NI-VXI™
Text Utilities Reference Manual

October 1993 Edition

Part Number 320321-01

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20, Canada (Ontario) (519) 622-9310,
Canada (Québec) (514) 694-8521, Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921, Netherlands 03480-33466, Norway 32-848400,
Spain (91) 640 0085, Sweden 08-730 49 70, Switzerland 056/20 51 51, U.K. 0635 523545

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

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

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

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

NI-VXI™ is a trademark of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

Warning Regarding Medical and Clinical Use of National Instruments Products

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual ix
- Organization of This Manual ix
- Conventions Used in This Manual ix
- How to Use the Documentation Set x
- Related Documentation x
- Customer Communication x

Chapter 1
Introduction 1-1
- vxiinit 1-1
- Startup Resource Manager 1-1
  - Logical Address 0 (VXI Resource Manager) Operation 1-1
  - Non-Logical Address 0 (Servant-Side) Operation 1-2
- vxitedit 1-2
- victext 1-2

Chapter 2
Startup Resource Manager 2-1
- VXI Resource Manager Operation 2-2
  - Examination of Non-VXI Devices 2-2
  - VXI Device Identification 2-2
  - Self-Test Management 2-2
  - Address Map Configuration 2-3
  - Commander/Servant Hierarchies 2-3
  - Allocation of IRQ Lines 2-3
  - Multiple Mainframe Interrupt, Trigger, and Utility Bus Extension 2-3
  - Initiating Normal Operation 2-3
- Message-Based Servant-Side Operation 2-4
  - VXI Device Identification 2-4
  - Self-Test Management 2-4
  - Address Map Configuration 2-4
  - Commander/Servant Hierarchies 2-4
  - Allocation of IRQ Lines 2-4
  - Initiating Normal Operation 2-4
- Non-Message-Based Servant-Side Operation 2-5
  - VXI Device Identification 2-5
  - Self-Test Management 2-5
  - Address Map Configuration 2-5
  - Commander/Servant Hierarchies 2-5
  - Allocation of IRQ Lines 2-5
  - Initiating Normal Operation 2-5
- Errors 2-5
- Running the Startup RM 2-6
- File Access 2-6

Chapter 3
VXI Text Resource Editor 3-1
- Introduction to vxitedit 3-1
- Using vxitedit 3-2
- File Access 3-4
- Running vxitedit 3-5

Resource Manager Display 3-5
Configuration Editors 3-5
Manufacturer Name Editor 3-6
Model Name Editor 3-7
Device Name Editor 3-8
Non-VXI Device Editor 3-9
Interrupt Configuration Editor 3-11
Trigger Configuration Editor 3-13
Utility Bus Configuration Editor 3-14

Chapter 4
VXIbus Text Interactive Control Program 4-1

Running victext 4-1
victext Command Descriptions 4-1
System Configuration Commands 4-2
finddevla 4-2
getdevinfo 4-3
setdevinfo 4-4
createdevinfo 4-5
Commander Word Serial Protocol Commands 4-6
wsrd 4-6
wsrdf 4-7
wswrt 4-7
wswrtf 4-8
wscmd 4-8
wscmd? 4-8
wsresp 4-9
wstrg 4-9
wsclr 4-9
wsabort 4-10
wslcmd 4-10
wslresp 4-11
wsecmd 4-11
wssettmo 4-11
wsgettmo 4-12
Servant Word Serial Protocol Commands 4-13
wssenable 4-13
wssdisable 4-13
wssrd 4-13
wsswrt 4-14
wssabort 4-14
High-Level VXIbus Access Commands 4-15
vxiin 4-15
vxiout 4-16
vxiinreg 4-17
vxioutreg 4-17
vximove 4-18
Local Resource Access Commands 4-19
getmyla 4-19
vxiinlr 4-19
vxioutlr 4-20
setmodid 4-20
readmodid 4-20
VXI Signal Commands 4-21
routesignal 4-21
enablesignalint 4-22
disablesignalint 4-22
signaldeq 4-22

signalenq ... 4-22
signaljam ... 4-23
waitforsignal ... 4-23
VXI Interrupt Commands ... 4-24
routevxiint ... 4-24
enablevxitosignalint ... 4-24
disablevxitosignalint ... 4-25
enablevxiint ... 4-25
disablevxiint ... 4-26
vxiintacknowledgemode ... 4-26
assertvxiint ... 4-27
deassertvxiint ... 4-27
acknowledgevxiint ... 4-27
VXI Trigger Commands ... 4-28
srctrig ... 4-28
enabletrigsense ... 4-29
disabletrigsense ... 4-29
waitfortrig ... 4-30
acknowledgetrig ... 4-30
maptrigtotrig ... 4-31
unmaptrigtotrig ... 4-32
trigassertconfig ... 4-33
trigcntrconfig ... 4-34
trigextconfig ... 4-35
trigtickconfig ... 4-36
System Interrupt Handler Commands ... 4-37
enablesysfail ... 4-37
disablesysfail ... 4-37
enableacfail ... 4-37
disableacfail ... 4-38
enablesoftreset ... 4-38
disablesoftreset ... 4-38
assertsysreset ... 4-39
enablesysreset ... 4-39
disablesysreset ... 4-40
Bus Extender Commands ... 4-41
mapecltrig ... 4-41
mapttltrig ... 4-42
maputilbus ... 4-42
mapvxiint ... 4-43
Auxiliary Commands ... 4-44
disablemonitor ... 4-44
enablemonitor ... 4-44
set ... 4-44
pparms ... 4-45
pwidth ... 4-45
peek ... 4-46
poke ... 4-46
scripton ... 4-47
scriptoff ... 4-47
cfon ... 4-47
cfoff ... 4-47
rmentry? ... 4-48
laddrs? ... 4-48
numladdrs? ... 4-48
cmdrtable? ... 4-48
a16memmap? ... 4-48
a24memmap? ... 4-49

a32memmap? 4-49
readregister? 4-49
writeregister 4-50
devicenumber? 4-50
deviceladd? 4-50
deviceconfigure? 4-51
deviceinformation? 4-51
devicereset? 4-51
$ 4-52
history 4-52
! 4-52
* 4-53
version 4-53
system 4-53
quit 4-53

Appendix A
NI-VXI File Formats A-1

Appendix B
Customer Communication B-1

Glossary Glossary-1

Index Index-1

Figures

Figure 3-1. vxitedit Main Menu 3-5

Figure 3-2. Multimainframe Example 3-12

Tables

Table 2-1. Startup RM Command Line Options 2-6
Table 2-2. Startup RM Function Parameters 2-6

Table 3-1. vxitedit Command Summary 3-2
Table 3-2. Files used by vxitedit 3-4

About This Manual

This manual describes in detail the interactive text utilities for the NI-VXI software.

Organization of This Manual

This manual is organized as follows.

- Chapter 1, *Introduction*, contains an overview of the NI-VXI text utilities.
- Chapter 2, *Startup Resource Manager*, describes the operation and applications of the Startup Resource Manager.
- Chapter 3, *VXI Text Resource Editor*, introduces you to `vxitedit`, the text-based VXI Resource Editor program that you use to edit system and device information.
- Chapter 4, *VXIbus Text Interactive Control Program*, introduces you to `victext`, the text-based VXIbus Interactive Control program you use to communicate directly with VXI devices through commands you enter at the keyboard.
- Appendix A, *NI-VXI File Formats*, contains descriptions of the files used by the NI-VXI software.
- Appendix B, *Customer Communication*, directs you where you can find forms to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, and symbols.
- The *Index* contains an alphabetical list of key terms and topics used in this manual, including the page where each one can be found.

Conventions Used in This Manual

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

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. In this manual, italics are also used to denote Word Serial commands and queries.
`monospace`	Text in this font denotes characters that are to be literally input from the keyboard, or as sections of code. This font is also used to indicate function syntax, program names, parameter names, console responses, and syntax examples.
<`address`>	Angle brackets enclosing a term in `monospace` denote a parameter to a VXI Text Interactive Control Program (`victext`) command.

Numbers in this manual are base 10 unless noted as follows:

- Binary numbers are indicated by a -b suffix (for example, 11010101b).
- Octal numbers are indicated by an -o suffix (for example, 325o).
- Hexadecimal numbers are indicated by an -h suffix (for example, D5h).

- ASCII character and string values are indicated by double quotation marks (for example, "This is a string").

Terminology that is specific to a chapter or section is defined at its first occurrence.

How to Use the Documentation Set

We suggest that you begin by reading the Getting Started manual that came in your kit, which gives you a hands-on introduction to the software and instructs you on installation procedures. When you are familiar with the material in the Getting Started manual, you can begin to use the information contained in this manual.

You can use the `victext` utility to program instruments interactively from the computer keyboard rather than from an application program. This program helps you to become familiar with the NI-VXI function calls and also helps you to troubleshoot problems with your device and develop applications.

After you are familiar with the NI-VXI function calls, write your application program. Use `victext` whenever possible to generate the sequences of NI-VXI function calls that your application program will make. Trying your function calls in `victext` is also helpful if your application program behaves differently than you expected.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- *IEEE Standard for a Versatile Backplane Bus: VMEbus*, ANSI/IEEE Standard 1014-1987
- VXI-1, *VXIbus System Specification*, Revision 1.4, VXIbus Consortium
  (available from National Instruments, part number 350083-01)
- VXI-6, *VXIbus Mainframe Extender Specification*, Revision 1.0, VXIbus Consortium
  (available from National Instruments, part number 340258-01)

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual and your Getting Started manual contain comment and configuration forms for you to complete. These forms are in the *Customer Communication* appendix at the end of our manuals.

Chapter 1
Introduction

This chapter contains an overview of the NI-VXI text utilities.

The NI-VXI text utilities consist of the following application programs.

- VXI local hardware initialization program (`vxiinit`)
- Startup Resource Manager (RM)
- VXI Text Resource Editor (`vxitedit`)
- VXIbus Text Interactive Control Program (`victext`)

vxiinit

`vxiinit` is an application program that must be run at system startup to configure the local hardware. This program verifies that the correct hardware is installed and operational. It also performs basic initializations that the Startup RM requires for operation.

Startup Resource Manager

The `resman` program performs system-level configuration of the local CPU. The configuration of the local CPU's logical address determines which of two possible modes of operation the Startup RM will use. This configuration is performed with the `vxitedit` program. If the local CPU's logical address is 0 (VXI Resource Manager), the Resource Manager functions are performed as defined in the VXIbus specification. This is the most common (and the default) operation of the Startup RM. If, however, the local CPU's logical address is not 0, a special *Servant-side* application is automatically executed. In this mode, you can integrate multiple CPUs into a single VXI system without the need for writing additional application code. You can execute any application you wish with your non-Resource Manager CPU after the `resman` program completes.

Logical Address 0 (VXI Resource Manager) Operation

If the local CPU is configured at Logical Address 0, the Startup RM performs VXIbus-defined RM operations. Section C.4.1 of the VXIbus specification describes the VXIbus RM as a device at Logical Address 0 that performs the following functions at system startup.

- Identifies all VXIbus devices in the system
- Manages the system self-tests and diagnostic sequence
- Configures the system's A24 and A32 address maps
- Configures the system's Commander/Servant hierarchies
- Allocates the VMEbus IRQ lines
- Initiates normal system operation

The Startup RM is a superset of the VXIbus-defined RM. It not only supports all of the required functions, but also performs some functions beyond Section C.4.1 of the VXI specification. The following capabilities are supported by the Startup RM.

- Multiple mainframe support using standard VXIbus mainframe extenders (VXI document VXI-6, *VXIbus Mainframe Extender Specification*)
- Support for dynamically configured devices on a per mainframe basis (Section F of the VXIbus specification)
- Integration of non-VXI (VME and pseudo-VXI) devices on a per mainframe basis using the `vxitedit` utility

Non-Logical Address 0 (Servant-Side) Operation

If the device is not configured at Logical Address 0 and it is configured as a Message-Based device, the Startup RM performs the following functions at system startup.

- Waits for the *Begin Normal Operation* command from the VXIbus Resource Manager
- Logs devices granted to it by the system Resource Manager and configures its own Commander/Servant hierarchies
- Initiates normal system operation for its Servants

When the Startup RM is complete, all required configurations have been performed. Any application-specific program can be executed at this time.

If the device is not configured at Logical Address 0 and it is *not* configured as a Message-Based device, the Startup RM performs local CPU initialization/configuration only and exits immediately. It is at the discretion of the application to determine when it can begin communication with other devices in the VXI system (if applicable).

vxitedit

`vxitedit` is an interactive program that serves as an adjunct to the Startup RM. You can use this program to edit information that cannot be obtained dynamically from VXI devices. This information includes local CPU parameters, manufacturer names, model names, device names, non-VXI device configurations, VXI interrupt configuration, and multiple mainframe configurations. The Startup RM uses this information to configure the devices in the system. `vxitedit` incorporates several resource editors and displays using the text-based interaction.

victext

You can use the `victext` program to communicate interactively with the VXI devices in your system. `victext` has multiple purposes. You can use it to learn the NI-VXI function calls, communicate with devices, troubleshoot problems, and develop your application.

The VXIbus status (BERR, SYSFAIL, ACFAIL, VXI interrupts, TTL/ECL triggers) is monitored and displayed. Hence, `victext` also serves as a bus monitor for monitoring the state of the bus interface.

Chapter 2
Startup Resource Manager

This chapter describes the operation and applications of the Startup Resource Manager (RM).

You use the VXI Text Resource Editor (`vxitedit`) to configure VXI device characteristics of the local CPU, such as logical address, manufacturer ID, model code, and so on. When the device is configured at Logical Address 0, the Startup RM interactively configures the VXI memory maps and devices to smoothly integrate compatible VXI devices with non-VXI-compatible devices. Because the RM supports mainframe extenders, it can bring up a single or a multiple-mainframe VXI system. It can be the first program to run after the operating system is started up, or it can be run at a later time. The RM can locate and understand preconfigured information about desired memory maps and the presence of both VXI and non-VXI devices, such as VME devices. It displays information about all devices. This information includes their names, logical addresses, assigned VXI address space locations, self-test status, communication capability, status of each slot, protocols supported, VMEbus IRQ line allocation, and Commander/Servant hierarchy. The Startup RM uses strict interpretation of the VXIbus specification to perform extensive checks for errors, ambiguities, and undefined status, and it reports detailed system status information as the checks are performed. Noncritical violations/inconsistencies with the VXIbus specification are flagged as warnings. Critical violations cause the RM to abort operation. In either case, a detailed description of the configuration results can be automatically displayed for the integrator's inspection. This information can also be saved to an ASCII file.

If the device is not configured at Logical Address 0 and it is configured as a Message-Based device, the Startup RM performs *Servant-side* operations necessary to record the configuration from the VXIbus Resource Manager. After waiting for the *Begin Normal Operation* Word Serial command, it configures the Commander/Servant hierarchy for its Servants and then initiates normal operation (sends *Begin Normal Operation* to its Message-Based Servants).

If the device is not configured at Logical Address 0 and it is *not* configured as a Message-Based device, the Startup RM performs local CPU initialization/configuration only and exits immediately. It is at the discretion of the application to determine when it can begin communication with other devices in the VXI system (if applicable).

If you need to re-execute the Startup RM, be sure to take the following guidelines into account.

- *All* devices in the system must be reset either by cycling power in each mainframe or by asserting SYSRESET* on each mainframe's backplane. This is the only way to guarantee that all devices are in the VXI Configure substate.
- It is a violation of the VXIbus specification to perform RM operations more than once. Message-Based devices are not required to respond to most of the configuration commands after the *Begin Normal Operation* command has been sent (and the device is in the Normal Operation substate).

VXI Resource Manager Operation

The following paragraphs describe the overall operation of the Startup RM when the local CPU is configured at Logical Address 0 (VXI Resource Manager).

Examination of Non-VXI Devices

The Startup RM examines the `nonvxi.tbl` file to determine whether there are any non-VXI devices in the system. The information for each device found is stored in a table and used later on by the RM to configure the address spaces for the VXI devices and to configure the IRQ lines. This procedure prevents any conflicts.

VXI Device Identification

The Startup RM waits for the specified time (the VXIbus specification requires at least five seconds) before accessing any VXIbus device's A16 configuration registers. It then scans the VXI configuration space to locate and identify all the devices on the VXIbus, including any mainframe extender interfaces that may be present in the system. A mainframe extender interface uses four address windows to map in and out of its VXI mainframe. These four windows represent each of the three VXI address spaces (A32, A24, and A16) and a dedicated window for VXI configuration space (upper 16 KB of A16 space). For each window, the range that maps the extender bus into the mainframe is whatever is left over from the window that maps the VXIbus out of the mainframe to the extender bus. The RM *turns on* the address windows to *see* inside each VXIbus mainframe, where it scans logical addresses starting from the logical address of the mainframe extender through Logical Address 254 for *statically configured* (SC) devices. The Startup RM does not support SC devices at Logical Address 255. This logical address is reserved for *dynamically configured* (DC) devices only. For each SC device found, it reads the device class and manufacturer's ID code from the ID register, and the model code from the Device Type register. If the device is an extended class device, the RM reads its Subclass register. Finally, the RM performs slot associations for each SC device by reading its Status register while asserting each slot's MODID line.

The RM then looks for DC devices by asserting each MODID line and reading the device's ID register at Logical Address 255. For each DC device found, it reads the device's configuration registers, as with the SC devices, but it also assigns a logical address to each device by writing an appropriate value to the device's Logical Address register.

After the RM has read the device's configuration registers, it uses the following algorithm to assign device names to the VXI devices in the system.

