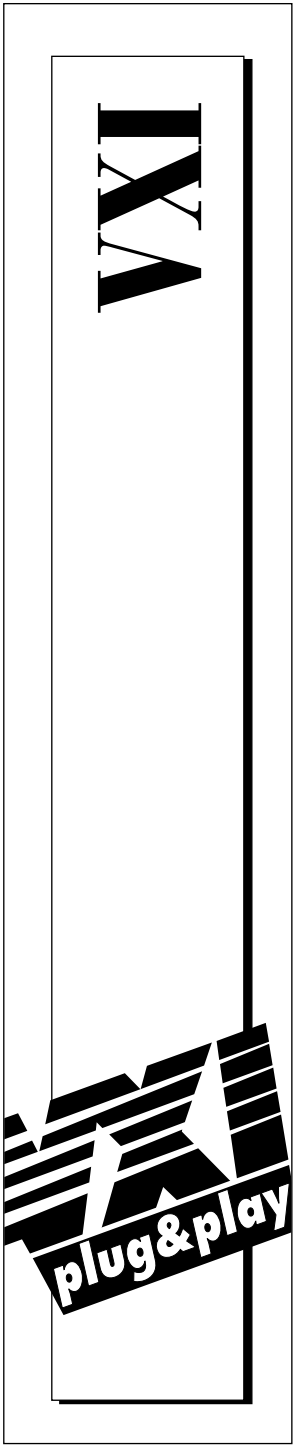


NI-VXI™ Graphical Utilities Reference Manual

June 1997 Edition
Part Number 371696A-01

Internet Support

support@natinst.com
E-mail: info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30,
Hong Kong 2645 3186, Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466,
Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70,
Switzerland 056 200 51 51, Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

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.

Table of Contents

About This Manual

Chapter 1
Introduction

Chapter 2
Startup Resource Manager

Chapter 3
VXI Resource Editor: VXIedit

Chapter 4
VXIbus Interactive Control (VIC) Program

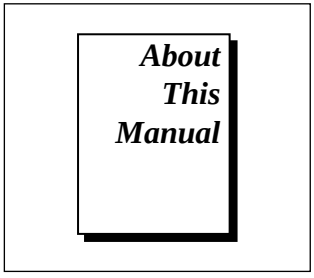
Customer Communication

Glossary

Index

Figures

Tables



This manual describes in detail the interactive graphical 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 graphical utilities that are available under the Microsoft operating systems (DOS, Windows 3.1, Windows 95, Windows NT) and the Solaris 2 operating system.
- Chapter 2, *Startup Resource Manager*, describes the operation and applications of the Startup Resource Manager.
- Chapter 3, *VXI Resource Editor: VXIedit*, describes `VXIedit`, the VXI Resource Editor program that you use to edit system and device information.
- Chapter 4, *VXIbus Interactive Control (VIC) Program*, introduces you to `VIC`, the VXIbus Interactive Control program, which you can use to communicate directly with VXI devices either through commands you enter at the keyboard or through the interactive utilities provided in `VIC`. This feature helps you learn how to communicate with devices, troubleshoot problems, and develop your application.
- The Appendix, *Customer Communication*, directs you where you can find forms you can use 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

The following conventions are used in this manual.

< >	Angle brackets enclose the name of a key on the keyboard (for example, <Enter>). Angle brackets also enclose parameter names.
[ ]	Square brackets enclose optional parameters (for example, `[la]`).
♦	The ♦ symbol indicates that the text following it applies only to a specific operating system.
bold	Bold text denotes the names of menus, menu items, dialog boxes, dialog box buttons or options, or windows.
bold italic	Bold italic text denotes a note, caution, or warning.
`bold monospace`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen.
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. The Glossary lists abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms.

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 VIC utilities 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 VIC whenever possible to generate the sequences of NI-VXI function calls that your application program will make. Trying your function calls in VIC 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:

- ANSI/IEEE Standard 1014-1987, *IEEE Standard for a Versatile Backplane Bus: VMEbus*
- ANSI/IEEE Standard 1155-1993, *IEEE VMEbus Extensions for Instrumentation: VXIbus*
- VXI-6, *VXIbus Mainframe Extender Specification*, Revision 1.0, VXIbus Consortium

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 graphical utilities that are available under the Microsoft operating systems (DOS, Windows 3.1, Windows 95, Windows NT) and the Solaris 2 operating system.

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

- VXI local hardware initialization program (`VXIinit`)
- Startup Resource Manager (RM, or `resman`)
- VXI Resource Editor (`VXIedit`)
- VXIbus Interactive Control Program (`VIC`)

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.

♦ You do not need to run `VXIinit` in Windows 95, Windows NT, or Solaris for the new MITE-based controllers, such as the PCI-MXI-2 or the VXIpc-850.

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 `VXIedit` 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 to write 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 map
- 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 Startup RM supports the following capabilities.

- Multiple-mainframe support using standard VXIbus mainframe extenders (see VXI document VXI-6, *Mainframe Extender Specification*)
- Support for dynamically configured devices on a per-mainframe basis (see Section F of the VXIbus specification)
- Integration of non-VXI (VME and pseudo-VXI) devices on a per-mainframe basis using the `VXIedit` 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 granted devices and configures its Commander/Servant hierarchies
- Initiates normal system operation for its Servants

When the Startup RM is complete, all required configurations have been performed. You can execute any application-specific program 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).

VXIedit

`VXIedit` 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.

`VXIedit` incorporates several resource editors and displays. Notice that there is no graphical version of `VXIedit` for the SB-MXI, or under Windows 3.1/95/NT for the AT-MXI-1 and VXIpc-500 Series controllers. A text-based interactive program `VXItedit` is available that is functionally equivalent to `VXIedit`.

VIC

You can use the `VIC` program to communicate interactively with the VXI devices in your system. `VIC` 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—SYSFAIL, ACFAIL, VXI interrupts, TTL/ECL triggers—is continuously monitored and displayed in all the panels. Hence, `VIC` also serves as a bus monitor for monitoring the state of the bus interface.

Notice that there is no graphical version of `VIC` for the SB-MXI, or under Windows 3.1/95/NT for the AT-MXI-1 and VXIpc-500 Series controllers. A text-based interactive program `VICtext` is available that is functionally equivalent to `VIC`.

Chapter 2

Startup Resource Manager

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

You can use the `VXIedit` utility to configure VXI device characteristics of the local CPU, such as its logical address. When the device is configured at Logical Address 0, the Startup RM interactively configures the VXI memory maps and devices to smoothly integrate fully 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 pre-configured 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 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 is automatically displayed for the integrator's inspection.

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 sub-state.
- It is a violation of the VXIbus specification to perform RM operations more than once. Devices are not required to respond to many of the configuration commands after the *Begin Normal Operation* command has been sent once.

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 as configured in `VXIedit`—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 VME 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 enables the address windows to detect devices 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 to each device a logical address by writing an appropriate value to the device's Logical Address register.

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 (that is, it has not asserted its PASSed bit in its Status register) within the specified time, the RM forces that device into the Safe state by setting the Sysfail Inhibit and Reset bits in the device's Control register. Any device forced into the Safe state 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 VXI 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,

the RM allocates a section of A24 or A32 address space 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 enables the appropriate address map windows on the mainframe extender so the whole system can access the memory space. Error 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 by the VXIbus specification of 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 `VXIedit` 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 `VXIedit` 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 start-up sequence is complete, the system enters the Normal Operation state, and the Startup RM terminates operation. You can execute any application after this point.

Message-Based Servant-Side Operation

This section describes 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 `VXIedit`.

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. The local CPU does not stop asserting SYSFAIL.

Address Map Configuration

You can use the `VXIedit` 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

The Startup RM automatically processes all Word Serial *Grant Device* commands it receives, and records information about the Servants assigned to it.

Allocation of IRQ Lines

The Startup RM automatically processes all VXI interrupt Word Serial commands (*Read Interrupters*, *Assign Handler Line*, and so on) it receives, and records the assignment information. You can use the `VXIedit` 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 message-based Servant devices.

Non-Message-Based Servant-Side Operation

This section describes 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 `VXIedit`.

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. The local CPU does not stop asserting SYSFAIL.

Address Map Configuration

You can use the `VXIedit` 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. You need to reserve interrupt lines by configuring the VXI RM (not the local CPU). Your application can then make function calls 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 under any of the following circumstances.

- The MXI-2 cable is not connected to the PCI-MXI-2 or AT-MXI-2 board (`VXIinit` will give a warning if it cannot locate the System Controller).
- 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

To run the Startup RM change the directory (or set the path) to the `NIVXI\` directory containing `resman.exe`. Enter *`resman`* at the prompt. Table 2-1 lists the command line options you can use in any of the Microsoft operating systems. Table 2-2 lists the command line options you can use in DOS only.

