, file);
```

Return Value: This function returns the number of elements successfully read, which may be less than ‘count’ if a read error or end-of-file is encountered. If ‘count’ or ‘bufelSize’ is zero, the function returns zero and the contents of ‘buff’ and the state of the stream remain unchanged.

See Also: e1562_fopen on page 294

e1562_fsetpos

Sets the file position indicator.

Synopsis:

```
#include "e1562lif.h"

e1562_errors e1562_fsetpos(e1562_FILE *stream,
                    N       const e1562_fpos_t *byteOffset);
```

Description: This function sets the file position indicator for the stream to the value of the object pointed to by 'byteOffset.' A successful call to `e1562_fsetpos` clears the end-of-file indicator for the stream. The next operation on an update stream may be either input or output.

Notes: The file must be opened with `e1562_fopen` before this function can be executed.

If the file position is set beyond the end of file, an error will be returned, but the file position will be set to the end of file.

This function differs from the ANSI C `fsetpos` in two ways:

The data received in the object pointed to by 'byteOffset' need not have been obtained from a call to `e1562_fgetpos` but may be set by the caller as a byte offset from the beginning of the file.

Upon failure the value of errno is NOT modified.

Example:

```
e1562_FILE *fp;
e1562_fpos_t offset;

offset.positionHigh = 0;
offset.positionLow = 4096;

fp = e1562_fopen("V6x:temp", "r");
e1562_fsetpos(fp, &offset);
```

Return Value: This function returns 0 if successful and an error number if it fails.

See Also:
e1562_fgetpos on page 293
e1562_fopen on page 294

e1562_fwrite

Writes data from an array into the file associated with the stream.

Synopsis:

```
#include "e1562lif.h"

size_t e1562_fwrite(const void *buff, size_t bufelSize,
                    size_t count, e1562_FILE *stream);
```

Description: Data of a size up to ‘count’ elements is written to the file associated with the stream from the array pointed to by ‘buff’, whose size is specified by ‘bufelSize.’ The file position indicator for the steam is advanced by the number of bytes successfully written.

Notes: If an error occurs the resulting value of the file position indicator is indeterminate.

The file must be opened with `e1562_fopen` before this function can be executed.

Example:

```
#define DATA_SIZE 65536
e1562_FILE *file;
long data[DATA_SIZE];
size_t count;

file = e1562_fopen("Vxd:data", "w");
count = e1562_fwrite(data, sizeof(data[0]), DATA_SIZE, file);
if (count < DATA_SIZE)
     fprintf(stderr, "Error writing data: %ld\n", e1562_errno);
```

Return Value: This function returns the number of elements successfully written, which will be less than count only if a write error is encountered.

See Also: e1562_fopen on page 294

e1562_initializeLibrary

Initializes internal data structures which are referenced by the other functions in the VT2216A LIF library.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_initializeLibrary(void);
```

Description: This function allocates memory and initializes data structures in preparation for calling other functions in the LIF library.

Notes: This function MUST be called before using other functions in the LIF library.

Example:

```
e1562_initializeLibrary();
```

e1562_mapModule

Associates a VT2216A module with the given id.

Synopsis:

```
#include "e1562lif.h"

e1562_errors e1562_mapModule(e1562ID id, const char *interface,
                             unsigned char logicalAddr);
```

Description: Four modules may be open at a time; the valid range for ‘id’ is 0-3. The argument ‘interface’ is a string appropriate to be passed to the SICL `iopen` function. If ‘id’ already refers to a valid VT2216A, the module will no longer be accessible from the previous id upon successful completion of this function.

Notes: This function MUST be called before using id as an argument to another function since initially all ‘id’ are null.

Example:

```
e1562_mapModule(0, "vxi", 32);
```

Return Value: If ‘logical address’ does not refer to a VT2216A, an error will be returned.

e1562_pack

Removes empty space left by deleted files.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_pack(e1562ID id, const char *volname);
```

Description: This function reorganizes the file system specified by ‘volname’ such that there is no empty space caused by deleted files.

Unlike other file systems, data in a LIF file system is all sequential. This means that as files are created and deleted the largest available contiguous file space becomes smaller due to fragmentation. This function makes all the files contiguous at the beginning of the file system, possibly allowing a larger file to be created at the end of the file system.

Notes: Once this operation begins it must not be interrupted until it has completed or the file system will be corrupted.

Example:

```
e1562_pack(2, "V05256V12256");
```

Return Value: This function returns zero if successful and an error number if it fails.

See Also: e1562_mapModule on page 299

e1562_remove

Delete the specified file from the LIF directory.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_remove(e1562ID id, const char *filename);
```

Description: The space used by the file ‘filename’ is released to the file system for use by other files.