1. Based on the read manufacturer ID, the RM finds out the manufacturer name for that ID from the file `mfnameid.tbl` (database for Manufacturer Name Editor). If this step fails, the RM creates a name for the device.

2. Based on the read model code and the manufacturer name found in step 1, the RM finds out the model name from the file `model.tbl` (database for Model Name Editor). If this step fails, the RM creates a name for the device.

3. Based on the model name found in step 2, the manufacturer name found in step 1, the logical address, and the physical location (mainframe/slot), the RM finds out the device name from the file `device.tbl` (database for Device Name Editor). If this step fails, the RM creates a name for the device based on the model name found in step 2.

Self-Test Management

Because the Startup RM waited for the specified time before accessing any other VXIbus device's A16 configuration registers, all devices in the VXIbus system should have completed their self-tests. If any device has not passed its self-test (has not asserted its PASSed bit in its Status register) within the specified time, the RM sets the Sysfail Inhibit and Reset bits in the device's Control register. Any device with its Reset bit set is considered offline and is

not assigned as a Servant to any device. All known information about the device, however, is logged for later inspection by an application program.

Address Map Configuration

A device must have A16 registers and can optionally have address space assigned to it in either A24 or A32 space. The RM determines the address space of each device by reading the Address Space bits in the device's ID register. If the device requests address space in A24 or A32 space, the RM allocates it to the device according to the size requirements indicated by the contents of the Required Memory field of its Device Type register. The RM determines an available base address for the device's address space and writes the appropriate value to the device's Offset register. In addition, the A24/A32 Enable bit in the device's Control register is enabled. If the device resides in a remote mainframe, the RM also *turns on* the appropriate address map windows on the mainframe extender so the whole system can *see* the memory space. Warning messages are output if the requested memory of the available devices exceeds the amount of address space available. In this configuration, some devices do not have address space assigned to them.

Commander/Servant Hierarchies

The Startup RM finds all Commanders by checking the CMDR bit in the Protocol register of each Message-Based device. It uses the Word Serial query *Read Servant Area* to read the Servant Area size from each Commander. All devices that fall within this area are assigned as Servants to the Commander. In this way, any type of static Commander/Servant hierarchy can be created. The RM uses the Word Serial command *Grant Device* to assign Message-Based Servants to Commanders. DC devices are not assigned to any Commander. If a particular application requires that a DC device be the Servant of another Commander (other than the RM), the application must send the Word Serial command *Grant Device* to the Commander, keeping in mind the restrictions made by the VXIbus specification on the Configure substate and the Normal Operation substate.

Allocation of IRQ Lines

The Startup RM allocates the VMEbus IRQ lines among the various VXI interrupt handlers and interrupters in the system. The RM is programmed to handle not only devices with jumper-selectable IRQ line assignments (static handlers and interrupters) but also VXI-defined, Message-Based programmable interrupters (PI devices) and programmable interrupt handlers (PH devices). The RM determines which devices are PI or PH devices by sending the Word Serial query *Read Protocols* to the device. You can use the Interrupt Configuration Editor in `vxitedit` to create a table that specifies how to map programmable handlers to static interrupters, static handlers to programmable interrupters, and static handlers to static interrupters. This algorithm is described in Section C.4.1.5 of the *VXIbus System Specification*.

Multiple Mainframe Interrupt, Trigger, and Utility Bus Extension

The Startup RM fully supports the transparent VXI interrupt, TTL/ECL Trigger, and Utility Bus extension mechanism specified in the VXI document VXI-6, *Mainframe Extender Specification.* Using the `vxitedit` utility, you can specify what interrupts, triggers, and utility bus signals (SYSFAIL*, ACFAIL*, and SYSRESET*) are transparently mapped between mainframes.

Initiating Normal Operation

After allocating IRQ lines, the RM sends the *Begin Normal Operation* command to all the top-level Commanders in order of increasing logical address. At this point, the RM's startup sequence is complete and it terminates operation as the system enters the Normal Operation state. You can execute any application after this point.

Message-Based Servant-Side Operation

The following paragraphs describe the overall operation of the Startup RM when the local CPU is configured as a Message-Based device at a logical address other than 0.

VXI Device Identification

A device that is not the RM needs to configure only itself. The local registers are automatically initialized according to the capabilities of the local device and the settings created in `vxitedit`.

Self-Test Management

A device that is not the RM manages only its own self-test. If the local CPU fails its own self-test, the PASSed bit in its Status register remains unasserted and the local CPU does not stop asserting SYSFAIL*.

Address Map Configuration

The `vxitedit` utility can be used to specify any A24 or A32 address space requirements. The Startup RM automatically logs the settings that the VXI RM assigned the local CPU.

Commander/Servant Hierarchies

The Startup RM automatically processes all Word Serial *Grant Device* commands received, recording information about the Servants it has been assigned.

Allocation of IRQ Lines

The Startup RM automatically processes all VXI interrupt Word Serial commands (*Read Interrupters*, *Assign Handler Line*, and so on) received, recording the assignment information. You can use the `vxitedit` utility to set the number of interrupters and handlers supported.

Initiating Normal Operation

After it receives the Word Serial query *Begin Normal Operation*, the local CPU propagates the query to all of its Message-Based Servant devices.

Non-Message-Based Servant-Side Operation

The following paragraphs describe the overall operation of the Startup RM when the local CPU is configured as a non-Message-Based device at a logical address other than 0.

VXI Device Identification

A device that is not the RM needs to configure only itself. The local registers are automatically initialized according to the capabilities of the local device and the settings created in `vxitedit`.

Self-Test Management

A device that is not the RM manages only its own self-test. If the local CPU fails its own self-test, the PASSed bit in its Status register remains unasserted and the local CPU does not stop asserting SYSFAIL*.

Address Map Configuration

You can use the `vxitedit` utility to specify any A24 or A32 address space requirements. The Startup RM automatically logs the settings that the VXI RM assigned to the local CPU.

Commander/Servant Hierarchies

Because the device is not configured as a Message-Based device, no Servants can be granted to the local CPU.

Allocation of IRQ Lines

Because the device is not configured as a Message-Based device, no programmable interrupters or programmable handlers are possible. Interrupt lines must be reserved by configuring the VXI RM (not the local CPU). Application-level function calls can then be made to use the interrupt lines.

Initiating Normal Operation

Because the device is not configured as a Message-Based device, the Startup RM does not wait for the Word Serial command *Begin Normal Operation*. Instead, the Startup RM exits immediately. It is at the discretion of the application to determine when it can begin communication with other devices in the VXI system (if applicable).

Errors

The RM performs extensive error-checking during each of its operations. The errors are categorized as *fatal* and *warning*. Fatal errors usually cause the RM to terminate prematurely. An example of a fatal error would be if two devices were configured for Slot 0 operation within a single mainframe. Warning messages represent cautionary information. The error messages in the RM are self-explanatory.

The RM can hang (crash the local CPU) under any of the following conditions.

- The VXI hardware is not properly configured (for example, VXIbus arbiter not enabled).
- The devices in the VXI chassis are not properly inserted in the backplane.
- The VXIbus backplane(s) is not properly configured.

If the RM hangs, attempt to isolate and correct the problem and then restart the system.

Running the Startup RM

Type `resman` to run the RM. The command line options (if applicable) you can use are shown in Table 2-1.

Table 2-1. Startup RM Command Line Options

Code	Description
`-o`	Output the Resource Manager information to the file `resman.out` instead of to the monitor or terminal.
`-d`	Do not send BNO to Message-Based devices in a DC system.
`-R`	Reset extenders only.
`-t`	Read device type of LA 0 (otherwise assume Message-Based device)

Under some operating systems, an execution shell is not always available. In these cases, the RM is provided in a library (object) format. The function takes a parameter mode described in Table 2-2, and returns a value of 0 if it ran successfully, or a value of -1 if an error occurred.

Table 2-2. Startup RM Function Parameters

Mode Bit	Description
0	Output the Resource Manager information to the file `resman.out` instead of to the monitor or terminal.
1	Do not send BNO to Message-Based devices in a DC system.
2	Reset extenders only.
3-31	Reserved.

File Access

The RM reads the files `model.tbl`, `mfnameid.tbl`, `intcfg.tbl`, `device.tbl`, `nonvxi.tbl`, `vxidev.cfg`, `vxila.cfg`, `vxibus.cfg`, `trigcfg.tbl`, and `utilbus.tbl` for information. You can alter the information in these files by using the `vxitedit` utility (refer to Chapter 3, *VXI Text Resource Editor*, for more details). After the RM configures the system, it writes all the information about all the known devices in the system to the file `resman.tbl`.

Chapter 3
VXI Text Resource Editor

This chapter describes the VXI Text Resource Editor (`vxitedit`) program.

Introduction to vxitedit

`vxitedit` is a collection of interactive editors you can use to configure your system hardware and maintain your databases of VXI information. Each of the editors in `vxitedit` generates a database table that is used by the Startup Resource Manager (RM) when it configures the VXI system. With this information, the RM can associate actual names and numbers with the bits encoded in the device's registers. The RM can then display easy-to-understand information about the status of the system as it is initialized and begins operation.

`vxitedit` incorporates the following resource editors and displays.

- **Resource Manager Display**

  Use this display to view the RM table. It displays all the information about all the VXI devices present in the system, such as the manufacturer name, logical address, model name, model code, manufacturer ID, mainframe, Commander's logical address, Servants' logical addresses, memory requirements, and interrupt line and handler assignments.

- **Configuration Editor**

  Use this editor to interactively configure the local VXI hardware. This includes setting the VXIbus interface and local CPU logical address parameters.

- **Manufacturer Name Editor**

  Use this editor to update the list of manufacturer names and ID numbers. The RM uses this data to assign symbolic (ASCII) names to VXI devices.

- **Model Name Editor**

  Use this editor to update the list of model names and their associated manufacturer names and model numbers. The RM uses this data to assign symbolic (ASCII) names to VXI devices.

- **Device Name Editor**

  Use this editor to assign device names to the VXI devices installed in the system. The device name is associated with a manufacturer name, model name, logical address, physical location (slot), and/or the mainframe extender. If the RM finds the device name with the proper attributes when it is assigning symbolic names, the RM assigns that device name to the VXI device with those attributes.

- **Non-VXI Device Editor**

  Use this editor to statically assign address space and interrupt lines to non-VXI devices. The RM needs information about the attributes of the installed non-VXI devices so it can avoid conflicts when it dynamically configures the VXI devices.

- **Interrupt Configuration Editor**

  Use this editor to define individual and intermainframe interrupt configurations. Individual mainframe configuration is used to match programmable handlers/interrupters to static handlers/interrupters. Intermainframe interrupt configuration is used to define which interrupts will be imported to or exported from a particular mainframe.

- **Trigger Configuration Editor**

  Use this editor to define intermainframe trigger (TTL and/or ECL) configurations. Intermainframe trigger configuration is used to define which triggers are imported to or exported from a particular mainframe.

- **Utility Bus Configuration Editor**

  Use this editor to define intermainframe configurations for the utility bus. Intermainframe utility bus configuration is used to define whether SYSRESET, SYSFAIL, and ACFAIL are imported to or exported from a particular mainframe.

Using vxitedit

The editors in `vxitedit` are text-based editors that accept commands entered from the keyboard. You can obtain help on `vxitedit` by selecting the `help` option from the top-level menu. For help on individual editors, type *`help`* in the corresponding editor. Table 3-1 shows the commands supported by `vxitedit`.

Table 3-1. vxitedit Command Summary

Command		Description
`list`	`[<index#>[, <index#>]]`	Lists all the entries in the specified range
`display`	`[<index#>[, <index#>]]`	Displays the entries in more detail
`add`		Adds an entry
`duplicate`	`<index#>`	Duplicates an existing entry
`modify`	`<index#>[, <field#>]`	Modifies an existing entry
`delete`	`<index#>`	Deletes an entry
`show`	`<field#>`	Shows information about the field
`save`		Saves the information
`abort`		Exits the editor, ignoring the changes made
`exit`		Exits the editor
`help`		Displays help file for the editor

You can type in the full command name or just a partial name such as *di* for *display* or *mod* for *modify*. Notice that in Table 3-1 some parameters are enclosed in brackets. These are optional parameters. In some cases, there is only one field that could possibly be modified, so you can modify the value without specifying a second number. In addition, sometimes you will have so many entries in your database that you will need to specify a range of entries shown in a `list` or `display` command. For example, to list only the first 20 entries, type *`list 0, 19`*.

For some editors not all of the commands apply. Type *help* to obtain online assistance within any editor, including determining which commands are applicable to the editor you are using. For example, the `add`, `duplicate`, and `delete` commands are not applicable in the Configuration Editors because you cannot alter the actual number of entries. However, you can change the information in the fields by using the `modify` command. For example, to change the value of a specific field, you might type *`modify 10, 2`*. The `10` denotes the index number of the entry and the `2` represents the field containing the value. (The index is the number to the left of each row of data and the field is the number at the top of each column of data.) A prompt requests the new value for the field.

The `list` and `display` commands can take several options. They can display entries in a specified range (for example, `list 1, 5`). They can display a single entry (for example, `list 5`). Or they can display all entries by merely typing *`list`* or *`display`*. The difference between `list` and `display` is that `list` shows only the most essential information for each entry, while `display` shows all the information for each entry. Use these commands to determine the contents of the database when you first enter one of the editors. You should also use them to verify that you entered all the information correctly in the database before you save the new or modified entry.

The `show` command displays information regarding a specified field. For example, if you type *`show`* to get information about the manufacturer name field under the Manufacturer Name Editor, you will learn that you can enter a maximum of 12 alphanumeric characters. In certain editors, it displays the value of the field for all entries.

Type *`save`* to save the database with the changes you have made, and then type *`exit`*. If you want to exit without saving your changes, type *`abort`*. When you return to the main menu, you can enter another editor by selecting the number for that editor.

Note: `vxitedit` displays all numbers in hexadecimal notation, with the exception of the index and field numbers, which are in decimal format. While you are not restricted to entering numerical values into a field in hex, you must be sure to prefix a hex number with *`0x`* when you do so. Otherwise, `vxitedit` will treat the number as a decimal number and will translate it into hex. For example, to enter the hex number of `0x101` you can enter the decimal equivalent *`257`* and it will be correctly translated. But if you entered *`101`* it would be translated to the wrong value.

File Access

`vxitedit` reads files `vxila.cfg`, `vxidev.cfg`, `vxibus.cfg`, `model.tbl`, `mfnameid.tbl`, `device.tbl`, `fordev.tbl`, `intcfg.tbl`, `trigcfg.tbl`, `utilbus.tbl`, and `resman.tbl` for information used in the corresponding editors, and updates the information in those files as necessary. It also uses several help files to assist you with the corresponding editors.

Table 3-2 shows the files accessed by each editor.

Table 3-2. Files used by vxitedit

Editor	Files Accessed
Resource Manager Display	`resman.tbl`

Configuration Editor	`vxila.cfg`
	`vxibus.cfg`
	`vxidev.cfg`

Manufacturer Name Editor	`mfnameid.tbl`
Model Name Editor	`model.tbl` `mfnameid.tbl`

Device Name Editor	`device.tbl`
	`model.tbl`
	`mfnameid.tbl`

Non-VXI Device Editor	`nonvxi.tbl`
Interrupt Configuration Editor	`intcfg.tbl`
Trigger Configuration Editor	`trigcfg.tbl`
Utility Bus Configuration Editor	`utilbus.tbl`

Running vxitedit

Type *vxitedit* to run the vxitedit program. The main menu appears with the *vxitedit* prompt and displays the options in vxitedit. Figure 3-1 shows the main menu of vxitedit.

```
                    VXI Text Resource Editor
                    (c) National Instruments

                  (1) resource manager display
                  (2) configuration editor
                  (3) manufacturer name editor
                  (4) model name editor
                  (5) device name editor
                  (6) non-VXI device editor
                  (7) interrupt conf editor
                  (8) trigger conf editor
                  (9) utility bus conf editor
                 (10) help
                 (11) exit

vxitedit (command or option number) >>
```

Figure 3-1. vxitedit Main Menu

To select an editor, enter the option number or type in the editor name (partial or complete). The prompt reflects the name of the editor you are currently in. Type *exit* to return to the main menu.

Resource Manager Display

Use this option to view the RM table. It displays all the information about the installed devices, including the manufacturer name, logical address, model name, model code, manufacturer ID, mainframe, Commander's logical address, Servants' logical addresses, memory requirements, and interrupt line and handler assignments.

Type *list* to examine all the entries. Type *display* for a particular entry to examine all the fields of the entry. Type *exit* to return to the previous screen. Refer to Appendix A, *NI-VXI File Formats*, for detailed information on the defined format of the resman.tbl file.

Configuration Editors

Use these editors to change the VXI hardware interface configuration. For more information on these editors, refer to the section on software installation and configuration in the Getting Started manual you received with your VXI hardware.

Manufacturer Name Editor

Use this editor to update the current list of known manufacturer names (for the associated VXI Manufacturer ID number). The RM uses this information to assign a manufacturer name to each device. The System Configuration functions of NI-VXI can be used to programmatically access this information.

When you have entered the Manufacturer Name Editor you can type *help* to learn more about how to use the editor.

Type *list* to examine all the entries. To edit any existing entry, type *modify* and give the index of the entry and the field to modify. (The index is the number to the left of each row of data and the field is the number at the top of each column of data.) You will be prompted for the new field value. After you modify the entry, `vxitedit` sorts the list in alphabetical order. Be aware that the sorting can affect the index numbers for the entries.

An easy way to create a new entry is to first duplicate a similar entry by typing *duplicate* and giving the index of the existing entry. After you duplicate an entry, modify its fields according to your requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *delete* and giving the index of the entry. You can save the information you have modified by typing *save*, or type *abort* to ignore the modifications. Type *exit* to return to the previous screen.