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
`-d`	Do not send BNO to message-based devices in a DC system
`-R`	Reset known extender devices only
`-t`	Read device type of LA 0 (otherwise assume message-based device)

Table 2-2. Startup RM Command Line Options (DOS Only)

Code	Description
`-a`	Automatic display adapter determination (default)
`-m`	Monochrome Display Adapter (MDA)
`-e`	Enhanced Graphics Adapter (EGA)
`-v`	Video Graphics Array (VGA)
`-i`	Multi-Color Graphics Adapter (MCGA)
`-g`	Color Graphics Adapter (CGA)
`-h`	Hercules Graphics Adapter
`-f`	AT&T Display or DBE Adapter
`-s`	Maximize screen refresh rate (for use with CGA only)
`-r`	Color mode
`-n`	Monochrome mode

Using the Startup RM

Using the Startup RM under DOS

You can use the Startup RM with or without a mouse. Without a mouse, use the <TAB> and <Shift-TAB> keys to jump from one field to the next, and the cursor keys to scroll within the window fields. Use the <RETURN> or <Enter> key to enter a selection.

Using the Startup RM under Windows 3.1/95/NT

You can scroll the display up or down using the scrollbars or the Page-Up/Page-Down keys. Unlike with the DOS Startup RM, you cannot directly access other NI-VXI applications. If you want to run either or these programs, quit the Startup RM and double-click on the appropriate icons in the NI-VXI folder.

Using the Startup RM under Solaris 2

The Startup RM for Solaris displays all information to the standard output. If you wish to scroll up to see prior information, you should run the RM in a scrollable window. Alternatively, you can run `resman` with the `-o` flag to output all configuration information into the file `NIVXIAPATH/tbl/resman.out`.

Chapter 3

VXI Resource Editor: VXIedit

This chapter describes VXIedit, the VXI Resource Editor program that you use to edit system and device information.

Introduction to VXIedit

VXIedit 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 VXIedit 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.

This chapter shows the interactive editors for the Windows 95 version only. The descriptions and instructions, however, are equally applicable for all versions of VXIedit.

☞ **Note:** ***There is no graphical version of*** VXIedit ***for the SB-MXI or under Windows 3.1/95/NT for the AT-MXI-1 and VXIpc-500 Series controllers.***

VXIedit incorporates the following resource editors and displays.

Resource Manager Display

Use this display to view the RM table in an easily understood, graphical manner. 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 Configuration Editor

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

Model Name Configuration 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 Configuration 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 Configuration 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 inter-mainframe interrupt configurations. Individual mainframe configuration is used to match programmable handlers/interrupters to static handlers/interrupters. Inter-mainframe interrupt configuration is used to define which interrupts will be imported from or exported to a particular mainframe.

Trigger Configuration Editor

Use this editor to define inter-mainframe trigger (TTL and/or ECL) configurations. Inter-mainframe trigger configuration is used to define which triggers will be imported from or exported to a particular mainframe.

Utility Bus Configuration Editor

Use this editor to define inter-mainframe configurations for the utility bus. Inter-mainframe utility bus configuration is used to define whether SYSRESET, SYSFAIL, and ACFAIL will be imported from or exported to a particular mainframe.

Running VXIedit

To run the DOS version of `VXIedit`, change the directory (or set the path) to the `NIVXI` directory containing `VXIEDIT.EXE`. Type `VXIEDIT` at the prompt. You can use a mouse with this program and select any option by double-clicking on any item.

To run the Windows 3.1/95/NT version of `VXIedit`, double-click on the **VXIedit** icon under the NI-VXI folder.

To run the Solaris 2 version of `VXIedit`, change the directory (or set the path) to the `NIVXI` directory. Type `VXIedit` at the prompt. You can use a mouse with this program and select any option by double-clicking on any item.

Using VXIedit

You can use `VXIedit` with or without a mouse. Without a mouse, use the <TAB> and <Shift-TAB> keys to jump from one field to the next, and the cursor keys to scroll within the window fields. Use the <Return> or <Enter> key to enter a selection, and the <Delete> and <Backspace> keys to delete text within a field.

Figure 3-1 shows an example of an opening screen of `VXIedit`. The screen shown in this example appears when used in a system containing a PCI-MXI-2.

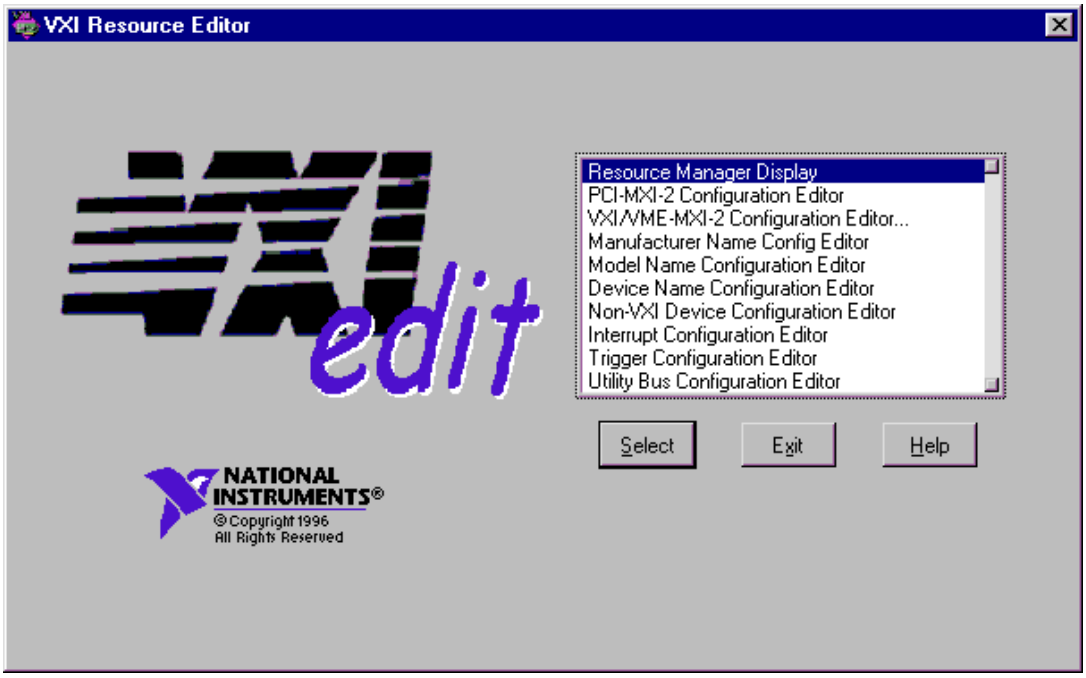


Figure 3-1. VXIedit Main Screen

Help

You can obtain help on any editor by selecting the **Help** button on the screen.

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, Servant names and logical addresses, memory requirements, and interrupt lines and handler assignments. Select any device entry in the scrollable box at the bottom of the screen to display RM information about that device.

Figure 3-2 shows a sample Resource Manager display.