Example:

```
e1562_remove(0, "Vx9:myfile");
```

Return Value: This function returns 0 if successful. If the file is not found or the file is open an error is returned and the file is not removed.

See Also: e1562_mapModule on page 299

e1562_rename

Change the name of a file.

Synopsis:

```
#include "e1562lif.h"
e1562_errors e1562_rename(e1562ID id, const char *oldname,
                          const char *newname);
```

Description: This function changes the name of a file from ‘oldname’ to ‘newname.’ ‘Newname’ need not contain the volume in its name.

The following conditions generate an error:

- ‘oldname’ does not exist on either the default volume or on the volume specified in the name
- the volume name for ‘newname’ (if used) does not match the volume on which ‘oldname’ resides
- ‘newname’ is the same as a current file in the directory
- the file ‘oldname’ is open

Example:

```
e1562_rename(3, "Vex:myfile", "yourfile");
```

Return Value: If successful this function returns a 0 and ‘oldname’ no longer refers to an existing file, otherwise an error is generated and the original filename is not changed.

See Also:
e1562_defaultVolume on page 287
e1562_mapModule on page 299

e1562_setEOF

Sets the end-of-file marker for the file.

Synopsis:

```
#include "e1562lif.h"

e1562_errors e1562EOF(e1562_FILE *stream,
                      const e1562_fpos_t *byteOffset);
```

Description: This function is useful in establishing a logical end of file when the data is written to the file by some method other than the LIF file system (such as by using VT2216A sequences).

Notes: The EOF cannot be set beyond the size of the file specified when it was created.

Example:

```
e1562_FILE *f;
e1562_fpos_t offset;

f = e1562_fopen("Vax:auto_jun14", "w");
offset.positionHigh = 0;
offset.positionLow = 8388608;
e1562_setEOF(f, &offset);
```

Return Value: This function returns zero if successful and an error code if it fails.

See Also: e1562_fopen on page 294

VT2216A LIF Commands

The following six commands allow certain actions to be performed on volumes directly from the command line, without having to write and compile a C program.

The same volume and file name conventions apply as for the previous functions. See Naming Conventions on page 279.

e1562cp

Copy files.

Synopsis: `e1562cp [-Lisuv] file 1 [file2 ...] target`

Description: Copy `file1` to `target`. If `target` specifies either a LIF volume or is "." (for the current directory on the host), `file1` is copied to that directory, otherwise a file with the name `target` is created with the contents of `file1`. If more than one file is specified, `target` must be a volume name or ".". e1562cp may be used to copy files from the VT2216A to the host, from the host to the VT2216A or from the VT2216A to the VT2216A (either the same volume or a different volume).

Each file and/or target must be prefixed with a LIF volume specifier to indicate files on the VT2216A. A target consisting of only a volume name must include the ":" at the end of the name.

Option	Description
-L	Specifies the logical address of the VT2216A. Default address is 144.
-i	Specifies the interface which connects to the VXI cardcage containing the VT2216A. The default is "vxi."
-s	Specifies the size of the block used to copy files between the host and the VT2216A. The default is 8192.
-u	Specifies that usage information should be printed then exit.
-v	Specifies that the verbose mode should be enabled.

Example: Copy a file from the host to a VT2216A volume:

```
e1562cp jan20note Vx0:jan20note
```

or

```
e1562cp jan20note Vx0:
```

Copy several files from a VT2216A to the host:

```
e1562cp -L96 -s524288 V24:engNotes V24:engVib V24:engTemp .
```

Copy a file between VT2216A volumes:

```
e1562cp Vx0:SPLmar12 Vx4:SPLmax
```

e1562in

Create a LIF file system on the specified volume.

Synopsis: `e1562in [-Liuv] volume`

Description: Initializes a LIF file system on the specified volume.

Caution — This will destroy the contents of the disk.