As an example, use the following steps to add a new entry with the manufacturer name = “Nat’l Insts” and the manufacturer ID = 0xff6.

1. Type *add* at the Manufacturer Name Editor prompt and press <RETURN>.
2. At the prompt for a manufacturer name, enter *Nat’l Insts* and press <RETURN>.
   (Maximum length for this field is 12 characters.)
3. At the prompt for a manufacturer ID, enter *0xff6* and press <RETURN>.
   (The manufacturer ID number must be within the range of 0x0 through 0xfff.)
4. Type *list* to verify that the entry you have added is entered correctly in the database.
5. Type *save* to save the modifications.
6. Type *exit* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the `mfnameid.tbl` file.

Model Name Editor

Use this editor to update the current list of known model names, along with their associated manufacturer names and model numbers. The RM uses this information to assign a model name to each device. The System Configuration functions of NI-VXI can be used to programmatically access this information.

When you have entered the Model Name Editor you can type *help* to learn more about how to use the editor.

Type *list* to examine all the entries. To edit any existing entry, type *modify* and give the index of the entry and the field to modify. You will be prompted for the new field value. For the manufacturer names, the Model Name Editor accepts only those names that are displayed by the Manufacturer Name Editor. After you modify the entry, `vxitedit` sorts the list in alphabetical order.

An easy way to create a new entry is to first duplicate a similar entry by typing *duplicate* and giving the index of the existing entry. After you duplicate an entry, modify its fields according to your requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *delete* and giving the index of the entry. You can save the information you have modified by typing *save*, or type *abort* to ignore the modifications. Type *exit* to return to the previous screen.

As an example, use the following steps to add a new entry with the model name = “AT-MXI,” the manufacturer name = “Nat’l Insts,” and the model code = 0xff3.

1. Type *add* at the Model Name Editor prompt and press <RETURN>.
2. At the prompt for a model name, enter *AT-MXI* and press <RETURN>.
   (Maximum length for this field is 12 characters.)
3. At the prompt for a manufacturer name, enter *Nat’l Insts* and press <RETURN>.

   Note: This entry must exactly match the manufacturer name entered through the Manufacturer Name Editor.
4. At the prompt for a model code, enter *0xff3* and press <RETURN>.
5. Type *list* to verify that the entry you have added is entered correctly in the database.
6. Type *save* to save the modifications.
7. Type *exit* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the `model.tbl` file.

Device Name Editor

Use this editor to update the current list of known device names installed in the system. The RM uses this information to assign a device name to each device. The System Configuration functions of NI-VXI can be used to programmatically access this information.

When you have entered the Device Name Editor you can type *help* to learn more about how to use the editor.

Type *list* to examine all the entries. To edit any existing entry, type *modify* and give the index of the entry and the field to modify. You will be prompted for the new field value. For the model and manufacturer names, the Device Name Editor accepts only those pairs that are displayed by the Model Name Editor. After you modify the entry, `vxitedit` sorts the list in alphabetical order.

An easy way to create a new entry is to first duplicate a similar entry by typing *duplicate* and giving the index of the existing entry. After you duplicate an entry, modify its fields according to your requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *delete* and giving the index of the entry. You can save the information you have modified by typing *save*, or type *abort* to ignore the modifications. Type *exit* to return to the previous screen.

As an example, use the following steps to add a new device entry with the device name = "AT-MXI0," model name = "AT-MXI," manufacturer name = "Nat'l Insts," logical address = 0, and frame and slot = 0xffff (not applicable).

1. Type *add* at the Device Name Editor prompt and press <RETURN>.
2. At the prompt for a device name, enter *AT-MXI0* and press <RETURN>.
   (Maximum length for this field is 12 characters.)
3. At the prompt for a model name, enter *AT-MXI* and press <RETURN>.

   Note: This entry must exactly match the model name entered through the Model Name Editor.
4. At the prompt for a logical address, enter *0* and press <RETURN>.
5. At the prompt for a frame, enter *0xffff* and press <RETURN>.
6. At the prompt for a slot, enter *0xffff* and press <RETURN>.
7. Type *list* to verify that the entry you have added is entered correctly in the database.
8. Type *save* to save the modifications.
9. Type *exit* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the `device.tbl` file.

Non-VXI Device Editor

Use this editor to provide non-VXI device information to the NI-VXI software. You need to assign non-VXI devices *pseudo logical addresses* (between 256 and 511 inclusive), as well as device names, manufacturer names, and model names. Using this information, the RM can integrate these devices with VXI devices. The manufacturer and model names will not normally correspond to VXI manufacturers and models; this information, however, is used by the NI-VXI software as needed to associate symbolic manufacturer and model names with the device. You can also use this editor to enter the memory and interrupt requirements for the non-VXI devices. If you choose, you can enter configuration information for numerous non-VXI devices and use the `InSystem` parameter to indicate only those devices actually installed in your system. In this way you can maintain a *database* of non-VXI devices. Keep in mind that the RM uses only the configuration information for those devices that you indicated are in the system by typing a *1* in the `InSystem` parameter. When the RM executes, it displays information regarding the non-VXI devices by referencing their assigned pseudo-logical addresses.

When you have entered the Non-VXI Device Editor you can type *help* to learn more about how to use the editor.

Type *list* to examine all the entries. Type *display* for a particular entry to examine all the fields of the particular entry. To edit any existing entry, type *modify* and give the index of the entry and the field to modify. You will be prompted for the new field value. After you modify the entry, `vxitedit` sorts the list in alphabetical order. Make sure to set (1) or clear (0) the `InSystem` field of an entry, depending on whether that device is actually present in your system.

An easy way to create a new entry is to first duplicate a similar entry by typing *duplicate* and giving the index of the existing entry. After you duplicate an entry, modify its pseudo-logical address and the other address and interrupt requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *delete* and giving the index of the entry. You can save the information you have modified by typing *save*, or type *abort* to ignore the modifications. Type *exit* to return to the previous screen.

As an example, assume you want to add a VME memory device in your system that requires 64 kilobytes of memory in A24 space and you want to configure it at 0x200000 in the A24 space. For this example we will duplicate an existing entry and rename it `NONVXIDEV1` rather than add a new one.

1. Type *list* at the Non-VXI Device Name Editor prompt and press <RETURN> to examine the current entries in the database. Find an entry you want to duplicate and take note of the index number. Remember that you can also use the `display` command to get more information about the entries so that you can find one containing the most similarities to the device you want to add to the system. In this manner, you will have fewer fields to modify.

2. Type *duplicate*, the index of the entry to be duplicated, and press <RETURN>.

3. Type *list* at the Non-VXI Device Name Editor prompt and press <RETURN> to determine the new index number.

4. Type *modify*, the index number of the new entry, and the field number of the pseudo-logical address field at the Non-VXI Device Name Editor prompt and press <RETURN>. Type in a new pseudo-logical address within the range of 256 and 511.

   Note: You can enter the number in either decimal or hex form, but `vxitedit` will display the number in hex form. For example, you can type in *257* and it will be recorded as `0x101`. You cannot, however, enter *101* without the *0x*, because `vxitedit` will see that number as outside the range of 256 to 511.

5. Type *modify*, the index number, and the field number of the device name field and press <RETURN>. Enter *NONVXIDEV1* as the name.

6. Type *modify*, the index number, and the field number of the frame field and press <RETURN>. Set the frame field to the logical address of the extended controller (in an external CPU case) or the embedded controller (in an embedded CPU case) plugged into the same mainframe as the VME device.
7. Modify the A24membase field to *0x200000*.
8. Modify the A24memsize field to *0x10000* (64 kilobytes).
9. Modify the A16membase, A16memsize, A32membase, and A32memsize fields to *0* (if they are not already set to 0).
10. Modify both the IrqLevel and IrqHandle (interrupt requirements) fields to *0*.
11. Modify the InSystem field to *1*, indicating that this non-VXI device is actually present in the system.
12. Type *list* to find the index number of the entry and then type *display*, the index number of the entry, and press <RETURN> to verify that you have correctly modified all the information.
13. Type *save* to save the modifications.
14. Type *exit* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the nonvxi.tbl file.

Interrupt Configuration Editor

Use this editor to configure individual and intermainframe interrupts. You can use this editor along with a mainframe extender to route interrupts generated in a given mainframe to other mainframes, and route interrupts generated in other mainframes into its frame. You can use the intermainframe interrupt mapping table to define whether a particular interrupt is routed in or out of the mainframe. You can use the individual interrupt mapping table to allocate interrupt lines to the static interrupters. The RM skips these interrupt lines when it dynamically assigns the interrupt lines to programmable handlers. Programmable interrupters are assigned to their Commander's levels regardless of whether the handlers are programmable or static.

If you have a multiple-mainframe configuration, use intermainframe interrupt configuration to define which interrupts are routed in or out of a particular mainframe. The `levels` field is a bit vector of VXI interrupt levels where Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively (1 = Enable for appropriate level, 0 = Disable for appropriate level). The `directions` field is a bit vector of VXI interrupt levels where Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively (1 = Into the mainframe, 0 = Out of the mainframe).

Use the next seven fields to configure the internal (individual) interrupt mapping for each mainframe. These seven fields list the logical addresses to handle the seven VXI interrupt levels. The value of 0xff (255) indicates that the corresponding VXI interrupt level is free to be assigned by the RM. If you have a static interrupter (VXI or non-VXI) device in the system, you need to enter the logical address of its handler device in the appropriate interrupt level field. The RM can then assign the corresponding interrupt level to that handler device.

When you have entered the Interrupt Configuration Editor you can type *`help`* to learn more about how to use the editor.

Type *`list`* to examine all the entries. To edit any existing entry, type *`modify`* and give the index of the entry and the field to modify. You will be prompted for the new field value.

An easy way to create a new entry is to first duplicate a similar entry by typing *`duplicate`* and giving the index of the existing entry. After you duplicate an entry, modify its fields according to your requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *`delete`* and giving the index of the entry. You can save the information you have modified by typing *`save`*, or type *`abort`* to ignore the modifications. Type *`exit`* to return to the previous screen.

As a single-mainframe example, assume you have a static interrupter device in your system that interrupts on interrupt level 4 and you want want the RM to be the handler of that device. For this example we will modify an existing entry.

1. Type *`list`* at the Interrupt Configuration Editor prompt and press <RETURN> to examine the current entries in the database. Find the entry you want to modify and take note of the index number.
2. Type *`modify`*, the index number of the entry, and the field number of the Frame field and press <RETURN>. This field designates the logical address of the extended controller (in an external CPU case) or the embedded controller (in an embedded CPU case) plugged into the same mainframe as the static interrupter device.
3. Type *`modify`*, the index number, and the field number of the `levels` field and press <RETURN>. Enter *`0`* for this field because this is a single-mainframe system.
4. Type *`modify`*, the index number, and the field number of the `directions` field and press <RETURN>. Enter *`0`* for this field because this is a single-mainframe system.
5. Modify the `IntLevel4` field to the logical address of the RM, which is 0.
6. Modify all the other interrupt level fields to *`0xff`* (if they are not already set to `0xff`).
7. Type *`display`* and the index number of the entry to verify that you have correctly modified all the information.

8. Type *save* to save the modifications.
9. Type *exit* to return to the previous screen.

As a multiple-mainframe example, assume you have a static interrupter device interrupting on interrupt level 3 in your system in one mainframe and you want want the device in another mainframe to be the handler for the interrupter, as shown in Figure 3-2. For this example we will add the following entries.

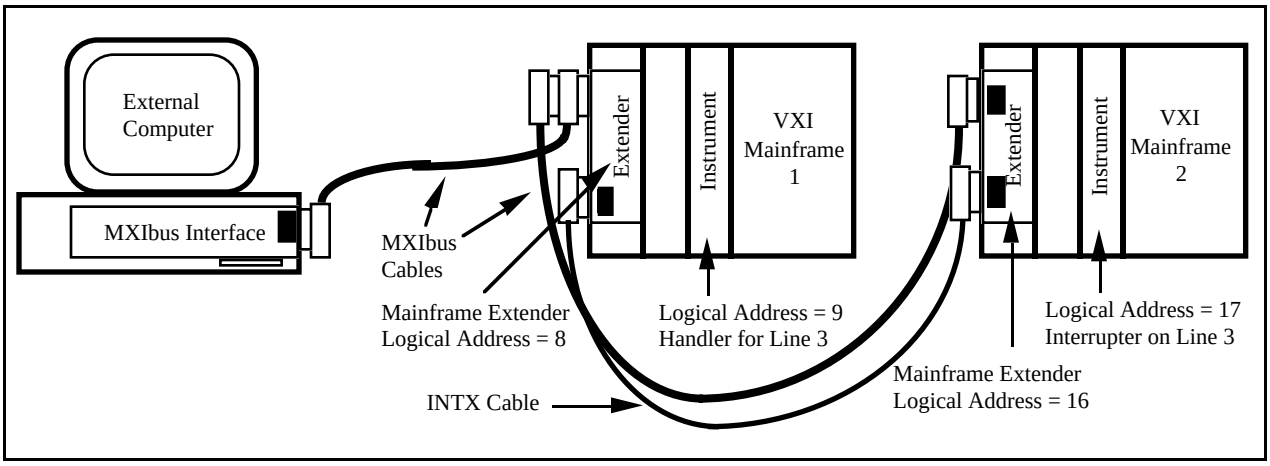

Figure 3-2. Multimainframe Example

1. Type *add* at the Interrupt Configuration Editor prompt and press <RETURN>.
2. At the prompt for `frame` enter *8* (logical address of the mainframe extender in the first frame) and press <RETURN>.
3. At the prompt for `levels` enter *0x4* (bit 2 set for level 3) and press <RETURN>.
4. At the prompt for `directions` enter *0x4* (bit 2 set for level 3 to be mapped into the mainframe) and press <RETURN>.
5. Set all the `IntLevels` to *0xff*.
6. Type *list* to verify that the entry you have added is entered correctly in the database.
7. Type *save* to save the modifications.
8. Type *add* at the Interrupt Configuration Editor prompt and press <RETURN>.
9. At the prompt for `frame` enter *16* (Logical address of the mainframe extender in the second frame) and press <RETURN>.
10. At the prompt for `levels` enter *0x4* (bit 2 set for level 3) and press <RETURN>.
11. At the prompt for `directions` enter *0x0* (bit 2 cleared for level 3 to be mapped out of the mainframe) and press <RETURN>.
12. At the prompt for `IntLevel3` enter *9* (Logical address of the handler device).
13. Set all the other `IntLevels` to *0xff*.
14. Type *list* to verify that the entry you have added is entered correctly in the database.
15. Type *save* to save the modifications.
16. Type *exit* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the `intcfg.tbl` file.

Trigger Configuration Editor

Use this editor to configure intermainframe TTL and/or ECL triggers. You can route triggers sourced in the mainframe extender to other mainframes, or bring triggers sourced in other mainframes into its frame. Use the external TTL trigger mapping table to define whether a particular trigger is imported or exported out of the mainframe.

Note: This editor is not applicable for a single-mainframe system.

The `TTLlines` field is a bit vector of TTL trigger lines where Bits 7 to 0 correspond to TTL trigger lines 7 to 0, respectively (1 = Enable for appropriate line, 0 = Disable for appropriate line). The `directions` field is a bit vector of TTL trigger lines where Bits 7 to 0 correspond to TTL trigger lines 7 to 0, respectively (1 = Into the mainframe, 0 = Out of the mainframe).

The `ECLlines` field is a bit vector of ECL trigger lines where Bits 5 to 0 correspond to ECL trigger lines 5 to 0, respectively (1 = Enable for appropriate line, 0 = Disable for appropriate line). The `directions` field is a bit vector of ECL trigger lines where Bits 5 to 0 correspond to ECL trigger lines 5 to 0, respectively (1 = Into the mainframe, 0 = Out of the mainframe).

When you have entered the Trigger Configuration Editor you can type *`help`* to learn more about how to use the editor.

Type *`list`* to examine all the entries. To edit any existing entry, type *`modify`* and give the index of the entry and the field to modify. You will be prompted for the new field value.

An easy way to create a new entry is to first duplicate a similar entry by typing *`duplicate`* and giving the index of the existing entry. After you duplicate an entry, modify its fields according to your requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *`delete`* and giving the index of the entry. You can save the information you have modified by typing *`save`*, or type *`abort`* to ignore the modifications. Type *`exit`* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the `trigcfg.tbl` file.

Utility Bus Configuration Editor

Use this editor to configure the intermainframe Utility Bus. You can route SYSRESET, SYSFAIL and ACFAIL from their source in the mainframe extender to other mainframes, or bring them from their sources in other mainframes into its frame. Use the mapping table to define whether SYSRESET, SYSFAIL and ACFAIL are routed in or out of the mainframe (or both).

Note: This editor is not applicable for a single-mainframe system.

The `modes` field is a bit vector of utility bus signals (1 = Enable for corresponding signal and direction, 0 = Disable for corresponding signal and direction) where the individual bits are defined as follows.

- Bit 0 is for SYSRESET out of the mainframe
- Bit 1 is for SYSRESET into the mainframe
- Bit 2 is for SYSFAIL out of the mainframe
- Bit 3 is for SYSFAIL into the mainframe
- Bit 4 is for ACFAIL out of the mainframe
- Bit 5 is for ACFAIL into the mainframe

When you have entered the Utility Bus Configuration Editor you can type *`help`* to learn more about how to use the editor.

Type *`list`* to examine all the entries. To edit any existing entry, type *`modify`* and give the index of the entry and the field to modify. You will be prompted for the new field value.