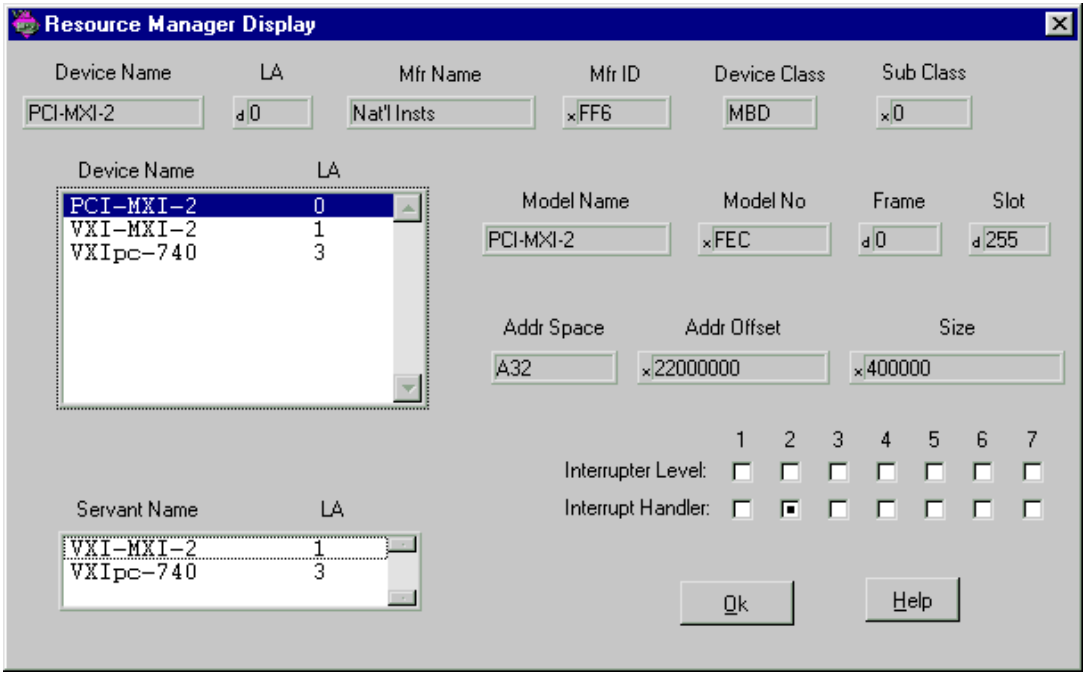


Figure 3-2. Resource Manager Display

Configuration Editor

The sample opening screen shown in Figure 3-1 lists two configuration editors called the **PCI-MXI-2 Configuration Editor** and the **VXI/VME-MXI-2 Configuration Editor**. These are examples of configuration editors that you use to change the VXI hardware interface configuration. For more information on the Configuration Editor, refer to the section on software installation and configuration in the getting started manual you received with your VXI hardware.

Manufacturer Name Configuration Editor

Use this editor to update the list of manufacturer names and ID numbers. To edit an existing entry, select it in the window. The entry will be copied into the edit fields at the top of the screen.

Figure 3-3 shows an example of this editor.

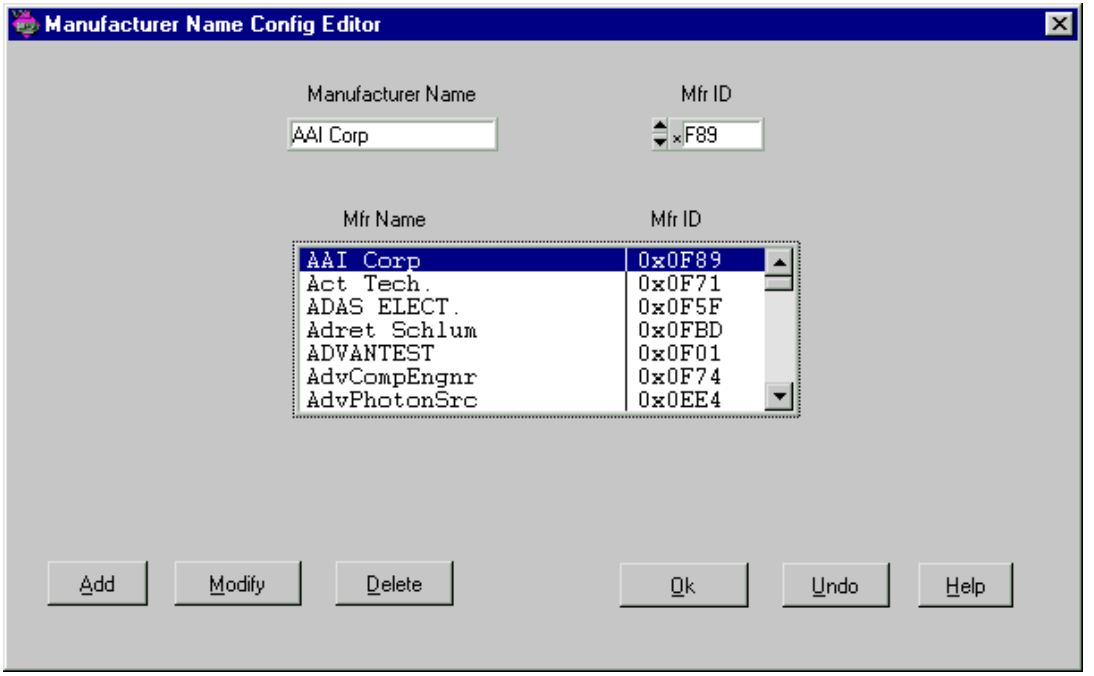


Figure 3-3. Manufacturer Name Configuration Editor

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Model Name Configuration Editor

Use this editor to update the list of model names, along with their associated manufacturer names and model numbers. To edit an existing entry, select it in the window. The entry will be copied into the edit fields at the top of the screen.

Figure 3-4 shows an example of this editor.

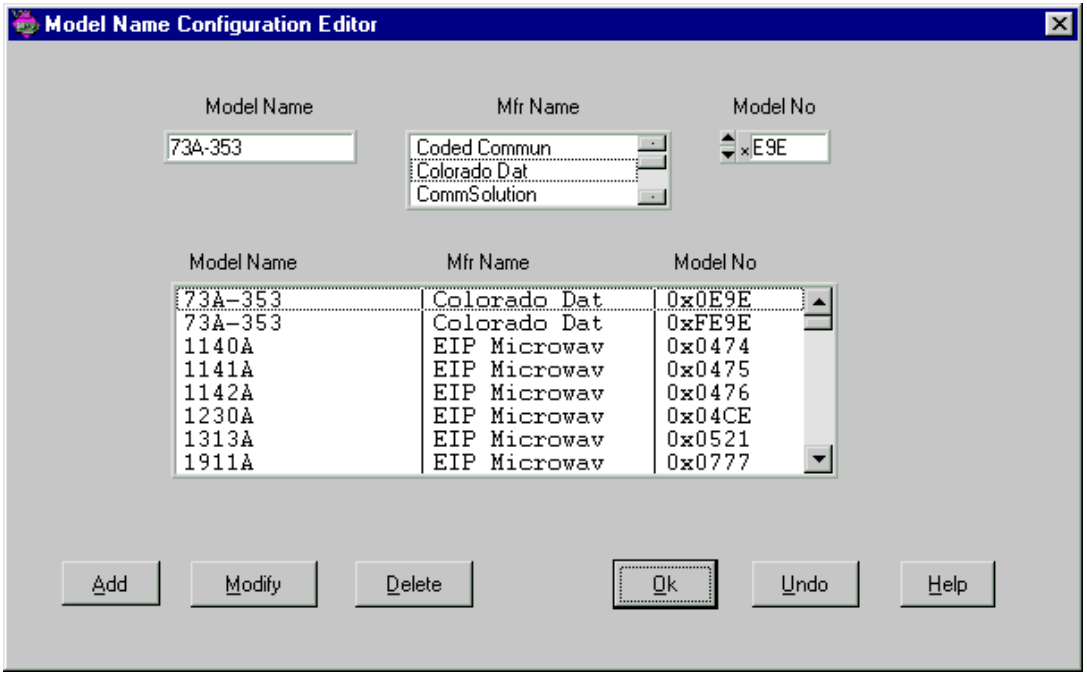


Figure 3-4. Model Name Configuration Editor

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Device Name Configuration Editor

Use this editor to assign a device name to each VXI device in the system. To edit an existing entry, select it in the window. The entry will be copied into the edit fields at the top of the screen.

Figure 3-5 shows an example of this editor.