Option	Description
-L	Specifies the logical address of the VT2216A. Default address is 144.
-i	Specifies the interface which connects to the VXI cardcage containing the VT2216A. The default is "vxi."
-u	Specifies that usage information should be printed then exit.
-v	Specifies that the verbose mode should be enabled.

Example:

```
e1562in V00
```

e1562ls

List contents of a LIF volume.

Synopsis: `e1562ls [-Liluv] volume`

Description: This command lists the files on the specified LIF volume to STDOUT. One file per line is printed to STDOUT. The volume name must not contain the ":" which normally separates a file name from the volume name.

Option	Description
-L	Specifies the logical address of the VT2216A. Default address is 144.
-i	Specifies the interface which connects to the VXI cardcage containing the VT2216A. The default is "vxi."
-l	Specifies the long format of directory listing. The default is to list just the names of the files.
-u	Specifies that usage information should be printed then exit.
-v	Specifies that the verbose mode should be enabled.

Example:

```
e1562ls V00
```

e1562mv

Rename a file on a LIF volume.

Synopsis: `e1562mv [-LVivu] file newname`

Description: This command renames an existing file. `file` must exist and the `newname` must not. If `newname` contains a volume name prefix, it must be the same as that of `file.`

Option	Description
-L	Specifies the logical address of the VT2216A. Default address is 144.
-V	Specifies a default volume so that several files may be specified without including the volume in each one.
-i	Specifies the interface which connects to the VXI cardcage containing the VT2216A. The default is "vxi."
-u	Specifies that usage information should be printed then exit.
-v	Specifies that the verbose mode should be enabled.

Example:

```
e1562mv Va4:spl43 spl43.old
```

e1562pk

Coalesce files on a LIF volume.

Synopsis: `e1562pk [-Liuv] volume`

Description: This command coalesces files on the specified LIF volume by packing together files in the directory and on the volume into contiguous space at the beginning of the volume. This allows a larger file to be created later if there were several deleted files or if a small file has been used to fill the spot originally used for a large file.

Caution Once this command starts working it must not be interrupted or the file system will be corrupted.

Option	Description
-L	Specifies the logical address of the VT2216A. Default address is 144.
-i	Specifies the interface which connects to the VXI cardcage containing the VT2216A. The default is "vxi."
-u	Specifies that usage information should be printed then exit.
-v	Specifies that the verbose mode should be enabled.

Example: `e1562pk Vx3`

e1562rm

Remove one or more files from a LIF volume or volumes.

Synopsis: `e1562rm [-LViuv] file1 [file2 ...]`

Description: This command deletes each of the files specified. Usually, each file will be specified with the volume prefix as ‘volume:file’. If the -V option is used to specify the volume name, all files which do not contain a volume will use the specified default volume.

Option	Description
-L	Specifies the logical address of the VT2216A. Default address is 144.
-V	Specifies a default volume so that several files may be specified without including the volume in each one.
-i	Specifies the interface which connects to the VXI cardcage containing the VT2216A. The default is "vxi."
-u	Specifies that usage information should be printed then exit.
-v	Specifies that the verbose mode should be enabled.

Example:

```
Remove a single file:
e1562rm -L32 -ivxi2 Vx2:temp.old

Remove multiple files, most from V8x, one from Vxc:
e1562rm -VV8x abc def ghi Vxc:xyz mno
```

LIF Library Errors

Error Number	Name	Description
0	e1562Err_noError	No error was generated
1	e1562Err_noSessionAvailable	A Session is not available
2	e1562Err_invalidVolumeName	The specified volume name is not valid
3	e1562Err_missingVolumeName	No volume name was specified
4	e1562Err_VolumeOpen	The specified volume is already open
5	e1562Err_interfaceError	An error was detected on the interface
6	e1562Err_outOfMemory	Memory space is insufficient for the designated function
7	e1562Err_systemError	A system error occurred
8	e1562Err_idInvalid	The specified id is invalid
9	e1562Err_fileSizeInvalid	The specified file size is invalid
10	e1562Err_fileNameInvalid	The specified file name is invalid
11	e1562Err_fileModeInvalid	The specified file mode is invalid
12	e1562Err_fileDoesNotExist	The specified file does not exist
13	e1562Err_fileDoesNotExistNoSize	The specified file has no size designation
14	e1562Err_fileExistsWithSize	The specified file already has a specified file size; remove size designation
15	e1562Err_fileEOF	The end of the file was encountered before the transfer was completed
16	e1562Err_fileTypeBad	The file type specified is not valid
17	e1562Err_endOfDirectory	The end of the directory was encountered before the transfer was completed
18	e1562Err_volumeNotLIF	The specified volume is not of a LIF type
19	e1562Err_renameVolumeDifferent	The file to be renamed is specified with two different volume designations
20	e1562Err_seekPastEOF	The read/write location designated is past the end of file marker
21	e1562Err_setEOFpastSize	The EOF specified is beyond the created size of the file
22	e1562Err_fileOpen	A file which is open has been designated to delete, rename or copy