An easy way to create a new entry is to first duplicate a similar entry by typing *`duplicate`* and giving the index of the existing entry. After you duplicate an entry, modify its fields according to your requirements. Optionally, you can use the `add` command and enter values for all the fields for a new entry. After you add or modify the entry, `vxitedit` sorts the list in alphabetical order.

You can delete any entry in the list by typing *`delete`* and giving the index of the entry. You can save the information you have modified by typing *`save`*, or type *`abort`* to ignore the modifications. Type *`exit`* to return to the previous screen.

Refer to Appendix A for detailed information on the defined format of the `utilbus.tbl` file.

Chapter 4
VXIbus Text Interactive Control Program

This chapter introduces you to the VXIbus Text Interactive Control (`victext`) program, which you can use to communicate with VXI devices through commands you enter at the keyboard. This feature helps you learn how to communicate with devices, troubleshoot problems, and develop your application.

The `victext` command set includes the same capability of NI-VXI function calls in addition to auxiliary commands that are unique to `victext`. You can use this utility to send data and Word Serial commands to devices from the keyboard and to display data received from devices on the screen. After each function executes, `victext` displays the device's response and the status.

The `victext` utility is designed to help you learn how to use the NI-VXI functions to program devices. When you develop a sequence of steps that works successfully for your system, you can easily incorporate the sequence into an application program using the appropriate language and syntax.

The VXIbus status (BERR, SYSFAIL, ACFAIL, VXI interrupts, TTL/ECL triggers) is also monitored and displayed.

Running victext

Type *`victext`* to run the `victext` program. The `help` command gives online information about `victext` commands with a quick reference for checking syntax and function of the NI-VXI call. Type *`help`* followed by a command name to get more information about the particular command.

`victext` reads the `resman.tbl` file and several help (`.hlp`) files.

victext Command Descriptions

Command syntaxes in `victext` are different from function calls. The syntax of each of the commands is described in the remainder of this chapter.

System Configuration Commands

The `victext` utility supports the following system configuration commands.

finddevla

Purpose: Returns the logical address of a device with the specified attributes.

Command Syntax:

```
finddevla <namepat>[, <manid>[, <modelcode>[, <devclass>[,
<slot>[, <mainframe>,[ <cmdrla>]]]]]]
```

where

`<namepat>`	Name pattern
`<manid>`	VXI manufacturer identification number
`<modelcode>`	Manufacturer's 12-bit model number
`<devclass>`	Device class of the device 0 = Memory Class Device 1 = Extended Class Device 2 = Message-Based Device 3 = Register-Based Device
`<slot>`	Slot location of the device
`<mainframe>`	Mainframe location of device (logical address of extender)
`<cmdrla>`	Commander's logical address

If `namepat` is `""` or any other attribute is -1 or missing, that attribute is not used in the matching algorithm.

Example: Find the logical address of a device with the device name `"GPIB-VXI`."

```
finddevla "GPIB-VXI"
```

getdevinfo

Purpose: Returns the desired information about a device in the NI-VXI RM table.

Command Syntax: `getdevinfo [<la>, ]<field>`

where

`<la>` Logical address of device to get information about

`<field>` Field identification number

Field	Description
0	Entire RM table entry for specified device
1	Device name
2	Commander's logical address
3	Mainframe
4	Slot
5	Manufacturer identification number
6	Manufacturer name
7	Model code
8	Model name
9	Device class
10	Extended subclass (if extended class device)
11	Address space used
12	Base of A24/A32 memory
13	Size of A24/A32 memory
14	Memory type and access time
15	Bit vector list of VXI interrupter lines
16	Bit vector list of VXI interrupt handler lines
17	Extender/controller information

Bits	Description
15 to 13	Reserved
12	1 = Child side extender 0 = Parent side extender
11	1 = Frame extender 0 = Not frame extender
10	1 = Extended controller
9	1 = Embedded controller
8	1 = External controller
7 to 0	Frame extender towards root frame

18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

Example: Get the name of a device at Logical Address 4.

```
getdevinfo 4, 1
```

setdevinfo

Purpose: Sets the desired information about a device in the NI-VXI RM table.

Command Syntax: `setdevinfo [<la>, ]<field>, <fieldvalue>`

where

`<la>` Logical address of device in RM table for which to set information

`<field>` Field identification number

Field	Description
0	Entire RM table entry for specified device
1	Device name
2	Commander's logical address
3	Mainframe
4	Slot
5	Manufacturer identification number
6	Manufacturer name
7	Model code
8	Model name
9	Device class
10	Extended subclass (if extended class device)
11	Address space used
12	Base of A24/A32 memory
13	Size of A24/A32 memory
14	Memory type and access time
15	Bit vector list of VXI interrupter lines
16	Bit vector list of VXI interrupt handler lines
17	Extender/controller information

Bits	Description
15 to 13	Reserved
12	1 = Child side extender 0 = Parent side extender
11	1 = Frame extender 0 = Not frame extender
10	1 = Extended controller
9	1 = Embedded controller
8	1 = External controller
7 to 0	Frame extender towards root frame

18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

`<fieldvalue>` Appropriate value to set for the specified field

Example: Set the name of a device at Logical Address 4 to "RELAY."

```
setdevinfo 4, 1, "RELAY"
```

createdevinfo

Purpose: Creates a new entry in the RM table for the device at specified logical address.

Command Syntax:

```
createdevinfo [<la>]
```

where

`<la>`	Logical address of device for which to create entry

Example: Create a new entry for a device at Logical Address 4.

```
createdevinfo 4
```

Commander Word Serial Protocol Commands

The `victext` utility supports the following Commander Word Serial Protocol commands.

wsrd

Purpose: Reads data from a Message-Based device and displays it on the console using the VXIbus Byte Transfer Protocol.

Command Syntax: `wsrd [<la>, ]<count>, <mode>`

where

`<la>` Logical address of Message-Based Servant

`<count>` Maximum number of bytes to transfer

`<mode>` Mode of transfer (bit vector)

Bit	Description
0	Not DOR 0 = Abort if not DOR 1 = Poll till DOR
1	END bit termination suppression 0 = Terminate transfer on END bit 1 = Do not terminate transfer on END
2	LF character termination 1 = Terminate transfer on LF bit 0 = Do not terminate transfer on LF
3	CR character termination 1 = Terminate transfer on CR bit 0 = Do not terminate transfer on CR
4	EOS character termination 1 = Terminate transfer on EOS bit 0 = Do not terminate transfer on EOS
8 to 15	EOS character (if enabled)

Example: Read and display 10 bytes from the device at Logical Address 5 (mode = poll until DOR; terminate transfer on END bit).

```
wsrd 5, 10, 1
```

wsrdf

Purpose: Transfers data from a Message-Based device to a file, using the VXIbus Byte Transfer Protocol.

Command Syntax:

```
wsrdf [<la>, ]<filename>, <count>, <mode>
```

where

`<la>`	Logical address of Message-Based Servant
`<filename>`	File name to which to transfer data
`<count>`	Maximum number of bytes to transfer
`<mode>`	Mode of transfer (bit vector) (same as in `wsrd`)

Example: Read 10 bytes from a device at Logical Address 5 (mode = poll till DOR, terminate transfer on END bit), and write them to the file `"temp.dat."`

```
wsrdf 5, "temp.dat", 10, 1
```

---

wswrt

Purpose: Transfers the specified string to a Message-Based Servant using the VXIbus Byte Transfer Protocol.

Command Syntax:

```
wswrt [<la>, ]<string>[, <mode>]
```

where

`<la>`	Logical address of Message-Based Servant
`<string>`	Buffer to send
`<mode>`	Mode of transfer (bit vector)

Bit	Description
0	Not DIR 0 = Abort if not DIR 1 = Poll till DIR
1	END bit termination 0 = Clear END bit on the last byte 1 = Set END bit on the last byte

Example: Send the string `"laddrs?"` to a device at Logical Address 5.

```
wswrt 5, "laddrs?"
```

---

wswrtf

Purpose: Transfers data from the specified file to a Message-Based Servant using the VXIbus Byte Transfer Protocol.

Command Syntax: `wswrtf [<la>, ]<filename>[, <count>]`

where

`<la>`	Logical address of Message-Based Servant
`<filename>`	Name of file from which to transfer data
`<count>`	Maximum number of bytes to transfer

Example: Send the data in the file `"temp.dat"` to a device at Logical Address 5.

```
wswrtf 5, "temp.dat"
```

---

wscmd

Purpose: Sends a Word Serial command to a Message-Based Servant device.

Command Syntax: `wscmd [<la>, ]<command>`

where

`<la>`	Logical address of Message-Based Servant
`<command>`	Word Serial command

Example: Send an *Identify Commander* command to a device at Logical Address 5.

```
wscmd 5, 0xbe00
```

---

wscmd?

Purpose: Sends a Word Serial query to a Message-Based Servant device and gets the response.

Command Syntax: `wscmd? [<la>, ]<command>`

where

`<la>`	Logical address of Message-Based Servant
`<command>`	Word Serial query

Example: Send a *Read STB* command to a device at Logical Address 5 and get the response.

```
wscmd? 5, 0xcfff
```

---

wsresp

Purpose: Gets the response to the previously sent Word Serial query (when sent as a command using `wscmd` rather than as a query using `wscmd?`). This function is intended for debug use only.

Command Syntax: `wsresp [<la>]`

where

`<la>`	Logical address of Message-Based Servant

Example: Get the response.

```
wsresp 5
```

---

wstrg

Purpose: Sends the Word Serial *Trigger* command to a Message-Based Servant device.

Command Syntax: `wstrg [<la>]`

where

`<la>`	Logical address of Message-Based Servant

Example: Send the Word Serial *Trigger* command to a device at Logical Address 5.

```
wstrg 5
```

---

wsclr

Purpose: Sends the Word Serial Protocol command *Clear* to a Message-Based Servant device.

Command Syntax: `wsclr [<la>]`

where

`<la>`	Logical address of Message-Based Servant

Example: Send a *Clear* command to a device at Logical Address 5.

```
wsclr 5
```

---

wsabort

Purpose: Aborts the Commander Word Serial operation(s) currently in progress.

Command Syntax: `wsabort [<la>, ]<abortop>`

where

`<la>` Logical address of Message-Based Servant

`<abortop>` Option for aborting Word Serial operation

1 = `ForcedAbort`: aborts `wswrt`, `wsrd`, and `wstrg`
2 = `UnRecCom`: aborts `wscmd`, `wslcmd`, and `wsecmd`
3 = `ForcedAbort`: aborts `wscmd`, `wslcmd`, and `wsecmd`
4 = `ForcedAbort`: aborts all Word Serial operations
5 = `Async ForcedAbort`: aborts all Word Serial operations immediately.

Note: Be cautious when using `<abortop>` option 5. During a Word Serial query, the Servant may be left in an invalid state if the operation is aborted after writing the query and before reading the response register. When using this option, the Word Serial operation is aborted immediately as compared to using options 1, 3, and 4, where the operation is not aborted until the response is read in that situation.

Example: Abort `wswrt` operation to Logical Address 5.

```
wsabort 5, 1
```

---

wslcmd

Purpose: Sends a Longword Serial command to a Message-Based Servant device and gets the response.

Command Syntax: `wslcmd [<la>, ]<command>, <respflag>`

where

`<la>` Logical address of Message-Based Servant

`<command>` Longword Serial command

`<respflag>` Response flag

0 = Do not get the response
1 = Get the response

Example: Send a user-defined Longword command 0xfffffffe to a device at Logical Address 5 and get the response.

```
wslcmd 5, 0xfffffffe, 1
```

---

wslresp

Purpose: Gets the response for the previously sent Longword query.

Command Syntax:

```
wslresp [<la>]
```

where

`<la>`	Logical address of Message-Based Servant

Example: Gets the response for the previously sent Longword query from Logical Address 5.

```
wslresp 5
```

---

wsecmd

Purpose: Sends an Extended Longword Serial command to a Message-Based Servant device and gets the response.

Command Syntax:

```
wsecmd [<la>, ]<cmdext>, <command>, <respflag>
```

where

`<la>`	Logical address of Message-Based Servant
`<cmdext>`	Upper 16 bits of 48-bit Extended Longword Serial command
`<command>`	Lower 32 bits of 48-bit Extended Longword Serial command
`<respflag>`	Response flag 0 = Do not get the response 1 = Get the response

Example: Send a user-defined Extended Longword command 0xfffdffffffe to a device at Logical Address 5 and get the response.

```
wsecmd 5, 0xfffd, 0xfffffffe, 1
```

---

wssettmo

Purpose: Sets the timeout period for all Word Serial commands.

Command Syntax:

```
wssettmo <timeout>
```

where

`<timeout>`	Timeout period (in milliseconds)

Example: Set the timeout value to 3 seconds (3,000 ms).

```
wssettmo 3000
```

---

wsgettmo

Purpose: Gets the timeout period set for all Word Serial commands.

Command Syntax: `wsgettmo`

Example: Get the timeout value.

```
wsgettmo
```

Servant Word Serial Protocol Commands

The `victext` utility supports the following Servant Word Serial Protocol commands.

wssenable

Purpose: Enables all Servant Word Serial Protocol commands.

Command Syntax:
```
wssenable
```

Example: Enable all Servant Word Serial Protocol commands.

```
wssenable
```

---

wssdisable

Purpose: Inhibits all Servant Word Serial Protocol commands from being received.

Command Syntax:
```
wssdisable
```

Example: Inhibit all Servant Word Serial Protocol commands from being received.

```
wssdisable
```

---

wssrd

Purpose: Receives the specified number of bytes from a Message-Based Commander using the VXIbus Byte Transfer Protocol.

Command Syntax:
```
wssrd <count>, <mode>
```

where

`<count>`	Maximum number of bytes to receive
`<mode>`	Mode of transfer (bit vector) (Bit 0) 0 = Do not send DIR signal to Commander 1 = Send DIR signal to Commander

Example: Receive 10 bytes from the Commander device.

```
wssrd 10, 0
```

---

wsswrt

Purpose: Transfers the specified string to a Message-Based Commander using the VXIbus Byte Transfer Protocol.

Command Syntax: `wsswrt <string>, <mode>`

where

`<string>` Buffer to send

`<mode>` Mode of transfer (bit vector)

(Bit 0)
0 = Do not send DOR signal to Commander
1 = Send DOR signal to Commander

(Bit 1)
0 = Do not send END with the last byte
1 = Send END with the last byte

Example: Send the string `"mydata 1.2.3"` to the Commander device.

```
wsswrt "mydata 1.2.3", 2
```

---

wssabort

Purpose: Aborts the Servant Word Serial operation currently in progress.

Command Syntax: `wssabort <abortop>`

where

`<abortop>` Option for aborting Word Serial operation (bit vector)

Bit	Option
1	Aborts `wsswrt`
2	Aborts `wssrd`
3	Aborts `wsssendresp`
15	Initialize Word Serial Servant hardware. This includes aborting all Word Serial Servant operations, clearing all errors, removing all pending Word Serial Servant interrupts, and disabling the interrupts.

Example: Abort `wsswrt` operation to Logical Address 5.

```
wssabort 5, 1
```

High-Level VXIbus Access Commands

The `victext` utility supports the following high-level VXIbus access commands.

vxiin

Purpose: Reads a single byte, word, or longword from the specified address in the specified address space with the specified access parameters.

Command Syntax: `vxiin <accessparms>, <address>, <width>`

where

`<accessparms>` (Bits 0 to 1) VXI address space
- 1 = A16
- 2 = A24
- 3 = A32

(Bits 2 to 4) Access privilege
- 0 = Nonprivileged data access
- 1 = Supervisory data access
- 2 = Nonprivileged program access
- 3 = Supervisory program access
- 4 = Nonprivileged block access
- 5 = Supervisory block access

(Bits 5 to 6)
- 0

(Bit 7) Byte order
- 0 = Motorola
- 1 = Intel

(Bits 8 to 15)
- 0

`<address>` VXI address

`<width>` Data width
- 1 = Byte
- 2 = Word
- 4 = Longword

Example: Read the ID register (2 bytes) of a device at Logical Address 4 (address = 0xc100).

```
vxiin 1, 0xc100, 2
```

vxiout

Purpose: Writes a single byte, word, or longword to the specified address in the specified address space with the specified access parameters.

Command Syntax:

```
vxiout <accessparms>, <address>, <width>, <value>
```

where

`<accessparms>` (Bits 0 to 1) VXI address space
- 1 = A16
- 2 = A24
- 3 = A32

(Bits 2 to 4) Access privilege
- 0 = Nonprivileged data access
- 1 = Supervisory data access
- 2 = Nonprivileged program access
- 3 = Supervisory program access
- 4 = Nonprivileged block access
- 5 = Supervisory block access

(Bits 5 to 6)
- 0

(Bit 7) Byte order
- 0 = Motorola
- 1 = Intel

(Bits 8 to 15)
- 0

`<address>` VXI address

`<width>` Data transfer width
- 1 = Byte
- 2 = Word
- 4 = Longword

`<value>` Value to write

Example: Write a value of 0x1000 to the Offset register (2 bytes) of a device at Logical Address 4 (address = 0xc106).

```
vxiout 1, 0xc106, 2, 0x1000
```

---

vxiinreg

Purpose: Reads a single word from the specified register offset on the specified logical address.

Command Syntax: `vxiinreg [<la>, ]<reg>`

where

`<la>`	Logical address
`<reg>`	Offset within VXI Logical Address registers

Example: Read the ID register (reg = 0) of a device at Logical Address 4.

```
vxiinreg 4, 0
```

---

vxioutreg

Purpose: Writes a single word to the specified register offset on the specified logical address.