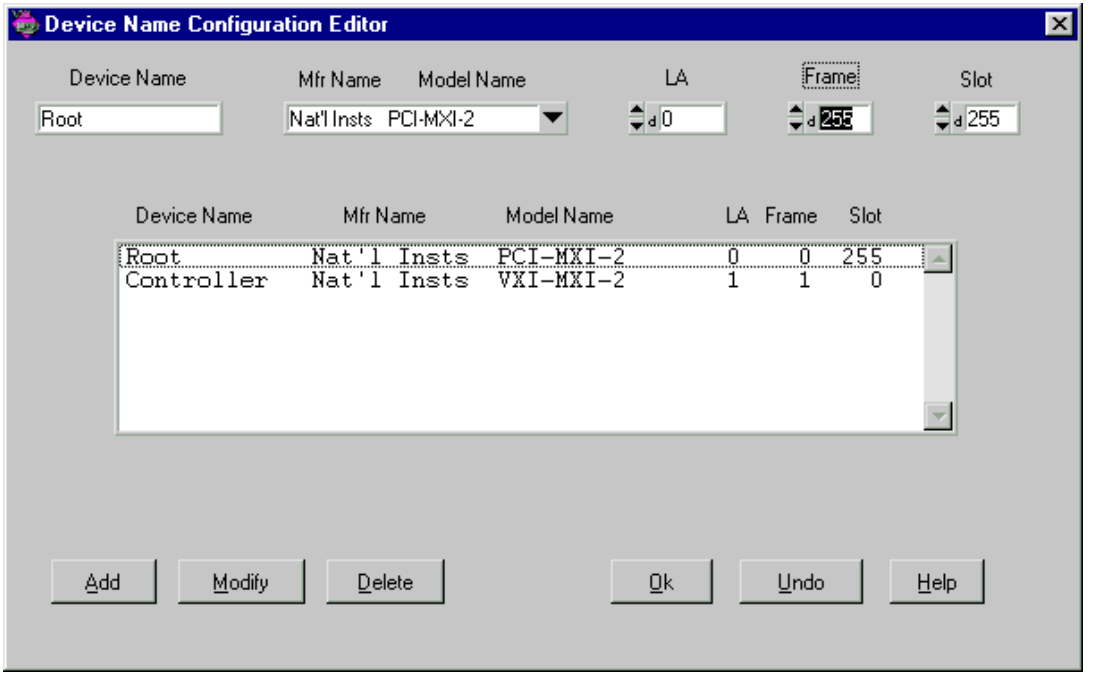


Figure 3-5. Device Name Configuration Editor

Normally, you can set the **Frame** and **Slot** fields to 255—255 corresponds to *Not Applicable*—because the Resource Manager can determine this information for VXI devices. However, if you have two or more of the same device, you should enter specific information for these fields and give each device a unique device name. The **Frame** field is the logical address of the remote controller (in either an external CPU or multiple-mainframe case) or the embedded controller (in an embedded CPU case) plugged into the same mainframe as the VXI device.

The example in Figure 3-5 shows a standard configuration using a PCI-MXI-2 and a VXI-MXI-2.

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Non-VXI Device Configuration Editor

The NI-VXI software uses this editor to get information about non-VXI devices. Figure 3-6 shows an example of this editor.

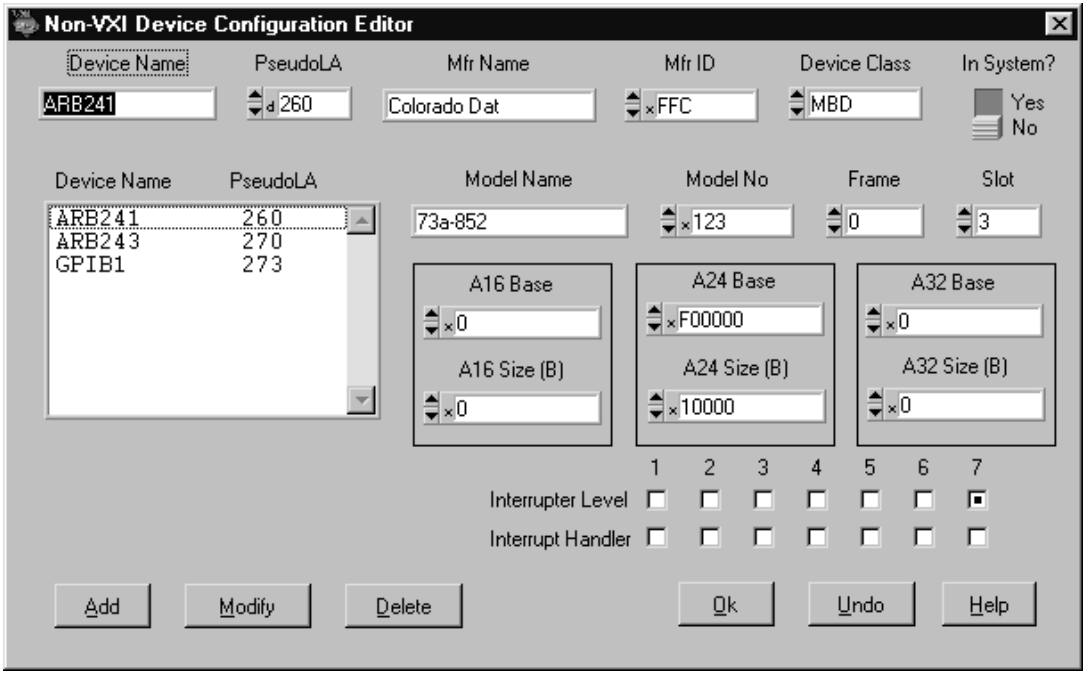


Figure 3-6. Non-VXI Device Configuration Editor

As you can see from observing the fields in Figure 3-6, we have assigned the memory device a *pseudo logical address* (between 256 and 511), as well as a manufacturer name and a model name. As a result, the device can be integrated with other VXI devices under the NI-VXI environment. 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. We have also noted that the device occupies memory in A24 address space. Remember to set the **Frame** field to the logical address of the parent-side extender (in either an external CPU or multiple-mainframe case) or the embedded controller (in an embedded CPU case) plugged into the same mainframe as the non-VXI device.

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Interrupt Configuration Editor

Use this editor to configure individual and inter-mainframe interrupts. Interrupts generated in the mainframe extender can be routed to other frames, and interrupts generated in other frames can be routed into its frame. You can use the external interrupt mapping table to define whether a particular interrupt is imported into or exported out of the frame. You can use the internal 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 they are programmable or static handlers. Figure 3-7 shows an example of this editor.

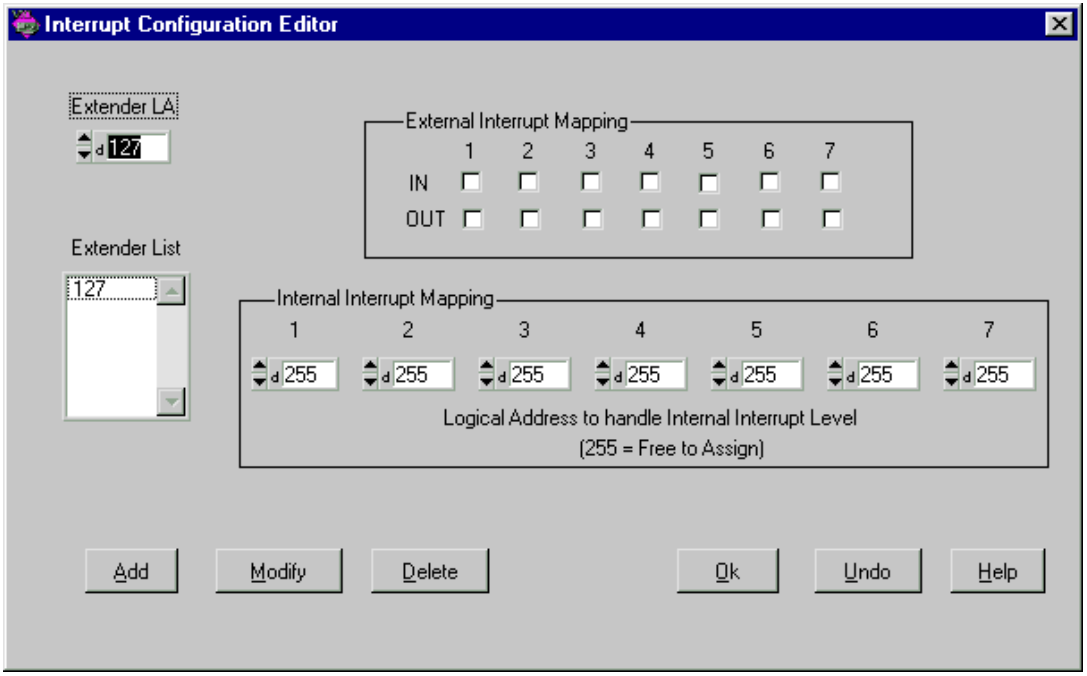


Figure 3-7. Interrupt Configuration Editor

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as

they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Trigger Configuration Editor

Use this editor to configure inter-mainframe triggers. Triggers sourced in the mainframe extender can be routed to other frames, and triggers sourced in other frames can be brought into its frame. You can use the external trigger mapping table to define whether a particular trigger is imported into or exported out of the frame.
Figure 3-8 shows an example of this editor.