Glossary

A16

16-bit address space. A16 has an upper limit of 6,5535.

A24

24-bit address space. A24 has an upper limit of 16,777,215.

A32

32-bit address space. A32 has an upper limit of 4,294,967,295.

ADC

an Analog-to-Digital Converter module used as the input to a VXI system. Examples include the VT1413C and Agilent/HP E1432A.

address space

a range of addresses in memory. See also A16, A24, A32, and Shared RAM.

bit bucket

a place to put unwanted data.

blocksize (Local Bus)

the amount of data, in bytes, moving in a block on the Local Bus.

blocksize (SCSI)

the size of a block of data on a disk or DAT.

cache

a block of RAM used to allow fast transfers to a slow device.

CVT

Current Value Table.

D16

a single 16-bit transfer over the VXI system bus.

D32

a single 32-bit transfer over the VXI system bus.

differential-wide

a SCSI connector in which the signal is difference between high and low wires. Wide refers to a 16-bit connection (narrow is 8 bits).

DMA

Direct Memory Access.

embedded computer

a computer (functioning as controller) which is installed in the VXI mainframe. An example is the V743.

GPIB

General Purpose Interface Bus.

implied mnemonic

keywords in a SCPI command which can be deleted without changing the effect of the command. Implied mnemonics are identified by brackets [ ] in SCPI syntax diagrams.

internal device address

a SCSI or DAT address.

LBUS

see Local Bus.

LIF libraries

Logical Interchange Format, a directory and file format used to exchange files among various Hewlett-Packard computer systems and instruments. VT2216A Sessions may be accessed by using LIF functions.

Local Bus

a daisy-chain bus structure connecting the modules in a VXI system.

logical address

the VXI address of a module.

memory space

see address space.

monitoring

A method of transferring data which allows the host computer to access part of the data during transfer operations. This is done by transferring part of the data to host memory at the same time as to the VT2216A Session.

MXI

an interface to extend the VXI bus to the memory space of a host computer.

primary address

one of three parts of VT2216A address in a SCPI environment. The primary address, typically 09, indicates which GPIB port in the system controller is used to communicate with the Slot 0 Control Module, for example the Agilent/HP E1406A.

SCPI

Standard Commands for Programmable Instruments, a standard instrument command language.

SCSI

Small Computer System Interface.

secondary address

one of three parts of VT2216A address in a SCPI environment. The secondary address indicates the device-specific address. In this case, the VXI logical address.

select code

one of three parts of VT2216A address in a SCPI environment. The select code specifies the interface. Seven is a typical number for the GPIB interface.

Sequence

specifies the order of operations for a throughput or playback Session.

Session

provides the ability to combine one or more Transfer Units together into one logical data repository.

shared memory

see Shared RAM.

Shared RAM

Memory space that is available to be shared with other devices, as a way of passing data. Shared RAM has an upper limit of 262143. (RAM = Random Access Memory).

single-ended

a SCSI connector in which one wire is ground and the other wire is the signal.

split session

data from one Session split across two SCSI devices.

SRAM

Static RAM.

SRQ

Service Request.

static+dynamic

a measurement that combines low sample-rate data from static sensors (such as temperature or pressure) with dynamic data (such as vibration or acoustics).

striping

Sessions using multiple Transfer Units containing data which has N blocks on Transfer Unit 1, M blocks on Transfer Unit 2 and so on.

system bus

a way of referring to the VXI bus not including the Local Bus.

Transfer Unit

a quantity of data transferred as a unit. A transfer unit can refer to data from either one or two devices. Also called a TUNIT.

TTLTRG

eight lines on the VXI backplane which are available to provide synchronization between devices. The VT2216A uses the TTLTRG lines for simple communication with other devices.

TUNIT

see Transfer Unit

Index