Command Syntax: `vxioutreg [<la>, ]<reg>, <value>`

where

`<la>`	Logical address
`<reg>`	Offset within VXI Logical Address registers
`<value>`	Value to write

Example: Write the Offset register (reg = 6) of a device at Logical Address 4 with 0x2000.

```
vxioutreg 4, 6, 0x2000
```

---

vximove

Purpose: Moves a block of bytes, words, or longwords from a source location in any address space to a destination in any address space with the specified access parameters.

Command Syntax: `vximove <srcparms>, <srcaddr>, <destparms>, <destaddr>, <length>, <width>`

where

`<srcparms>` and `<destparms>`

(Bits 0 to 1) Source and destination address space
- 0 = Local
- 1 = A16
- 2 = A24
- 3 = A32

(Bits 2 to 4) Access privilege
- 0 = Nonprivileged data access
- 1 = Supervisory data access
- 2 = Nonprivileged program access
- 3 = Supervisory program access
- 4 = Nonprivileged block access
- 5 = Supervisory block access

(Bits 5 to 6)
- 0

(Bit 7) Byte order
- 0 = Motorola
- 1 = Intel

(Bits 8 to 15)
- 0

`<srcaddr>` and `<destaddr>` — Address (offsets within source and destination address spaces)

`<length>` — Number of elements to transfer

`<width>` — Data transfer width
- 1 = Byte
- 2 = Word
- 4 = Longword

Example: Move 100 bytes from A24 space at 0x200000 to A32 space at 0x20000000.

```
vximove 2, 0x200000, 3, 0x20000000, 100, 1
```

Local Resource Access Commands

The `victext` utility supports the following local resource access commands.

getmyla

Purpose: Gets the logical address of the local VXI device.

Command Syntax: `getmyla`

Example: Get the logical address of the local VXI device.

```
getmyla
```

---

vxiinlr

Purpose: Reads a single byte, word, or longword from the specified register on the local VXI device.

Command Syntax: `vxiinlr <reg>, <width>`

where

`<reg>`	Offset within VXI Logical Address registers
`<width>`	Data transfer width 1 = Byte 2 = Word 4 = Longword

Example: Read the register at offset 0 (2 bytes) on the local device.

```
vxiinlr 0, 2
```

---

vxioutlr

Purpose: Writes a single byte, word, or longword from the specified register on the local VXI device.

Command Syntax:
```
vxioutlr <reg>, <width>, <value>
```

where

`<reg>`	Offset within VXI Logical Address registers
`<width>`	Data transfer width 1 = Byte 2 = Word 4 = Longword
`<value>`	Value to write

Example: Write the register at offset 0 (2 bytes) with 0x1000 on the local device.

```
vxioutlr 0, 2, 0x1000
```

---

setmodid

Purpose: Controls the MODID lines of the VXIbus backplane.

Command Syntax:
```
setmodid <enable>, <modid>
```

where

`<enable>`	1 = Set MODID enable bit 0 = Clear MODID enable bit
`<modid>`	Bit vector for Bits 0 to 12, corresponding to Slots 0 to 12

Example: Set all the MODID lines.

```
setmodid 1, 0xffff
```

---

readmodid

Purpose: Reads the MODID lines of the VXIbus backplane.

Command Syntax:
```
readmodid
```

Example: Read the MODID lines.

```
readmodid
```

VXI Signal Commands

The `victext` utility supports the following VXI signal commands.

routesignal

Purpose: Selects whether each type of signal is enqueued or handled by the signal handler.

Command Syntax: `routesignal [<la>, ]<modemask>`

where

`<la>` Logical address to set handler for

`<modemask>` A bit vector that specifies whether each type of signal is enqueued or handled by the signal handler. A zero in any bit position causes signals of the associated type to be queued. All other signals are handled by the signal handler.

If `la` is a Message-Based device:

Bit	Event Signal
14	User-Defined events
13	VXI Reserved events
12	Shared Memory events
11	Unrecognized Command events
10	Request False (REQF) events
9	Request True (REQT) events
8	No Cause Given events

Bit	Response Signal
7	Unused
6	B14
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

If `la` is *not* a Message-Based device:

Bit	Type of Signal (status/Id) values
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)

Example: Put REQT, REQF and *Unrecognized Command* signals on the signal queue and handle the rest of the signals using the signal handler for Logical Address 5.

```
routesignal 5, 0xf1ff
```

enablesignalint

Purpose: Sensitizes `victext` to receive all signals.

Command Syntax: `enablesignalint`

---

disablesignalint

Purpose: Desensitizes `victext` to not receive signals.

Command Syntax: `disablesignalint`

---

signaldeq

Purpose: Gets a signal specified by the signalmask for the specified logical address from the signal queue.

Command Syntax: `signaldeq [<la>, ]<signalmask>`

where

`<la>`	Logical address of signal to dequeue for (-1 = any)
`<signalmask>`	Bit vector (same as in `routesignal` command) (0xffff = any)

Example: Dequeue the first signal on the signal queue.

```
signaldeq -1, 0xffff
```

---

signalenq

Purpose: Puts the specified signal on the tail of the signal queue for the logical address.

Command Syntax: `signalenq <signal>`

where

`<signal>`	Signal to enqueue at the end of the queue

Example: Enqueue the signal 0x1111 on the end of signal queue.

```
signalenq 0x1111
```

---

signaljam

Purpose: Puts the signal on the head of the signal queue for the logical address.

Command Syntax: `signaljam <signal>`

where

`<signal>`	Signal to jam on the front of the queue

Example: Enqueue the signal 0x1111 on the front of signal queue.

```
signaljam 0x1111
```

---

waitforsignal

Purpose: Waits for one of the specified signals to be received.

Command Syntax: `waitforsignal [<la>, ]<signalmask>, <timeout>`

where

`<la>`	Logical address of the device sourcing the signal (-1 = any)
`<signalmask>`	Bit vector (same as in `routesignal` command) (0xffff = any)
`<timeout>`	Time period to wait (in milliseconds)

Example: Wait for two seconds for REQT signal from Logical Address 5.

```
waitforsignal 5, 0x0200, 2000
```

VXI Interrupt Commands

The `victext` utility supports the following VXI interrupt commands.

routevxiint

Purpose: Selects whether VXI interrupts are handled as VXI signals or routed to the VXI interrupt handler for the specified controller.

Command Syntax: `routevxiint <controller>, <sroute>`

where

`<controller>`	Controller for which to configure interrupt routing -1 = Embedded controller or the immediate extended controller
`<sroute>`	Bit vector of VXI interrupt levels to route. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Handle as a VXI signal for appropriate level 0 = Handle as a VXI interrupt for appropriate level

Example: Handle VXI interrupt 4 as a VXI signal and handle the rest as VXI interrupts for the controller at Logical Address 5.

```
routevxiint 5, 0x08
```

---

enablevxitosignalint

Purpose: Sensitizes `victext` to the specified VXI interrupt levels being processed as VXI signals for the specified controller.

Command Syntax: `enablevxitosignalint <controller>, <levels>`

where

`<controller>`	Controller for which to enable VXI interrupt -1 = Embedded controller or the immediate extended controller
`<levels>`	Bit vector of VXI interrupt levels to enable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Enable for appropriate level

Example: Enable VXI interrupt 4 on the controller at Logical Address 5 to be handled as a VXI signal.

```
enablevxitosignalint 5, 0x08
```

---

disablevxitosignalint

Purpose: Desensitizes `victext` to the specified VXI interrupt levels being processed as VXI signals for the specified controller.

Command Syntax: `disablevxitosignalint <controller>, <levels>`

where

`<controller>`	Controller for which to disable VXI interrupt -1 = Embedded controller or the immediate extended controller
`<levels>`	Bit vector of VXI interrupt levels to disable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Disable for appropriate level

Example: Disable VXI interrupt 4 on the controller at Logical Address 5 to be handled as a VXI signal.

```
disablevxitosignalint 5, 0x08
```

---

enablevxiint

Purpose: Sensitizes `victext` to the specified VXI interrupt levels being processed as VXI/VME interrupts (not as VXI signals) for the specified controller.

Command Syntax: `enablevxiint <controller>, <levels>`

where

`<controller>`	Controller for which to enable VXI interrupt -1 = Embedded controller or the immediate extended controller
`<levels>`	Bit vector of VXI interrupt levels to enable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Enable for appropriate level

Example: Enable VXI interrupt 4 on the controller at Logical Address 5 to be handled as a VXI/VME interrupt (not as a VXI signal).

```
enablevxiint 5, 0x08
```

---

disablevxiint

Purpose: Desensitizes `victext` to the specified VXI interrupt levels being processed as VXI/VME interrupts (not as VXI signals) for the specified controller.

Command Syntax:

```
disablevxiint <controller>, <levels>
```

where

`<controller>`	Controller for which to disable VXI interrupt -1 = Embedded controller or the immediate extended controller
`<levels>`	Bit vector of VXI interrupt levels to disable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Disable for appropriate level

Example: Disable VXI interrupt 4 on the controller at Logical Address 5 to be handled as a VXI/VME interrupt (not as a VXI signal).

```
disablevxiint 5, 0x08
```

---

vxiintacknowledgemode

Purpose: Specifies whether the VXI interrupt acknowledge cycle for the specified controller for the specified levels should be handled as Release On Acknowledge (ROAK) or as Release On Register Access (RORA).

Command Syntax:

```
vxiintacknowledgemode <controller>, <modes>
```

where

`<controller>`	Controller to configure -1 = Embedded controller or the immediate extended controller
`<modes>`	Bit vector of VXI interrupt levels to set to ROAK or RORA 1 = Set to RORA VXI interrupt 0 = Set to ROAK VXI interrupt

Example: Handle VXI interrupt 2 as RORA and the rest as ROAK for controller at Logical Address 5.

```
vxiintacknowledgemode 5, 0x02
```

---

assertvxiint

Purpose: Asserts an interrupt line in a particular controller.

Command Syntax:

```
assertvxiint <controller>, <level>, <statusId>
```

where

`<controller>`	Controller for which to assert VXI interrupt -1 = Embedded controller or the immediate extended controller
`<level>`	Interrupt level
`<statusId>`	Status/ID to present during IACK cycle

Example: Assert interrupt 4 in Controller 1 with status/ID of 0x1234.

```
assertvxiint 1, 4, 0x1234
```

---

deassertvxiint

Purpose: Deasserts an interrupt line in a particular controller.

Command Syntax:

```
deassertvxiint <controller>, <level>
```

where

`<controller>`	Controller for which to deassert VXI interrupt -1 = Embedded controller or the immediate extended controller
`<level>`	Interrupt level

Example: Deassert interrupt 4 in Controller 1.

```
deassertvxiint 1, 4
```

---

acknowledgevxiint

Purpose: Performs an IACK cycle on the VXIbus in the specified controller for the specified level.

Command Syntax:

```
acknowledgevxiint <controller>, <level>
```

where

`<controller>`	Controller for which to acknowledge VXI interrupt -1 = Embedded controller or the immediate extended controller
`<level>`	Interrupt level

Example: Acknowledge interrupt 4 in Controller 1.

```
acknowledgevxiint 1, 4
```

VXI Trigger Commands

The `victext` utility supports the following VXI trigger commands.

srctrig

Purpose: Sources the specified protocol on the specified TTL, ECL, or external trigger line on the specified controller.

Command Syntax: `srctrig <controller>, <line>, <protocol>, <timeout>`

where

`<controller>` Controller on which to source trigger line
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to source

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
40 to 49	External source/destination (GPIO 0 to 9)
50	TIC counter
60	TIC TICK timers

`<protocol>` Trigger Protocol

0 = Continuous ON
1 = Continuous OFF (Disable)
2 = Start
3 = Stop
4 = Sync
5 = Semi-sync
6 = Async
7 = Semi-sync (and wait for ACK)
8 = Async (and wait for ACK)

`<timeout>` Timeout value in milliseconds

Example: Source VXIbus TTL trigger line 4 on Controller 1 with Continuous ON protocol.

```
srctrig 1, 4, 0, 0
```

enabletrigsense

Purpose: Enables the sensing of the specified trigger line, or starts up the counter or tick timer for the specified protocol.

Command Syntax: `enabletrigsense <controller>, <line>, <protocol>`

where

`<controller>` Controller on which to enable sensing of TTL trigger line
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to enable sensing

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC tick timers

`<protocol>` Trigger Protocol

2 = Start
3 = Stop
4 = Sync
5 = Semi-sync
6 = Async

Example: Enable sensing of VXIbus TTL trigger line 4 on Controller 1 for Semi-sync protocol.

```
enabletrigsense 1, 4, 5
```

---

disabletrigsense

Purpose: Disables the sensing of the specified trigger line, counter, or tick timer that was enabled by `enabletrigsense`.

Command Syntax: `disabletrigsense <controller>, <line>`

where

`<controller>` Controller on which to disable sensing of TTL trigger line
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to disable sensing

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC tick timers

Example: Disable sensing of VXIbus TTL trigger line 4 on Controller 1.

```
disabletrigsense 1, 4
```

---

waitfortrig

Purpose: Waits for the specified TTL trigger line to be encountered on the specified controller.

Command Syntax:

```
waitfortrig <controller>, <line>, <timeout>
```

where

`<controller>` Controller waiting for trigger line to be encountered
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to wait on

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC tick timers

`<timeout>` Timeout value in milliseconds

Example: Wait for VXIbus TTL trigger line 4 to be sourced on Controller 1.

```
waitfortrig 1, 4, 10000
```

---

acknowledgetrig

Purpose: Acknowledges the specified TTL/ECL or external trigger on the specified controller.

Command Syntax:

```
acknowledgetrig <controller>, <line>
```

where

`<controller>` Controller on which to acknowledge trigger interrupt
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to acknowledge

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
40 to 49	External source/destination lines 0 to 9

Example: Acknowledge a trigger interrupt for TTL line 4 on Controller 1.

```
acknowledgetrig 1, 4
```

---

maptrigtotrig

Purpose: Maps the specified TTL, ECL, Star X, Star Y, external connection, or miscellaneous signal line to another.

Command Syntax: `maptrigtotrig <controller>, <srctrig>, <desttrig>, <mode>`

where

`<controller>` Controller on which to map signal lines
-1 = Embedded controller or the immediate extended controller

`<srctrig>` Source line to map to destination

`<desttrig>` Destination line to map from source

Value	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12
27 to 39	Star Y lines 0 to 12
40 to 49	External source/destination lines 0 to 9
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	Connection to EXTCLK input pin
44 to 49	Hardware-dependent external lines 4 to 9
50	TIC counter pulse output (TCNTR)
51	TIC counter finished output (GCNTR)
60	TIC TICK1 tick timer output
61	TIC TICK2 tick timer output

`<mode>` Signal conditioning mode (0 = no conditioning)

Bit	Conditioning Effect
0	Synchronize with next CLK10 edge
1	Invert signal polarity
2	Pulse stretch to one CLK minimum
3	Use EXTCLK (not CLK10) for conditioning

All other values reserved for future expansion

Example: Map VXIbus TTL trigger line 4 on Controller 1 out the front panel.

```
maptrigtotrig 1, 4, 41, 0
```

unmaptrigtotrig

Purpose: Unmaps the specified TTL, ECL, Star X, Star Y, external connection, or miscellaneous signal line that was mapped to another line using `maptrigtotrig`.

Command Syntax:

```
unmaptrigtotrig <controller>, <srctrig>, <desttrig>
```

where

`<controller>` Controller on which to unmap signal lines
-1 = Embedded controller or the immediate extended controller

`<srctrig>` Source line to unmap to destination

`<desttrig>` Destination line mapped from source

Value	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12
27 to 39	Star Y lines 0 to 12
40 to 49	External source/destination lines 0 to 9
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	Connection to EXTCLK input pin
44 to 49	Hardware-dependent external lines 4 to 9
50	TIC counter pulse output (TCNTR)
51	TIC counter finished output (GCNTR)
60	TIC TICK1 tick timer output
61	TIC TICK2 tick timer output

Example: Unmap VXIbus TTL trigger line 4 on Controller 1 that was mapped out the front panel.

```
unmaptrigtotrig 1, 4, 49
```

trigassertconfig

Purpose: Configures the specified TTL/ECL trigger line assertion method.

Command Syntax: `trigassertconfig <controller>, <line>, <mode>`

where

`<controller>` Controller on which to configure assertion mode
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to configure

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
ffffh	General assertion configuration (all lines)

`<mode>` Configuration mode

Bit	Specific Line Configuration Modes
0	1 = Synchronize falling edge of CLK10 0 = Synchronize rising edge of CLK10

Bit	General Configuration Modes
1	1 = Pass trigger through asynchronously 0 = Synchronize with next CLK10 edge
2	1 = Participate in Semi-sync with external trigger acknowledge protocol 0 = Do not participate

Example: Configure TTL trigger line 4 on Controller 1 to synchronize to CLK10 for any assertion method and do not participate in Semi-sync.

```
trigassertconfig 1, 4, 0
```

trigcntrconfig

Purpose: Configures TIC chip internal 16 bit counter.

Command Syntax: `trigcntrconfig <controller>, <mode>, <source>, <count>`

where

`<controller>` Controller on which to configure the TIC counter
-1 = Embedded controller or the immediate extended controller

`<mode>` Configuration mode

Value	Configuration Modes
0	Initialize the counter
2	Reload the counter leaving enabled
3	Disable/abort any count in progress

`<source>` Trigger line to configure as input to counter

Value	Trigger line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
70	CLK10
71	EXTCLK connection

`<count>` Number of input pulses to count before terminating

Example: Configure the counter to count 25 assertions on TTL trigger line 5 on Controller 1.

```
trigcntrconfig 1, 0, 5, 25
```

trigextconfig

Purpose: Configures the external trigger lines.