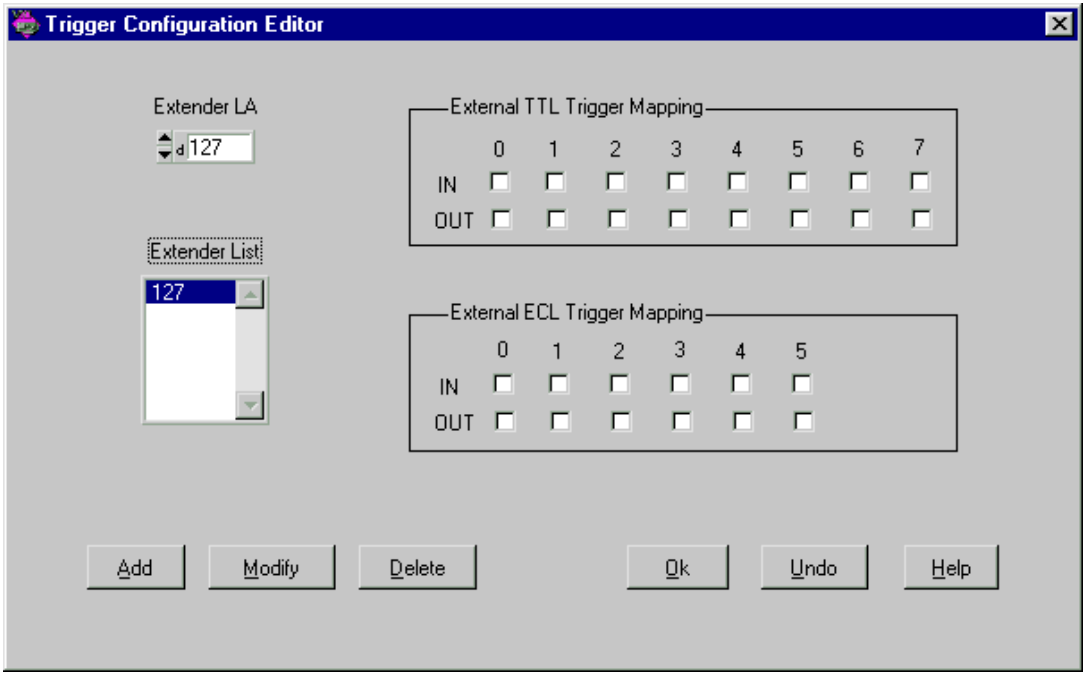


Figure 3-8. Trigger Configuration Editor

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Utility Bus Configuration Editor

Use this editor to configure the inter-mainframe utility bus. You can route SYSRESET, SYSFAIL and ACFAIL sourced in the mainframe extender to other frames, and you can route SYSRESET, SYSFAIL and ACFAIL sourced in other frames into the mainframe extender. Use the mapping table to define whether SYSRESET, SYSFAIL and ACFAIL are imported into or exported out of the frame (or both).

Figure 3-9 shows an example of this editor.

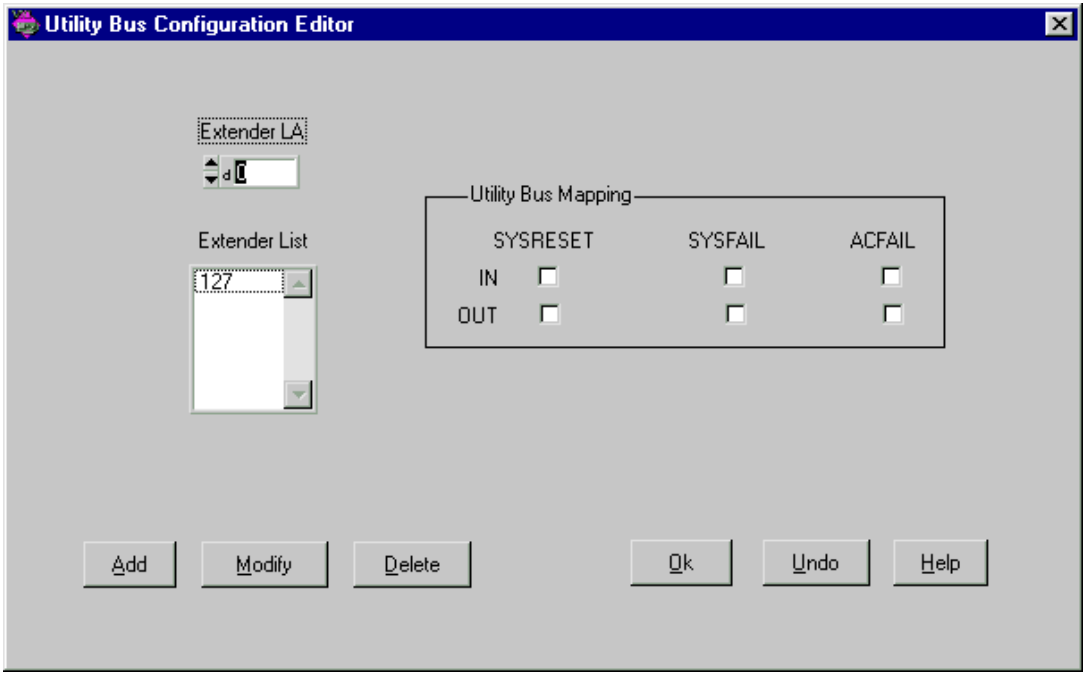


Figure 3-9. Utility Bus Configuration Editor

After editing the entry, select **Modify** to replace the original entry. Select **Add** to add it to the alphabetical list. To delete an entry, select the entry in the list and then select **Delete**. **Undo** reverts to the original settings as they were when you opened the screen. **OK** prompts you to save your changes and returns to the main screen.

Chapter 4

VXIbus Interactive Control (VIC) Program

This chapter introduces you to VIC, the VXIbus Interactive Control program, which you can use to communicate directly with VXI devices either through commands you enter at the keyboard or through the interactive controls provided in VIC. This feature helps you learn how to communicate with devices, troubleshoot problems, and develop your application.

You can use VIC with or without a mouse. When you use VIC without a mouse, use the <Tab> and <Shift-Tab> keys to jump from one field to the next, and the cursor keys to scroll the cursor within the window fields. Press <Return> or <Enter> to enter the selection. Press <Delete> or <Backspace> to delete text within a field.

☞ **Note:** ***You can right-click on any element in the interface for a pop-up help description of the element.***

Introduction

The VXI Interactive Control screen has seven tabs that you can use to communicate with VXI devices in various ways.

- **Text**—The Text Control tab is a text-oriented, interactive control program. Its command set includes commands that correspond to NI-VXI function calls, in addition to auxiliary commands that are unique to VIC. You can use this tab to interact with devices from the keyboard and to display information received from devices on the screen. After each function executes, VIC displays the device's response and a graphical representation of the command status.

The Text Control tab is designed to help you learn how to use the NI-VXI functions to program devices. After 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.

- **Word Serial**—The Word Serial tab is a graphical interface to the Word Serial Protocol functions. These functions include command and query sending as well as buffer sending and receiving using the Word Serial Protocol.
- **Bus Access**—The Bus Access tab is a graphical interface to the high-level VXIbus access functions. These functions include reading from and writing to the VXI address space.
- **Interrupts**—Through the Interrupts tab you can manually assert, deassert, and acknowledge interrupts.
- **Signals**—Through the Signals tab you can manually enable and disable signal interrupts; enqueue, dequeue, and jam signals in the signal queue; and wait for signals.

♦ In DOS only, the Interrupts and Signals tabs are combined into a single utility.

- **Triggers**—Through the Trigger tab you can assert and deassert TTL and ECL triggers using VXI-defined trigger protocols.
- **Log**—This is a configuration panel that you can use to enable logging in several different modes and choose a log file name.

DOS Users

Table 4-1 lists the command line options you can use in DOS.

Table 4-1. VIC Command Line Options for DOS Users

Code	Description
-a	Automatic display adapter determination (default)
-m	Monochrome Display Adapter (MDA)
-e	Enhanced Graphics Adapter (EGA)
-v	Video Graphics Array (VGA)
-i	Multi-Color Graphics Adapter (MCGA)
-g	Color Graphics Adapter (CGA)
-h	Hercules Graphics Adapter
-f	AT&T Display or DBE Adapter
-s	Maximize screen refresh rate (for use with CGA only)
-r	Color mode
-n	Monochrome mode

How to Use the User Interface Tabs

Figure 4-1 shows an example of a `VIC` panel. The user interface tabs are at the top of the panel.

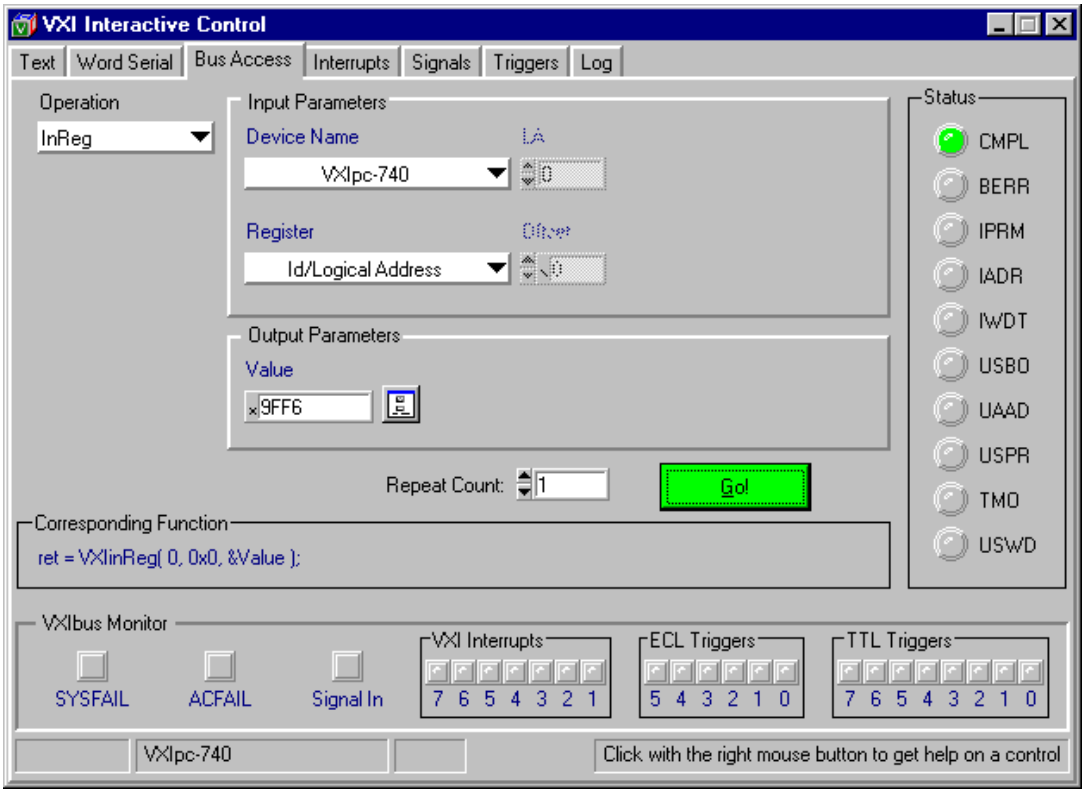


Figure 4-1. VXI User Interface

☞ **Note:** ***You can right-click on any element of the panels to see a pop-up help description of the element.***

- ***Not available in DOS.***

Except for the Text and Log tabs, each tab has the following general elements. These instructions apply to the Word Serial, Bus Access, Interrupts, Signals, and Triggers tabs.

Operation

Select a function from the **Operation** menu.

Input Parameters

Enter the input parameters required by the operation. The parameter options will change depending on which function you have chosen. Some operations, such as the Word Serial `Get Timeout` operation, do not have input parameters. For such functions, the **Input Parameters** group is hidden.

Output Parameters

This group displays the output parameters (if the operation has any). Otherwise, the **Output Parameters** group is hidden.

☞ **Note:** ***Certain parameters have a button to the right of the parameter field; click on the button to fill in more details for the parameter. The information you enter here is linked to the parameter field, and the values will automatically update to match the new information. For example, if you add information by clicking on the button, the value already entered in the parameter field may change to accommodate the new information.***

- ***In DOS, press <F1> to access this information.***

Repeat Count

Fill in a value in the **Repeat Count** box, indicating how many times to execute the run.

Go

Click on the green **Go!** button. At any time during the run, you can stop the operation by clicking the red **Stop** button, or pressing the <Escape> key. After command execution completes, `VIC` automatically fills in the **Output Parameters** with the results of the command. Again, these parameters will change depending on which function you have chosen.

Corresponding Function

This section of the panel shows the command syntax for the currently selected command. Right click to select **C Function** or **VIC Command**. You can copy and paste this text into the Command box of the Text Control tab, or just record the information for future use.

♦ In DOS, only C functions are displayed. In addition, you cannot use the copy/paste feature in DOS.

Status

The Status LEDs on the right-hand side of the panel will light green for OK and red for Error; right-click on each LED to see a pop-up help description of the button.

VXIbus Monitor

The VXIbus status (BERR, SYSFAIL, ACFAIL, VXI interrupts, TTL/ECL triggers) is monitored and displayed graphically in all of the interface tabs. This section is a visual monitor of the VXIbus.

- To detach this monitor from the panel, right-click anywhere in the area and choose **Detach**.
- Once detached, by default, the VXIbus Monitor window will stay on top of other windows on your screen. To move it to the back, right-click anywhere in the area and deselect **Always on Top**.
- To reattach the monitor to the main window, either right-click anywhere in the area and choose **Reattach**, or just close the window, and it will automatically be reattached where it belongs.
- To turn off the monitor, right-click anywhere in the area and choose **Disable**.

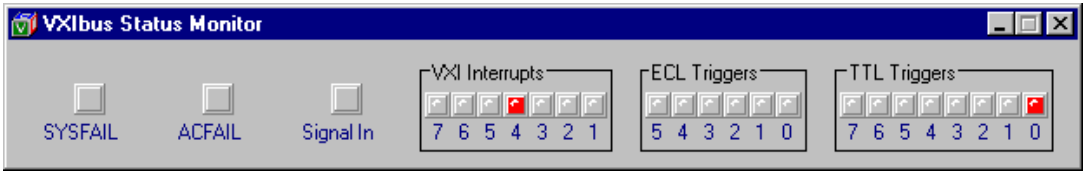


Figure 4-2. Detached VXIbus Status Monitor

♦ In DOS only, the VXIbus Monitor cannot be detached. To disable the bus monitor in DOS, use the `disablemon` command in the Text tab.

Status Bar

This is the horizontal bar at the very bottom of the panel. The first box indicates the status of the test, for example, it might say **`Running`**. The second box indicates the currently selected device, or the active device. The third box indicates the logging mode.

Figure 4-3. VIC Status Bar

Help

Online help is available for each interface tab. Click the right mouse button to get content sensitive help on any control. The **help** command gives online information about `VIC` commands with a quick reference for checking syntax and function of the VXI call. Type `help` followed by a command name to get more information about the particular command.

For an explanation of the status codes, refer to the corresponding function description in the *NI-VXI Programmer Reference Manual*.

Text Tab

The Text Control tab is a line-oriented, interactive control panel. Its command set includes commands that correspond to NI-VXI function calls, in addition to auxiliary commands that are unique to `VIC`. This tab is for experienced users who know the syntax of the commands they wish to enter. To interact with devices, enter commands by typing them on the computer keyboard. After each function executes, `VIC` displays the device's response in the history window and a graphical representation of the command status.

The Text Control tab is designed to help you learn how to use the NI-VXI functions to program devices. Once 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.

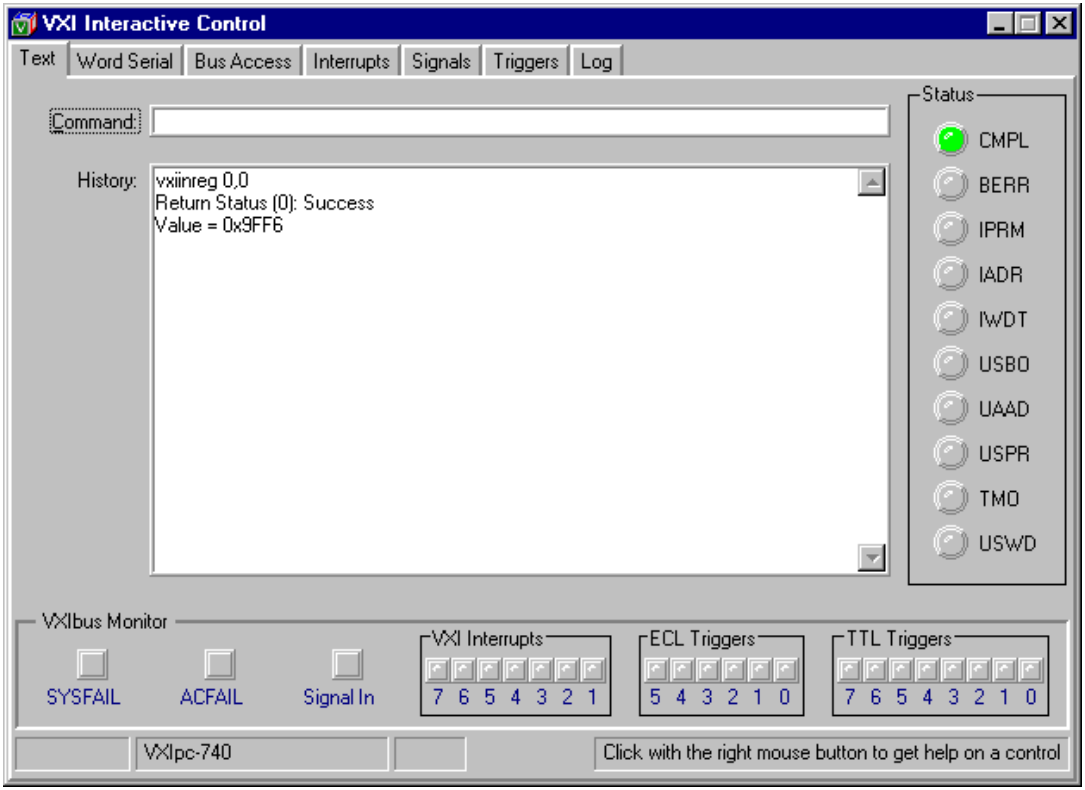


Figure 4-4. VIC Text Control Tab

Command syntax available in the Text screen are slightly different from NI-VXI function calls. Type `VIC` commands in the **Command** window. The **History** section lists the history of the commands and the corresponding outputs.