Command Syntax: `trigextconfig <controller>, <line>, <mode>`

where

`<controller>` Controller on which to configure the external connection
-1 = Embedded controller or the immediate extended controller

`<line>` Trigger line to configure

Value	Trigger line
40 to 49	External source/destination lines 0 to 9
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	EXTCLK
44 to 49	Hardware-dependent external lines 4 to 9

`<mode>` Configuration mode

Bit	Configuration Modes
0	1 = Feed back any line mapped as input into the crosspoint switch 0 = Drive input to external pin
1	1 = Assert input (regardless of feedback) 0 = Leave input unconfigured
2	1 = If assertion selected, assert low 0 = If assertion selected, assert high
3	1 = Invert external input (not feedback) 0 = Pass external input unchanged

All other values are reserved for future expansion.

Example: Configure external line 41 (front panel Out) to not be fed back and left tristated for use as a mapped output via `maptrigtotrig`.

```
trigextconfig -1, 41, 0
```

trigtickconfig

Purpose: Configures TIC chip internal dual 5 bit tick timers.

Command Syntax: `trigtickconfig <controller>, <mode>, <source>, <tcount1>, <tcount2>`

where

`<controller>` Controller on which to configure the TIC chip internal dual 5 bit tick timers
-1 = Embedded controller or the immediate extended controller

`<mode>` Configuration mode

Value	Configuration Modes
0	Initialize the tick timers (rollover mode)
1	Initialize the tick timers (non-rollover mode)
2	Reload the tick timers leaving enabled
3	Disable/abort any count in progress

`<source>` Trigger line to configure as input to counter

Value	Trigger line
40 to 49	External source/destination lines 0 to 9
70	CLK10
71	EXTCLK connection

`<tcount1>` Number of input pulses (as a power of two) to count before asserting TICK1 output (and terminating the tick timer if configured for non-rollover mode)

`<tcount2>` Number of input pulses (as a power of two) to count before asserting TICK2 output

Example: Configure the tick timers to interrupt every 6.55 milliseconds by dividing down CLK10 as an input.

```
trigtickconfig -1, 0, 70, 16, 0
```

System Interrupt Handler Commands

The `victext` utility supports the following system interrupt handler commands.

enablesysfail

Purpose: Sensitizes `victext` to SYSFAIL interrupts from the specified controller.

Command Syntax: `enablesysfail <controller>`

where

`<controller>`	Controller on which to enable interrupt -1 = Embedded controller or the first extended controller

Example: Enable the SYSFAIL interrupt on the embedded CPU (or first extended controller).

```
enablesysfail -1
```

---

disablesysfail

Purpose: Desensitizes `vixtext` to SYSFAIL interrupts from the specified controller.

Command Syntax: `disablesysfail <controller>`

where

`<controller>`	Controller on which to disable interrupt -1 = Embedded controller or the first extended controller

Example: Disable the SYSFAIL interrupt on the embedded CPU (or first extended controller).

```
disablesysfail -1
```

---

enableacfail

Purpose: Sensitizes `victext` to ACFAIL interrupts from the specified controller.

Command Syntax: `enableacfail <controller>`

where

`<controller>`	Controller on which to enable interrupt -1 = Embedded controller or the first extended controller

Example: Enable the ACFAIL interrupt on the embedded CPU (or first extended controller).

```
enableacfail -1
```

---

disableacfail

Purpose: Desensitizes `victext` to ACFAIL interrupts from the specified controller.

Command Syntax: `disableacfail <controller>`

where

`<controller>` Controller on which to disable interrupt
-1 = Embedded controller or the first extended controller

Example: Disable the ACFAIL interrupt on the embedded CPU (or first extended controller).

```
disableacfail -1
```

---

enablesoftreset

Purpose: Enables the local Soft Reset interrupt being generated from a write to the Reset bit of the local CPU's Control register.

Command Syntax: `enablesoftreset`

Example: Enable the local Soft Reset interrupt.

```
enablesoftreset
```

---

disablesoftreset

Purpose: Disables the local Soft Reset interrupt being generated from a write to the Reset bit of the local CPU's Control register.

Command Syntax: `disablesoftreset`

Example: Disable the local Soft Reset interrupt.

```
disablesoftreset
```

---

assertsysreset

Purpose: Asserts SYSRESET* on the backplane of the specified controller.

Command Syntax: `assertsysreset <controller>`

where

`<controller>` Controller on which to assert SYSRESET*

-1 = From embedded controller or the first extended controller
-2 = All extenders

`<mode>` Mode of execution

0 = Do not disturb original configuration
1 = Force link between SYSRESET* and local reset (SYSRESET* resets local CPU)
2 = Break link between SYSRESET and local reset (SYSRESET* does not reset local CPU)

Example: Assert the SYSRESET* interrupt on the embedded CPU (or first extended controller) without changing the current configuration.

```
assertsysreset -1, 0
```

---

enablesysreset

Purpose: Sensitizes `victext` to SYSRESET* interrupts from the specified controller.

Command Syntax: `enablesysreset <controller>`

where

`<controller>` Controller on which to enable interrupt
-1 = Embedded controller or the first extended controller

Example: Enable the SYSRESET* interrupt on the embedded CPU (or first extended controller).

```
enablesysfail -1
```

---

disablesysreset

Purpose: Desensitizes `victext` to SYSRESET* interrupts from the specified controller.

Command Syntax:

```
disablesysreset <controller>
```

where

`<controller>` Controller on which to disable interrupt
-1 = Embedded controller or the first extended controller

Example: Disable the SYSRESET* interrupt on the embedded CPU (or first extended controller).

```
disablesysfail -1
```

Bus Extender Commands

The `victext` utility supports the following Bus Extender commands.

mapecltrig

Purpose: Routes the VXIbus ECL trigger lines for the specified VXI extender in the specified directions.

Command Syntax: `mapecltrig <extender>, <lines>, <directions>`

where

`<extender>`	Logical address of VXI extender
`<lines>`	Bit vector of ECL lines to enable. Bits 5 to 0 correspond to ECL lines 5 to 0, respectively. 1 = Enable for appropriate line 0 = Disable for appropriate line
`<directions>`	Bit vector of directions for ECL lines. Bits 5 to 0 correspond to ECL lines 5 to 0, respectively. 1 = Into the VXI extender 0 = Out of the VXI extender

Example: Route VXIbus ECL trigger line 4 out of the mainframe through the VXI extender at Logical Address 1.

```
mapecltrig 1, 0x08, 0
```

mapttltrig

Purpose: Routes the VXIbus TTL trigger lines for the specified VXI extender in the specified directions.

Command Syntax:

```
mapttltrig <extender>, <lines>, <directions>
```

where

`<extender>`	Logical address of VXI extender
`<lines>`	Bit vector of TTL lines to enable. Bits 7 to 0 correspond to TTL lines 7 to 0, respectively. 1 = Enable for appropriate line 0 = Disable for appropriate line
`<directions>`	Bit vector of directions for TTL lines. Bits 7 to 0 correspond to TTL lines 7 to 0, respectively. 1 = Into the VXI extender 0 = Out of the VXI extender

Example: Route VXIbus TTL trigger line 4 out of the mainframe through the VXI extender at Logical Address 1.

```
mapttltrig 1, 0x08, 0
```

---

maputilbus

Purpose: Maps the utility bus for the specified VXI extender.

Command Syntax:

```
maputilbus <extender>, <utilmode>
```

where

`<extender>`	Logical address of VXI extender
`<utilmode>`	Bit vector of utility bus signals corresponding to the utility bus signals. 1 = Enable for corresponding signal and direction 0 = Disable for corresponding signal and direction

Bit	Utility Bus Signal and Direction
5	ACFAIL into the mainframe
4	ACFAIL out of the mainframe
3	SYSFAIL into the mainframe
2	SYSFAIL out of the mainframe
1	SYSRESET* into the mainframe
0	SYSRESET* out of the mainframe

Example: Map SYSFAIL into the mainframe through the VXI extender at Logical Address 5. Map SYSRESET into and out of the mainframe. Do not map ACFAIL at all.

```
maputilbus 5, 0xB
```

---

mapvxiint

Purpose: Maps the specified VXI interrupts for the specified VXI extender in the specified directions.

Command Syntax:

```
mapvxiint <extender>, <levels>, <directions>
```

where

`<extender>`	Logical address of VXI extender
`<levels>`	Bit vector of VXI interrupt levels to enable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Enable for appropriate level 0 = Disable for appropriate level
`<directions>`	Bit vector of directions for VXI interrupt levels. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively. 1 = Into the VXI extender 0 = Out of the VXI extender

Example: Map VXI interrupt 4 on the extender at Logical Address 5 to go out of the mainframe.

```
mapvxiint 5, 0x08, 0x00
```

Auxiliary Commands

The `victext` utility supports the following auxiliary commands.

disablemonitor

Purpose: Disables the VXIbus monitor, which monitors the status of the VXIbus in the system.

Command Syntax: `disablemonitor`

---

enablemonitor

Purpose: Enables the VXIbus monitor, which monitors the status of the VXIbus in the system, through the specified controller. If no controller is specified, enables the monitoring for the last controller being modified (-2 = OR of all controllers).

Command Syntax: `enablemonitor [<controller>]`

Example: Monitor Controller 24.

```
enablemonitor 24
```

---

set

Purpose: Invokes device-level calls. The command line prompt changes to the device name. At this stage, you no longer need to specify `<la>` in the `victext` Word Serial Protocol and some auxiliary commands.

Command Syntax: `set <la>` or `<devname>`

Note: `<la>` refers to logical address of device. `<devname>` refers to the name of the device inside `" "`. Partial names will work for `<devname>`.

Example: Set for the device call for `"GPIB-VXI`."

```
set "GPIB-VXI"
```

---

pparms

Purpose: Sets the specified access parameters. These access parameters are used in `peek` and `poke` `victext` commands. If no parameter is specified, it displays the current access parameters.

Command Syntax: `pparms [<accessparms>]`

where

`<accessparms>` (Bits 0 to 1) VXI address space
1 = A16
2 = A24
3 = A32

(Bits 2 to 4) access privilege
0 = Nonprivileged data access
1 = Supervisory data access
2 = Nonprivileged program access
3 = Supervisory program access
4 = Nonprivileged block access
5 = Supervisory block access

(Bits 5 to 6)
0

(Bit 7) Byte order
0 = Motorola
1 = Intel

(Bits 8 to 15)
0

Example: Set access parameters to A16 space, nonprivileged data, and Motorola byte order.

```
pparms 1
```

---

pwidth

Purpose: Sets the specified width. This width is used in `peek` and `poke` `victext` commands. If no parameter is specified, it displays the current width.

Command Syntax: `pwidth [<width>]`

where

`<width>` Data transfer width

1 = Byte
2 = Word
4 = Longword

Example: Set width as byte transfer.

```
pwidth 1
```

---

peek

Purpose: Reads a single byte, word, or longword (depending on the width set by the `pwidth` `victext` command) from the specified address (in the address space with access parameters set by the `pparms` `victext` command).

Command Syntax:

```
peek <address>
```

where

`<address>`	VXI address

Example: Read the ID register of a device at Logical Address 4.

```
peek 0xc100
```

---

poke

Purpose: Writes a single byte, word, or longword (depending on the width set by the `pwidth` `victext` command) to the specified address (in the address space with access parameters set by the `pparms` `victext` command).

where

`<address>`	VXI address
`<value>`	Value to write

Command Syntax:

```
poke <address>, <value>
```

Example: Write a value of 0x1000 to the Offset register of a device at Logical Address 4.

```
poke 0xc106, 0x1000
```

---

scripton

Purpose: Captures the information as it is displayed on the `victext` screen and writes it to the specified file.

Command Syntax:
```
scripton <mode>, <filename>
```

where

`<mode>`	0 = Writes only the commands 1 = Writes both the `victext` commands entered and the responses
`<filename>`	File to which the information is to be written

Example: Capture all the information from `victext` and write it to the file `"temp.dat`."

```
scripton 1, "temp.dat"
```

---

scriptoff

Purpose: Closes the script session.

Command Syntax:
```
scriptoff
```

---

cfon

Purpose: Enables generation of a corresponding C function call for NI-VXI library calls.

Command Syntax:
```
cfon
```

---

cfoff

Purpose: Disables generation of a corresponding C function call for NI-VXI library calls.

Command Syntax:
```
cfoff
```

---

rmentry?

Purpose: Returns Resource Manager (RM) information about a device.

Command Syntax: `rmentry? [<la>]`

where

`<la>` VXI logical address

Example: Return RM information about a device at Logical Address 4.

```
rmentry? 4
```

---

laddrs?

Purpose: Gives a list of known logical addresses.

Command Syntax: `laddrs?`

---

numladdrs?

Purpose: Gives the number of known logical addresses.

Command Syntax: `numladdrs?`

---

cmdrtable?

Purpose: Displays the system hierarchy information.

Command Syntax: `cmdrtable?`

---

a16memmap?

Purpose: Gives A16 memory map of the system.

Command Syntax: `a16memmap?`

---

a24memmap?

Purpose: Gives A24 memory map of the system.

Command Syntax: `a24memmap?`

---

a32memmap?

Purpose: Gives A32 memory map of the system.

Command Syntax: `a32memmap?`

---

readregister?

Purpose: Reads the register on the specified logical address.

Command Syntax: `readregister? <la>, <reg>[, <"OFF/ON">]`

where

`<la>`	VXI logical address
`<reg>`	Offset within VXI Logical Address registers

Note: The flag "ON" or "OFF" (default = "OFF") gives more information on the individual fields of the register. Valid registers are all even numbers between 0 and 62 as well as the following register names: A24PL, A24PH, A32PL, A32PH, ATTR, DHIG, DLOW, DTYP, ICON, ID, IST, OFFS, PROT, RESP, SNL, SNH, STAT, SUBC, VNUM.

Example: Read the ID register of a device at Logical Address 4.

```
readregister? 4, 0, "ON"
```

---

writeregister

Purpose: Writes the register (16 bits) on the specified logical address with the specified value.

Command Syntax:

```
writeregister <la>, <reg>, <value>
```

where

`<la>`	VXI logical address
`<reg>`	Offset within VXI Logical Address registers
`<value>`	Value to write

Note: Valid registers are even numbers between 0 and 62 as well as the following register names: CONT, DEXT, DHIG, DLOW, ICON, LADD, OFFS, SIGN.

Example: Write the Offset register of a device at Logical Address 4 with 0x2000.

```
writeregister 4, OFFS, 0x2000
```

---

devicenumber?

Purpose: Gets the number of devices in the system if issued by the Resource Manager; otherwise, it gives the number of immediate Servants.

Command Syntax:

```
devicenumber?
```

Example: Find the number of devices in the system.

```
devicenumber?
```

---

deviceladd?

Purpose: Gets a list of all the devices in the system if issued by the Resource Manager; otherwise, it gives a list of all the immediate Servants.

Command Syntax:

```
deviceladd?
```

Example: Find all the Servants.

```
deviceladd?
```

---

deviceconfigure?

Purpose: Gives the current hierarchy information for the specified logical address. If the logical address is not specified, it gives the current hierarchy information for all the devices in the system.

Command Syntax: `deviceconfigure? [<la>]`

where

`<la>`	VXI logical address

Example: Get the hierarchy information for device at Logical Address 5.

```
deviceconfigure? 5
```

---

deviceinformation?

Purpose: Gives the device information for the specified logical address. If the logical address is not specified, it gives the information for all the devices in the system.

Command Syntax: `deviceinformation? [<la>]`

where

`<la>`	VXI logical address

Example: Get the device information for the Logical Address 5.

```
deviceinformation? 5
```

---

devicereset?

Purpose: Soft resets the device for the specified logical address. SYSFAIL generation is inhibited while the device is in the self-test state. The command waits for five seconds or until the selected device has indicated passed (whichever occurs first). If the device passes its self-test, SYSFAIL generation is re-enabled. If the device fails its self-test, SYSFAIL generation is inhibited.

Command Syntax: `devicereset? <la>`

where

`<la>`	VXI logical address

Example: Soft reset the device at Logical Address 4.

```
devicereset? 4
```

$

Purpose: Executes `victext` commands from a file.

Command Syntax: `$ <filename>`

where

`<filename>` File from which to execute commands

Example: Execute the `victext` commands in the file `"temp.dat`."

```
$ "temp.dat"
```

---

history

Purpose: Displays the last 20 `victext` commands.

Command Syntax: `history`

Example: Display the last 20 `victext` commands.

```
history
```

!

Purpose: Executes the specified `victext` command. If no parameter is specified, it executes the last `victext` command in the history of commands.

Command Syntax: `! [<command number>` or `<command string>]`

where

`<command number>` Command number (from `history`)

`<command string>` Command string

Example: Execute the command number 10 in the history of `victext` commands.

```
! 10
```

*****

Purpose: Executes the last `victext` command for the specified number of times. If no parameter is specified, it executes the last `victext` command once.

Command Syntax: `* [<repeat number>]`

where

`<repeat number>` Number of times to execute the last command

Example: Execute the last `victext` command 10 times.

```
* 10
```

---

version

Purpose: Shows the version of the NI-VXI software in use.

Command Syntax: `version`

---

system

Purpose: Exits the `victext` utility temporarily to the operating system (not supported in some operating systems).

Command Syntax: `system`

---

quit

Purpose: Quits the `victext` utility.

Command Syntax: `quit`

Appendix A
NI-VXI File Formats

This appendix describes the NI-VXI file formats. All of the following files created by the NI-VXI software are ASCII files. Each field in an entry is separated by a comma. Different entries in the file are separated by a carriage return.

resman.tbl

This file is created by the NI-VXI Resource Manager (RM). It contains all of the information about devices found in the system. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Logical address	16-bit integer in decimal	`%hd`
Device name	12 characters	`%[^,]`
Commander's logical address	16-bit integer in hex	`%hx`
Mainframe (extender logical address)	16-bit integer in hex	`%hx`
Slot	16-bit integer in hex	`%hx`
Manufacturer ID	16-bit integer in hex	`%hx`
Manufacturer name	12 characters	`%[^,]`
Model code	16-bit integer in hex	`%hx`
Model name	12 characters	`%[^,]`
VXI device class	16-bit integer in hex	`%hx`
VXI extended subclass	16-bit integer in hex	`%hx`
VXI address space	16-bit integer in hex	`%hx`
VXI address base for memory	32-bit integer in hex	`%lx`
VXI address size for memory	32-bit integer in hex	`%lx`
Memory class attributes	16-bit integer in hex	`%hx`
VXI interrupter levels	16-bit integer in hex	`%hx`
VXI interrupt handler levels	16-bit integer in hex	`%hx`
Extender/controller information	16-bit integer in hex	`%hx`
WS Asynch Mode Control response	16-bit integer in hex	`%hx`
WS Control Response response	16-bit integer in hex	`%hx`
Copy of Protocol register	16-bit integer in hex	`%hx`
Miscellaneous flags	16-bit integer in hex	`%hx`
Status state	16-bit integer in hex	`%hx`

mfnameid.tbl

This file is used by the NI-VXI RM to associate manufacturer names with manufacturer IDs. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Manufacturer name	12 characters	`%[^,]`
Manufacturer ID	16-bit integer in hex	`%hx`

model.tbl

This file is used by the NI-VXI RM to associate model names with model codes. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Model name	12 characters	`%[^,]`
Manufacturer name	12 characters	`%[^,]`
Model code	16-bit integer in hex	`%hx`

device.tbl

This file is used by the NI-VXI RM to associate device names with particular devices in a particular VXI system. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Device name	12 characters	`%[^,]`
Manufacturer name	12 characters	`%[^,]`
Model name	12 characters	`%[^,]`
Logical address	16-bit integer in hex	`%hx`
Mainframe	16-bit integer in hex	`%hx`
Slot	16-bit integer in hex	`%hx`

nonvxi.tbl

This file is used by the NI-VXI RM to avoid allocating VXI devices' resources that would conflict with non-VXI devices in the system. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Pseudo logical address	16-bit integer in hex	`%hx`
Device name	12 characters	`%[^,]`
Mainframe	16-bit integer in hex	`%hx`
Slot	16-bit integer in hex	`%hx`
Manufacturer ID	16-bit integer in hex	`%hx`
Model code	16-bit integer in hex	`%hx`
VXI device class	16-bit integer in hex	`%hx`
Model name	12 characters	`%[^,]`
Manufacturer name	12 characters	`%[^,]`
A16 address base	32-bit integer in hex	`%lx`
A16 address size	32-bit integer in hex	`%lx`
A24 address base	32-bit integer in hex	`%lx`
A24 address size	32-bit integer in hex	`%lx`
A32 address base	32-bit integer in hex	`%lx`
A32 address size	32-bit integer in hex	`%lx`
VME interrupter levels	16-bit integer in hex	`%hx`
VME interrupt handler levels	16-bit integer in hex	`%hx`
In system Boolean flag	16-bit integer in hex	`%hx`

intcfg.tbl

This file is used by the NI-VXI RM to associate a particular VXI interrupt line in a particular VXI mainframe to a VXI interrupt handler. It also defines the routing of VXI interrupts in and out of a particular mainframe. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Extender logical address	16-bit integer in hex	`%hx`
Interrupter levels (bit vector)	16-bit integer in hex	`%hx`
Interrupter directions (bit vector)	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 1	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 2	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 3	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 4	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 5	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 6	16-bit integer in hex	`%hx`
Handler for VXI interrupt level 7	16-bit integer in hex	`%hx`

trigcfg.tbl

This file defines the routing of VXI triggers in and out of a particular mainframe. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Extender logical address	16-bit integer in hex	`%hx`
TTL trigger lines (bit vector)	16-bit integer in hex	`%hx`
TTL trigger directions (bit vector)	16-bit integer in hex	`%hx`
ECL trigger lines (bit vector)	16-bit integer in hex	`%hx`
ECL trigger directions (bit vector)	16-bit integer in hex	`%hx`

utilbus.tbl

This file defines the routing of SYSRESET, SYSFAIL, and ACFAIL in and out of a particular mainframe. Each entry in the file consists of the following components.

Field Name	Field Syntax	C Syntax (in `sscanf` terms)
Extender logical address	16-bit integer in hex	`%hx`
Utility mode (value to write to reg.)	16-bit integer in hex	`%hx`

Appendix B
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Corporate Headquarters
(512) 795-8248
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices	**Phone Number**	**Fax Number**
Australia	(03) 879 9422	(03) 879 9179
Austria	(0662) 435986	(0662) 437010-19
Belgium	02/757.00.20	02/757.03.11
Denmark	45 76 26 00	45 76 71 11
Finland	(90) 527 2321	(90) 502 2930
France	(1) 48 14 24 00	(1) 48 14 24 14
Germany	089/741 31 30	089/714 60 35
Italy	02/48301892	02/48301915
Japan	(03) 3788-1921	(03) 3788-1923
Netherlands	03480-33466	03480-30673
Norway	32-848400	32-848600
Spain	(91) 640 0085	(91) 640 0533
Sweden	08-730 49 70	08-730 43 70
Switzerland	056/20 51 51	056/27 00 25
U.K.	0635 523545	0635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___)_____

Computer brand _____ Model _____ Processor _____

Operating system _____

Speed _____ MHz RAM _____ MB Display adapter _____

Mouse _____yes _____ no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is _____

_____

_____

_____

_____

List any error messages _____

_____

_____

_____

The following steps will reproduce the problem _____

_____

_____

_____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **NI-VXI™ Text Utilities Reference Manual**

Edition Date: **October 1993**

Part Number: **320321-01**

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
µ-	micro-	10^{-6}
n-	nano-	10^{-9}
k-	kilo-	10^{3}
M-	mega-	10^{6}
G-	giga-	10^{9}

A

A16 space	One of the VXIbus address spaces. Equivalent to the VME 64K *short* address space. In VXI, the upper 16K of A16 space is allocated for use by VXI devices configuration registers. This 16K region is referred to as VXI configuration space.
A24 space	One of the VXIbus address spaces. Equivalent to the VME 16M *standard* address space.
A32 space	One of the VXIbus address spaces. Equivalent to the VME 4 GB *extended* address space.
ACFAIL*	A VMEbus backplane signal that is asserted when a power failure has occurred (either AC line source or power supply malfunction), or if it is necessary to disable the power supply (such as for a high temperature condition).
address	Character code that identifies a specific location (or series of locations) in memory.
address modifier	One of six signals in the VMEbus specification used by VMEbus masters to indicate the address space and mode (supervisory/nonprivileged, data/program/block) in which a data transfer is to take place.
address space	A set of 2^n memory locations differentiated from other such sets in VXI/VMEbus systems by six signal lines known as address modifiers. *n* is the number of address lines required to uniquely specify a byte location in a given space. Valid numbers for *n* are 16, 24, and 32.
address window	A range of address space that can be accessed from the application program.
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange. A 7-bit standard code adopted to facilitate the interchange of data among various types of data processing and data communications equipment.
asserted	A signal in its active true state.
ASYNC Protocol	A two-device, two-line handshake trigger protocol using two consecutive even/odd trigger lines (a source/acceptor line and an acknowledge line).
asynchronous	Not synchronized; not controlled by periodic time signals, and therefore unpredictable with regard to the timing of execution of commands.

B

backplane	An assembly, typically a PCB, with 96-pin connectors and signal paths that bus the connector pins. A C-size VXIbus system will have two sets of bused connectors called the J1 and J2 backplanes. A D-size VXIbus system will have three sets of bused connectors called the J1, J2, and J3 backplane.
base address	A specified address that is combined with a *relative* address (or offset) to determine the *absolute* address of a data location. All VXI address windows have an associated base address for their assigned VXI address spaces.
BERR*	Bus Error signal. This signal is asserted by either a slave device or the BTO unit when an incorrect transfer is made on the Data Transfer Bus (DTB). The BERR* signal is also used in VXI for certain protocol implementations such as writes to a full Signal register and synchronization under the Fast Handshake Word Serial Protocol.
binary	A numbering system with a base of 2.
bit	Binary digit. The smallest possible unit of data: a two-state, true/false, 1/0 alternative. The building block of binary coding and numbering systems. Eight bits make up a *byte*.
bit vector	A string of related bits in which each bit has a specific meaning.
BNO	Begin Normal Operation command
byte	A grouping of adjacent binary digits operated on by the computer as a single unit. In VXI systems, a byte consists of 8 bits.
byte order	How bytes are arranged within a word or how words are arranged within a longword. Motorola ordering stores the most significant byte (MSB) or word first, followed by the least significant byte (LSB) or word. Intel ordering stores the LSB or word first, followed by the MSB or word.

C

CLK10	A 10 MHz, ± 100 ppm, individually buffered (to each module slot), differential ECL system clock that is sourced from Slot 0 and distributed to Slots 1 through 12 on P2. It is distributed to each slot as a single-source, single-destination signal with a matched delay of under 8 ns.
command	A directive to a device. In VXI, three types of commands are as follows: In Word Serial Protocol, a 16-bit imperative to a servant from its Commander (written to the Data Low register); In Shared Memory Protocol, a 16-bit imperative from a client to a server, or vice versa (written to the Signal register); In Instrument devices, an ASCII-coded, multi-byte directive.
commander	A Message-Based device which is also a bus master and can control one or more Servants.
communications registers	In Message-Based devices, a set of registers that are accessible to the device's Commander and are used for performing Word Serial Protocol communications.
configuration registers	A set of registers through which the system can identify a module device type, model, manufacturer, address space, and memory requirements. In order to support automatic system and memory configuration, the VXIbus specification requires that all VXIbus devices have a set of such registers.

controller	An intelligent device (usually involving a CPU) that is capable of controlling other devices.
CR	Carriage Return; the ASCII character 0Dh.

D

DC	See *dynamically configured device*.
decimal	Numbering system based upon the ten digits 0 to 9. Also known as base 10.
default handler	Automatically installed at startup to handle associated interrupt conditions; the software can then replace it with a specified handler.
DIR	Data In Ready
DOR	Data Out Ready
dynamically configured device	A device that has its logical address assigned by the Resource Manager. A VXI device initially responds at Logical Address 255 when its MODID line is asserted. A MXIbus device responds at Logical Address 255 during a priority select cycle. The Resource Manager subsequently assigns it a new logical address, which the device responds to until powered down.

E

ECL	Emitter-Coupled Logic
embedded controller	An intelligent CPU (controller) interface plugged directly into the VXI backplane, giving it direct access to the VXIbus. It must have all of its required VXI interface capabilities built in.
END	Signals the end of a data string.
EOS	End Of String; a character sent to designate the last byte of a data message.
ERR	Protocol error
Extended Class device	A class of VXIbus device defined for future expansion of the VXIbus specification. These devices have a subclass register within their configuration space that defines the type of extended device.
extended controller	A mainframe extender with additional VXIbus controller capabilities.
extended longword	Data type of 48-bit integers.

F

FHS	Fast Handshake; a mode of the Word Serial Protocol which uses the VXIbus signals DTACK* and BERR* for synchronization instead of the Response register bits.

G

GPIB	General Purpose Interface Bus; the industry-standard IEEE 488 bus.

H

handshaking	A type of protocol that makes it possible for two devices to synchronize operations.
hex	Hexadecimal; the numbering system with base 16, using the digits 0 to 9 and letters A to F.

I

IACK	Interrupt Acknowledge
IEEE	Institute of Electrical and Electronics Engineers
IEEE 1014	The VME specification
interrupt	A means for a device to notify another device that an event occurred.
interrupt handler	A functional module that detects interrupt requests generated by interrupters and performs appropriate actions.
interrupter	A device capable of asserting interrupts and responding to an interrupt acknowledge cycle.
IRQ	Interrupt Request

K

KB	1,024 or 2^{10}
kilobyte	A thousand bytes.

L

LF	Linefeed; the ASCII character 0Ah.
logical address	An 8-bit number that uniquely identifies the location of each VXIbus device's configuration registers in a system. The A16 register address of a device is C000h + Logical Address * 40h.
longword	Data type of 32-bit integers.

M

MB	1,048,576 or 2^{20}
mapping	Establishing a range of address space for a one-to-one correspondence between each address in the window and an address in VXIbus memory.

megabyte	A million bytes.
Memory Class device	A VXIbus device that, in addition to configuration registers, has memory in VME A24 or A32 space that is accessible through addresses on the VME/VXI data transfer bus.
Message-Based device	An intelligent device that implements the defined VXIbus registers and communication protocols. These devices are able to use Word Serial Protocol to communicate with one another through communication registers.
MODID	Module Identification lines; a set of 13 signal lines on the VXI backplane that VXI systems use to identify which modules are located in which slots in the mainframe.
MXIbus	Multisystem eXtension Interface Bus; a high-performance communication link that interconnects devices using round, flexible cables.

N

NI-VXI	The National Instruments bus interface software for VME/VXIbus systems.
nonprivileged access	One of the defined types of VMEbus data transfers; indicated by certain address modifier codes. Each of the defined VMEbus address spaces has a defined nonprivileged access mode.

O

octal	Numbering system with base 8, using numerals 0 to 7.

P

peek	To read the contents.
PH device	programmable interrupt handler
PI device	programmable interrupter
poke	To write a value.
privileged access	See *Supervisory Access*.
protocol	Set of rules or conventions governing the exchange of information between computer systems.

Q

query	Like command, causes a device to take some action, but requires a response containing data or other information. A command does not require a response.

R

read	To get information from any input device or file storage media.

register	A high-speed device used in a CPU for temporary storage of small amounts of data or intermediate results during processing.
Register-Based device	A Servant-only device that supports only the four basic VXIbus configuration registers. Register-Based devices are typically controlled by Message-Based devices via device-dependent register reads and writes.
REQF	Request False; a VXI Event condition transferred using either VXI signals or VXI interrupts, indicating that a Servant no longer has a need for service.
REQT	Request True; a VXI Event condition transferred using either VXI signals or VXI interrupts, indicating that a Servant has a need for service.
resman	The name of the National Instruments Resource Manager application in the NI-VXI bus interface software. See *Resource Manager*.
Resource Manager	A Message-Based Commander located at Logical Address 0, which provides configuration management services such as address map configuration, Commander and Servant mappings, and self-test and diagnostic management.
RM	See *Resource Manager*.
ROAK	Release On Acknowledge; a type of VXI interrupter which always deasserts its interrupt line in response to an IACK cycle on the VXIbus. All Message-Based VXI interrupters must be ROAK interrupters.
RORA	Release On Register Access; a type of VXI/VME interrupter which does not deassert its interrupt line in response to an IACK cycle on the VXIbus. A device-specific register access is required to remove the interrupt condition from the VXIbus. The VXI specification recommends that VXI interrupters be only ROAK interrupters.
RR	Read Ready; a bit in the Response register of a Message-Based device used in Word Serial Protocol indicating that a response to a previously sent query is pending.

S

s	seconds
SC device	See *statically configured device*.
SEMI-SYNC Protocol	A one-line, open collector, multiple-device handshake trigger protocol.
servant	A device controlled by a Commander; there are Message-Based and Register-Based Servants.
setting	To place a binary cell into the 1 (non-zero) state.
signal	Any communication between Message-Based devices consisting of a write to a Signal register. Sending a signal requires that the sending device have VMEbus master capability.
statically configured device	A device whose logical address cannot be set through software; that is, it is not dynamically configurable.

status/ID	A value returned during an IACK cycle. In VME, usually an 8-bit value which is either a status/data value or a vector/ID value used by the processor to determine the source. In VXI, a 16-bit value used as a data; the lower 8 bits form the VXI logical address of the interrupting device and the upper 8 bits specify the reason for interrupting.
STST	START/STOP trigger protocol; a one-line, multiple-device protocol which can be sourced only by the VXI Slot 0 device and sensed by any other device on the VXI backplane.
supervisory access	One of the defined types of VMEbus data transfers; indicated by certain address modifier codes.
SYNC Protocol	The most basic trigger protocol, simply a pulse of a minimum duration on any one of the trigger lines.
synchronous communications	A communications system that follows the command/response cycle model. In this model, a device issues a command to another device; the second device executes the command and then returns a response. Synchronous commands are executed in the order they are received.
SYSFAIL*	A VMEbus signal that is used by a device to indicate an internal failure. A failed device asserts this line. In VXI, a device that fails also clears its PASSed bit in its Status register.
SYSRESET*	A VMEbus signal that is used by a device to indicate a system reset or power-up condition.
system hierarchy	The tree structure of the Commander/Servant relationships of all devices in the system at a given time. In the VXIbus structure, each Servant has a Commander. A Commander can in turn be a Servant to another Commander.

T

trigger	Either TTL or ECL lines used for intermodule communication.
TTL	Transistor-Transistor Logic

U

unasserted	A signal in its inactive false state.

V

VME	Versa Module Eurocard or IEEE 1014
victext	VXI Interactive Control program, a part of the NI-VXI bus interface software package. Used to program VXI devices, and develop and debug VXI application programs.
VXIbus	VMEbus Extensions for Instrumentation

vxitedit	VXI Resource Editor program, a part of the NI-VXI bus interface software package. Used to configure the system, edit the manufacturer name and ID numbers, edit the model names of VXI and non-VXI devices in the system, as well as the system interrupt configuration information, and display the system configuration information generated by the Resource Manager.