How to Use the Text Control Tab

☞ **Note:** ***You can You can right-click on any element of the panel to see a pop-up help description of the element.***

Type the command in the **Command** box and press <Enter>.

The **History** section, below the **Command** box, keeps a log of recently entered commands and command output for your information. You can cut and paste them into the command box as needed.

The user interface tabs can help you choose the commands you want without the need to know the exact syntax of the command. Command syntax in the Text Control tab is slightly different from NI-VXI function calls. For detailed information on specific commands, refer to the *Text Command Groups* section later in this chapter.

You can use the Up/Down buttons to scroll through the previously entered commands. The **Command** field supports command completion. Use <Tab> or <Shift-Tab> to cycle through possible commands that begin with the prefix you enter.

Word Serial Tab

The Word Serial tab is a graphical user interface panel that executes the following Word Serial commands. These functions use the Word Serial Protocol and include command and query sending as well as buffer sending and receiving.

Through this tab you can interactively execute the Commander Word Serial Protocol commands. The following section describes the Word Serial commands and their available options.

☞ **Note:** ***You can right-click on any element of the panel to see a pop-up help description of the element.***

For further instructions, review the *How to Use the User Interface Tabs* section, previously in this chapter.

For an explanation of the status codes, refer to the corresponding function description in the *NI-VXI Programmer Reference Manual*.

Word Serial Commands

Write

Purpose

Transfers the specified string to a message-based Servant using the VXIbus Byte Transfer Protocol.

Select the logical address, mode, the string to write, and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wswrt` function description for more detailed information.

Read

Purpose

Transfers data from a message-based device to the console using the VXIbus Byte Transfer Protocol.

Select the logical address, mode, count, and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wsrd` function description for more detailed information.

Cmd

Purpose

Sends a Word Serial command to a message-based Servant device.

Select the logical address, command, and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wscmd` function description for more detailed information.

Query

Purpose

Sends a Word Serial command to a message-based Servant device and returns the response.

Select the logical address, command, and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wscmd?` function description for more detailed information.

Resp

Purpose

Gets the response for the previously sent Word Serial query.

Select the logical address and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wsresp` function description for more detailed information.

Trg

Purpose

Sends the Word Serial *Trigger* command to a Servant device.

Select the logical address and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wstrg` function description for more detailed information.

Clr

Purpose

Sends the Word Serial *Clear* command to a Servant device.

Select the logical address and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wsclr` function description for more detailed information.

SetTmo

Purpose

Sets the timeout period for all the Word Serial functions.

Select the timeout period and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `wssettmo` function description for more detailed information.

GetTmo

Purpose

Gets the timeout period set for the Word Serial functions.

Select **Go!** to execute the command.

See the `wsgettmo` function description for more detailed information.

Bus Access Tab

Through this graphical user interface tab you can interactively execute the high-level VXIbus access functions. The following section describes the commands in the Bus Access tab.

☞ **Note:** ***You can right-click on any element of the panel to see a pop-up help description of the element.***

For further instructions, review the *How to Use the User Interface Tabs* section, previously in this chapter.

For an explanation of the status codes, refer to the corresponding function description in the *NI-VXI Programmer Reference Manual*.

Bus Access Commands

InReg

Purpose

Reads a single word from the device specified by the logical address at the specified offset. To interpret the value, click the button located to the right of the output value.

- In DOS, press <F1> to interpret the value.

Select the logical address, register offset, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `vxiinreg` function description for more detailed information.

OutReg

Purpose

Writes a single word to the device specified by the logical address at the specified offset. To compose the value, click the button to the right of the output value.

- In DOS, press <F1> to compose the value.

Select the logical address, register offset, value, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `vxioutreg` function description for more detailed information.

In

Purpose

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

Select the access parameters, width, address, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `vxiin` function description for more detailed information.

Out

Purpose

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

Select the access parameters, width, address, repeat number (number of times to execute the function), and the value, and then select **Go!** to execute the command.

See the `vxiout` function description for more detailed information.

Move

Purpose

Moves a block of bytes, words, or longwords from a source location in any address space to a destination in any other address space.

Select the source space, source address, destination space, destination address, count, width, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `vximove` function description for more detailed information.

Interrupts Tab

The Interrupts tab is a graphical user interface panel that you can use to manually assert, deassert, and acknowledge interrupts. Through this tab you can execute the VXI Interrupt functions. The following section describes the commands in the Interrupts tab.

☞ **Note:** ***You can right-click on any element of the panel to see a pop-up help description of the element.***

- In DOS, the Interrupts and Signals panels are combined.

  For further instructions, review the *How to Use the User Interface Tabs* section, previously in this chapter.

  For an explanation of the status codes, refer to the corresponding function description in the *NI-VXI Programmer Reference Manual*.

Interrupt Commands

Ack

Purpose

Acknowledges the VXI interrupt.

Select the controller, the VXI interrupt, and repeat number, and then select **Go!** to execute the command.

See the `acknowledgevxiint` function description for more detailed information.

Assert

Purpose

Asserts the VXI interrupt.

Select the controller, the status/ID, the VXI interrupt, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `assertvxiint` function description for more detailed information.

Deassert

Purpose

Deasserts the VXI interrupt.

Select the controller, the VXI interrupt, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `deassertvxiint` function description for more detailed information.

Signals Tab

The Signals tab is a graphical user interface panel that you can use to enable and disable signal interrupts; enqueue, dequeue, and jam signals in the signal queue; and wait for signals.

Through this tab you can interactively execute the VXI Signal functions. The following paragraphs describe the commands in the Signals tab.

☞ **Note:** ***You can right-click on any element of the panel to see a pop-up help description of the element.***

- In DOS, the Interrupts and Signals panels are combined.

  For further instructions, review the *How to Use the User Interface Tabs* section, previously in this chapter.

  For an explanation of the status codes, refer to the corresponding function description in the *NI-VXI Programmer Reference Manual*.

Signal Commands

Enable

Purpose

Sensitizes VIC to all received signals.

Select the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the enablesignalint function description for more detailed information.

Disable

Purpose

Desensitizes VIC to all received signals.

Select the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the disablesignalint function description for more detailed information.

Deq

Purpose

Dequeues a signal specified by the signalmask for the specified logical address from the signal queue.

Enter the logical address, the signalmask, and repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the signaldeq function description for more detailed information.

Enq

Purpose

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

Enter the signal and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `signalenq` function description for more detailed information.

Jam

Purpose

Puts the specified signal on the head of the signal queue for the logical address.

Enter the signal and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `signaljam` function description for more detailed information.

Wait

Purpose

Waits for a particular signal from a particular device.

Enter the logical address, the signalmask, and the repeat number (number of times to execute the function), and then select **Go!** to execute the command.

See the `waitforsignal` function description for more detailed information.

Triggers Tab

The Triggers tab is a graphical, user interface panel that you can use to assert and deassert TTL and ECL triggers using VXI-defined trigger protocols.

Through this tab you can interactively execute the VXI trigger function `SrcTrig`.

☞ **Note:** ***You can right-click on any element of the panel to see a pop-up help description of the element.***

For further instructions, review the *How to Use the User Interface Tabs* section, previously in this chapter.

For an explanation of the status codes, refer to the corresponding function description in the *NI-VXI Programmer Reference Manual*.

Trigger Commands

SrcTrg

Purpose

Sources a trigger on a trigger line using the specified protocol.

Select the controller, the trigger line, the trigger protocol, the timeout period, and the repeat number, and then select **Go!** to execute the command.

See the `srctrig` function description for more detailed information.

Log Tab

This is a configuration panel where you can save your actions to whatever file you specify.