W

Word Serial Protocol	The simplest required communication protocol supported by Message-Based devices in the VXIbus system. It utilizes the A16 communication registers to perform 16-bit data transfers using a simple polling handshake method.
word	A data quantity consisting of 16 bits.
write	Copying data to a storage device.
WR	Write Ready; a bit in the Response register of a Message-Based device used in Word Serial Protocol indicating the ability for a Servant to receive a single command/query written to its Data Low register.

Index

Symbols

! command, 4-52
$ command, 4-52
* command, 4-53

A

a16memmap? command, 4-48
a24memmap? command, 4-49
a32memmap? command, 4-49
abort command, vxitedit, 3-2
acknowledgetrig command, 4-30
acknowledgevxiint command, 4-27
add command, vxitedit, 3-2
address map configuration
    Servant-side operation, Message-Based, 2-4
    Servant-side operation, non-Message-Based, 2-5
    VXI Resource Manager operation, 2-3
assertsysreset command, 4-39
assertvxiint command, 4-27
Auxiliary commands
    !, 4-52
    $, 4-52
    *, 4-53
    a16memmap?, 4-48
    a24memmap?, 4-49
    a32memmap?, 4-49
    cfoff, 4-47
    cfon, 4-47
    cmdrtable?, 4-48
    deviceconfigure?, 4-51
    deviceinformation?, 4-51
    deviceladd?, 4-50
    devicenumber?, 4-50
    devicereset?, 4-51
    disablemonitor, 4-44
    enablemonitor, 4-44
    history, 4-52
    laddrs?, 4-48
    numladdrs?, 4-48
    peek, 4-46
    poke, 4-46
    pparms, 4-45
    pwidth, 4-45
    quit, 4-53
    readregister?, 4-49
    rmentry?, 4-48
    scriptoff, 4-47
    scripton, 4-47
    set, 4-44
    system, 4-53
    version, 4-53
    writeregister, 4-50

B

Bus Extender commands
    mapecltrig, 4-41
    mapttltrig, 4-42
    maputilbus, 4-42
    mapvxiint, 4-43

C

cfoff command, 4-47
cfon command, 4-47
cmdrtable? command, 4-48
command line options, for Startup RM, 2-6
Commander/Servant hierarchies
    Servant-side operation, Message-Based, 2-4
    Servant-side operation, non-Message-Based, 2-5
    VXI Resource Manager operation, 2-3
Commander Word Serial Protocol commands
    wsabort, 4-10
    wsclr, 4-9
    wscmd, 4-8
    wscmd?, 4-8
    wsecmd, 4-11
    wsgettmo, 4-12
    wslcmd, 4-10
    wslresp, 4-11
    wsrd, 4-6
    wsrdf, 4-7
    wsresp, 4-9
    wssettmo, 4-11
    wstrg, 4-9
    wswrt, 4-7
    wswrtf, 4-8
commands
    Auxiliary commands
        !, 4-52
        $, 4-52
        *, 4-53
        a16memmap?, 4-48
        a24memmap?, 4-49
        a32memmap?, 4-49
        cfoff, 4-47
        cfon, 4-47
        cmdrtable?, 4-48
        deviceconfigure?, 4-51
        deviceinformation?, 4-51
        deviceladd?, 4-50

devicenumber?, 4-50
devicereset?, 4-51
disablemonitor, 4-44
enablemonitor, 4-44
history, 4-52
laddrs?, 4-48
numladdrs?, 4-48
peek, 4-46
poke, 4-46
pparms, 4-45
pwidth, 4-45
quit, 4-53
readregister?, 4-49
rmentry?, 4-48
scriptoff, 4-47
scripton, 4-47
set, 4-44
system, 4-53
version, 4-53
writeregister, 4-50
Bus Extender commands
mapecltrig, 4-41
mapttltrig, 4-42
maputilbus, 4-42
mapvxiint, 4-43
Commander Word Serial Protocol commands
wsabort, 4-10
wsclr, 4-9
wscmd, 4-8
wscmd?, 4-8
wsecmd, 4-11
wsgettmo, 4-12
wslcmd, 4-10
wslresp, 4-11
wsrd, 4-6
wsrdf, 4-7
wsresp, 4-9
wssettmo, 4-11
wstrg, 4-9
wswrt, 4-7
wswrtf, 4-8
High-Level VXIbus Access commands
vxiin, 4-15
vxiinreg, 4-17
vximove, 4-18
vxiout, 4-16
vxioutreg, 4-17
Local Resource Access commands
getmyla, 4-19
readmodid, 4-20
setmodid, 4-20
vxiinlr, 4-19
vxioutlr, 4-20
Servant Word Serial Protocol commands
wssabort, 4-14
wssdisable, 4-13
wssenable, 4-13
wssrd, 4-13
wsswrt, 4-14
System Configuration commands
createdevinfo, 4-5
finddevla, 4-2
getdevinfo, 4-3
setdevinfo, 4-4
System Interrupt Handler commands
assertsysreset, 4-39
disableacfail, 4-38
disablesoftreset, 4-38
disablesysfail, 4-37
disablesysreset, 4-40
enableacfail, 4-37
enablesoftreset, 4-38
enablesysfail, 4-37
enablesysreset, 4-39
VXI Interrupt commands
acknowledgevxiint, 4-27
assertvxiint, 4-27
deassertvxiint, 4-27
disablevxiint, 4-26
disablevxitosignalint, 4-25
enablevxiint, 4-25
enablevxitosignalint, 4-24
routevxiint, 4-24
vxiintacknowledgemode, 4-26
VXI Signal commands
disablesignalint, 4-22
enablesignalint, 4-22
routesignal, 4-21
signaldeq, 4-22
signalenq, 4-22
signaljam, 4-23
waitforsignal, 4-23
VXI Trigger commands
acknowledgetrig, 4-30
disabletrigsense, 4-29
enabletrigsense, 4-29
maptrigtotrig, 4-31
srctrig, 4-28
trigassertconfig, 4-33
trigcntrconfig, 4-34
trigextconfig, 4-35
trigtickconfig, 4-36
unmaptrigtotrig, 4-32
waitfortrig, 4-30
vxitedit commands
summary of, 3-2
configuration. *See also* System Configuration commands.
address map configuration, 2-3, 2-4, 2-5
for Startup RM, 2-6
configuration editor
definition of, 3-1
files used by, 3-4
use of, 3-5
createdevinfo command, 4-5
customer communication, *x*, B-1

D

deassertvxiint command, 4-27
delete command, vxitedit, 3-2
device name editor
    definition of, 3-1
    files used by, 3-4
    use of, 3-8
deviceconfigure? command, 4-51
deviceinformation? command, 4-51
deviceladd? command, 4-50
devicenumber? command, 4-50
devicereset? command, 4-51
device.tbl file format, A-2
disableacfail command, 4-38
disablemonitor command, 4-44
disablesignalint command, 4-22
disablesoftreset command, 4-38
disablesysfail command, 4-37
disablesysreset command, 4-40
disabletrigsense command, 4-29
disablevxiint command, 4-26
disablevxitosignalint command, 4-25
display command, vxitedit, 3-2
documentation
    conventions used in, *ix*
    how to use, *x*
    organization of, *ix*
    related documentation, *x*
duplicate command, vxitedit, 3-2

E

enableacfail command, 4-37
enablemonitor command, 4-44
enablesignalint command, 4-22
enablesoftreset command, 4-38
enablesysfail command, 4-37
enablesysreset command, 4-39
enabletrigsense command, 4-29
enablevxiint command, 4-25
enablevxitosignalint command, 4-24
error-checking, Startup RM, 2-5
exit command, vxitedit, 3-2

F

file access
    Startup RM, 2-5
    victext, 4-1
    vxitedit, 3-4
file formats
    device.tbl, A-2
    intcfg.tbl, A-3
    mfnameid.tbl, A-1
    model.tbl, A-2
    nonvxi.tbl, A-2
    resman.tbl, A-1
    trigcfg.tbl, A-3
    utilbus.tbl, A-3
finddevla command, 4-2

G

getdevinfo command, 4-3
getmyla command, 4-19

H

help, vxitedit editors, 3-2
High-Level VXIbus Access commands
    vxiin, 4-15
    vxiinreg, 4-17
    vximove, 4-18
    vxiout, 4-16
    vxioutreg, 4-17
history command, 4-52

I

initiating normal operation. *See* normal operation, initiating.
intcfg.tbl file format, A-3
interrupt commands. *See* System Interrupt Handler commands; VXI Interrupt commands.
interrupt configuration editor
    definition of, 3-2
    files used by, 3-4
    single-mainframe example, 3-11
    multiple-mainframe example, 3-12
    use of, 3-11
IRQ line allocation
    Servant-side operation, Message-Based, 2-4
    Servant-side operation, non-Message-Based, 2-5
    VXI Resource Manager operation, 2-3

L

laddrs? command, 4-48
list command, vxitedit, 3-2
Local Resource Access commands
    getmyla, 4-19
    readmodid, 4-20
    setmodid, 4-20
    vxiinlr, 4-19
    vxioutlr, 4-20
Logical Address 0 operation, 1-1 to 1-2, 2-1 to 2-3

M

mainframe extenders
multiple mainframe interrupt, trigger, and utility bus extension, 2-3
Startup RM support for, 1-2, 2-1
VXI device identification, 2-2
manufacturer name editor
definition of, 3-1
files used by, 3-4
use of, 3-6
mapecltrig command, 4-41
maptrigtotrig command, 4-31
mapttltrig command, 4-42
maputilbus command, 4-42
mapvxiint command, 4-43
Message-Based Servant-side operation. *See* Servant-side operation, Message-Based.
mfnameid.tbl file format, A-1
model name editor
definition of, 3-1
files used by, 3-4
use of, 3-7
model.tbl file format, A-2
modify command, vxitedit, 3-2
multimainframe example, 3-12

N

NI-VXI file formats. *See* file formats.
NI-VXI Text utilities. *See* RM; victext; vxiinit; vxitedit.
non-Logical Address 0 operation, 1-2, 2-4 to 2-5
non-Message-Based Servant-side operation. *See* Servant-side operation, non-Message-Based.
non-VXI device editor
definition of, 3-1
files used by, 3-4
use of, 3-9
non-VXI device location, 2-1
nonvxi.tbl file format, A-2
normal operation, initiating
Servant-side operation, Message-Based, 2-4
Servant-side operation, non-Message-Based, 2-5
VXI Resource Manager operation, 2-3
numladdrs? command, 4-48

O

options for running Startup RM, 2-6

P

peek command, 4-46
poke command, 4-46
pparms command, 4-45
pwidth command, 4-45

Q

quit command, 4-53

R

re-execution of Startup RM, 2-1
readmodid command, 4-20
readregister? command, 4-49
resetting all devices, 2-1
resman.tbl file format, A-1
Resource Editor. *See* vxitedit.
Resource Manager. *See* Startup RM.
Resource Manager display
definition of, 3-1
files used by, 3-4
use of, 3-5
RM. *See* Startup RM.
rmentry? command, 4-48
routesignal command, 4-21
routevxiint command, 4-24
running the Startup RM, 2-6
running victext, 4-1
running vxitedit, 3-5

S

save command, vxitedit, 3-2
scriptoff command, 4-47
scripton command, 4-47
self-test management
Servant-side operation, Message-Based, 2-4
Servant-side operation, non-Message-Based, 2-5
VXI Resource Manager operation, 2-2
Servant-side. *See also* Commander/Servant hierarchies.
non-Logical Address 0 operation, 1-2, 2-4 to 2-5
Startup RM, 1-1, 2-1
Servant-side operation, Message-Based
address map configuration, 2-4
allocation of IRQ lines, 2-4
Commander-Servant hierarchies, 2-4
initiating normal operation, 2-4
self-test management, 2-4
VXI device identification, 2-4
Servant-side operation, non-Message-Based
address map configuration, 2-5
allocation of IRQ lines, 2-5
Commander/Servant hierarchies, 2-5

initiating normal operation, 2-5
self-test management, 2-5
VXI device identification, 2-5
Servant Word Serial Protocol commands
wssabort, 4-14
wssdisable, 4-13
wssenable, 4-13
wssrd, 4-13
wsswrt, 4-14
set command, 4-44
setdevinfo command, 4-4
setmodid command, 4-20
show command, vxitedit, 3-2
signaldeq command, 4-22
signalenq command, 4-22
signaljam command, 4-23
srctrig command, 4-28
Startup RM
configuration, 2-1
error-checking, 2-5
file access, 2-6
issuing commands, 2-1
Logical Address 0 (VXI Resource Manager) operation, 1-1 to 1-2, 2-2 to 2-3
non-Logical Address 0 operation, 1-2, 2-4 to 2-5
overview, 1-1
running, 2-6
Servant-side, 2-1
Servant-side application, 1-1
support for mainframe extenders, 2-1, 2-3
SYSRESET*, 2-1
Startup RM operation
Message-Based Servant-side operation, 2-4
non-Message-Based Servant-side operation, 2-5
overview, 2-1
re-execution of Startup RM, 2-1
resetting all devices, 2-1
VXI Resource Manager operation, 2-2 to 2-3
SYSRESET*, 2-1
system command, 4-53
System Configuration commands
createdevinfo, 4-5
finddevla, 4-2
getdevinfo, 4-3
setdevinfo, 4-4
System Interrupt Handler commands
assertsysreset, 4-39
disableacfail, 4-38
disablesoftreset, 4-38
disablesysfail, 4-37
disablesysreset, 4-40
enableacfail, 4-37
enablesoftreset, 4-38
enablesysfail, 4-37
enablesysreset, 4-39

T

Text Interactive Control Program, VXIbus. *See* victext.
Text Resource Editor. *See* vxitedit.
trigassertconfig command, 4-33
trigcfg.tbl file format, A-3
trigcntrconfig command, 4-34
trigextconfig command, 4-35
trigger configuration editor
definition of, 3-2
files used by, 3-4
use of, 3-13
trigtickconfig command, 4-36

U

unmaptrigtotrig command, 4-32
utilbus.tbl file format, A-3
utility bus configuration editor
definition of, 3-2
files used by, 3-4
use of, 3-14

V

version command, 4-53
victext
file access, 4-1
overview, 1-2
running, 4-1
using, 4-1
VXI device identification
Servant-side operation, Message-Based, 2-4
Servant-side operation, non-Message-Based, 2-5
VXI Resource Manager operation, 2-2
VXI Interrupt commands
acknowledgevxiint, 4-27
assertvxiint, 4-27
deassertvxiint, 4-27
disablevxiint, 4-26
disablevxitosignalint, 4-25
enablevxiint, 4-25
enablevxitosignalint, 4-24
routevxiint, 4-24
vxiintacknowledgemode, 4-26
VXI local hardware initialization program. *See* vxiinit.
VXI Resource Editor. *See* vxitedit.
VXI Resource Manager operation
address map configuration, 2-3
allocation of IRQ lines, 2-3
Commander/Servant hierarchies, 2-3
examination of non-VXI devices, 2-2
initiating normal operation, 2-3

multiple mainframe interrupt, trigger, and utility bus extension, 2-3
self-test management, 2-2
VXI device identification, 2-2
VXI Signal commands
disablesignalint, 4-22
enablesignalint, 4-22
routesignal, 4-21
signaldeq, 4-22
signalenq, 4-22
signaljam, 4-23
waitforsignal, 4-23
VXI Text Resource Editor. *See* vxitedit.
VXI Trigger commands
acknowledgetrig, 4-30
disabletrigsense, 4-29
enabletrigsense, 4-29
maptrigtotrig, 4-31
srctrig, 4-28
trigassertconfig, 4-33
trigcntrconfig, 4-34
trigextconfig, 4-35
trigtickconfig, 4-36
unmaptrigtotrig, 4-32
waitfortrig, 4-30
VXIbus Access Commands. *See* High-Level VXIbus Access Commands.
VXIbus Text Interactive Control Program. *See* victext.
vxiin command, 4-15
vxiinit, definition of, 1-1
vxiinlr command, 4-19
vxiinreg command, 4-17
vxiintacknowledgemode command, 4-26
vximove command, 4-18
vxiout command, 4-16
vxioutlr command, 4-20
vxioutreg command, 4-17
vxitedit
configuration editor, 3-1, 3-5
configuring VXI device characteristics, 2-1
device name editor, 3-1, 3-8
file access, 3-4
files used by, 3-4
interrupt configuration editor, 3-2, 3-11
introduction to, 3-1 to 3-2
manufacturer name editor, 3-1, 3-6
model name editor, 3-1, 3-7
non-VXI device editor, 3-1, 3-9
overview, 1-2
resource editors and displays in, 3-1 to 3-2
Resource Manager display, 3-1, 3-5
running, 3-5
summary of commands, 3-2
trigger configuration editor, 3-2, 3-13
utility bus configuration editor, 3-2, 3-14

W

waitforsignal command, 4-23
waitfortrig command, 4-30
writeregister command, 4-50
wsabort command, 4-6
wsclr command, 4-9
wscmd command, 4-8
wscmd? command, 4-8
wsecmd command, 4-11
wsgettmo command, 4-12
wslcmd command, 4-10
wslresp command, 4-11
wsrd command, 4-6
wsrdf command, 4-7
wsresp command, 4-9
wssabort command, 4-14
wssdisable command, 4-13
wssenable command, 4-13
wssettmo command, 4-11
wssrd command, 4-13
wsswrt command, 4-14
wstrg command, 4-9
wswrt command, 4-7
wswrtf command, 4-8