- In DOS, the Log tab is not available. Instead, you can use the `SCRIPTON/OFF` text commands.

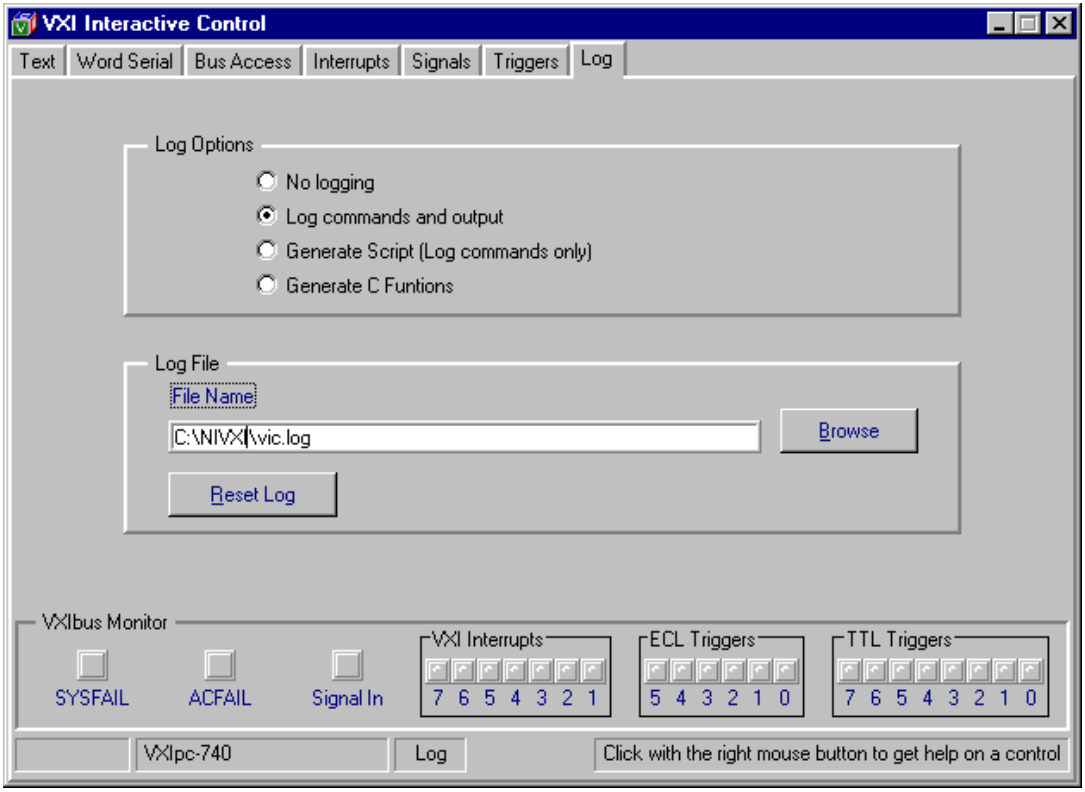


Figure 4-5. Log Tab Panel

☞ **Note:** ***You can right-click on any element of the panel to see a pop-up help description of the element.***

Log Options

In the **Log Options** section, click on the option you want to use.

No logging	Disables all logging.
Log commands and output	Logs all `VIC` interactions. Records both `VIC` commands and command outputs. For graphical tabs, records the corresponding `VIC` command.
Generate Script	Logs commands only. This creates a `VIC` script that you can execute using the "$" command.
Generate C Functions	Records NI-VXI C functions corresponding to user actions. You can insert the generated log file into your program.

Log File

Enter the path for the **Log File**, or click on the **Browse** button to select the path interactively. Click on the **Reset Log** button to clear the contents of the file.

Text Command Groups

Notice that angle brackets (< >) enclose parameter names in the following descriptions (for example, <filename>). In addition, optional parameters have square brackets (for example, [,<count>]).

- System Configuration Commands
- Commander Word Serial Commands
- Servant Word Serial Commands
- High Level Bus Access Commands
- Local Resources Access Commands
- VXI Signal Commands
- VXI Interrupts Commands
- VXI Triggers Commands
- System Interrupts Commands
- Bus Extender Commands
- Auxiliary Commands

☞ **Note:** ***The*** <la> ***parameter is optional only if you use the*** set ***command to select a device with which to communicate.***

System Configuration Commands

VIC 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>]]]]]]
```

Parameters

Name	Description
namepat	Name pattern
manid	VXI manufacturer identification number
modelcode	Manufacturer's 12-bit model number
devclass	Device class of the device
slot	Slot location of the device
mainframe	Mainframe location of device (logical address of extender)
cmdrla	Commander's logical address

The next table defines values for the <devclass> parameter.

Option	Description
0	Memory class device
1	Extended class device
2	Message-based device
3	Register-based device

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 specified information about a device in the NI-VXI RM table.

Command Syntax

```
getdevinfo [<la>, ]<field>
```

Parameters

Name	Description
la	Logical address of device to get information about
field	Field identification number

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

The next table defines values for the <field> parameter.

Value	Field
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 (see the following table for bit definitions)
18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

The following table defines possible bit values for field number 17, which describes extender/controller information.

Bit	Extender/Controller Information	
15 to 13	Reserved	
12	0	Parent-side extender
	1	Child-side extender
11	0	Not frame extender
	1	Frame extender
10	1	Remote controller
9	1	Local controller
8	1	External controller
7 to 0	Frame extender towards root frame	

Example

Get the name of a device at Logical Address 4.

```
getdevinfo 4, 1
```

setdevinfo

Purpose

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

Command Syntax

```
setdevinfo [<la>, ]<field>, <fieldvalue>
```

Parameters

Name	Description
la	Logical address of device in RM table for which to set information
field	Field identification number
fieldvalue	Appropriate value to set for the specified field

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

The next table defines values for the <field> parameter.

Value	Field
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 (see the following table for bit definitions)
18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

The following table defines possible bit values for field number 17, which describes extender/controller information.

Bit	Extender/controller Information	
15 to 13	Reserved	
12	0	Parent-side extender
	1	Child-side extender
11	0	Not frame extender
	1	Frame extender
10	1	Remote controller
9	1	Local controller
8	1	External controller
7 to 0	Frame extender towards root frame	

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>
```

Parameters

Name	Description
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

VIC supports the following Commander Word Serial Protocol commands.

wsrd

Purpose

Reads data from a message-based device using the VXIbus Byte Transfer Protocol and displays it on the console.

Command Syntax

```
wsrd [<la>, ]<count>, <mode>
```

Parameters

Name	Description
la	Logical address of message-based Servant
count	Maximum number of bytes to transfer
mode	Transfer mode bit vector

☞ **Note:** ***When using device-level commands, do not specify the*** la. ***The*** la ***of the active device will be used.***

The next table lists bits in the <mode> parameter.

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 bit
2	LF character termination	
	0	Do not terminate transfer on LF bit
	1	Terminate transfer on LF bit
3	CR character termination	
	0	Do not terminate transfer on CR bit
	1	Terminate transfer on CR bit
4	EOS character termination	
	0	Do not terminate transfer on EOS bit
	1	Terminate transfer on EOS bit
8 to 15	EOS character (valid if EOS termination)	

Example

Read and display 10 bytes from the device at Logical Address 5 (mode = poll until DIR; 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>
```

Parameters

Name	Description
la	Logical address of message-based Servant
filename	File name to which to transfer data
count	Maximum number of bytes to transfer
mode	Transfer mode bit vector (same as in wsrd)

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

Read 10 bytes from a device at Logical Address 5 (mode = poll till DIR, 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>]
```

Parameters

Name	Description
la	Logical address of message-based Servant
string	Buffer to send
mode	Transfer mode bit vector

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

The next table lists bits in the `<mode>` parameter.

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>]
```

Parameters

Name	Description
la	Logical address of message-based Servant
filename	Name of file from which to transfer data
count	Maximum number of bytes to transfer

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>
```

Parameters

Name	Description
la	Logical address of message-based Servant
command	Word Serial command

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>
```

Parameters

Name	Description
la	Logical address of message-based Servant
command	Word Serial query

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>]
```

Parameters

Name	Description
`la`	Logical address of message-based Servant

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

Get the response.

```
wsresp 5
```

wstrg

Purpose

Sends the Word Serial *Trigger* command to a message-based Servant device.

Command Syntax

```
wstrg [<la>]
```

Parameters

Name	Description
la	Logical address of message-based Servant

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

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

```
wstrg 5
```

wsclr

Purpose

Sends the Word Serial *Clear* command to a message-based Servant device.

Command Syntax

```
wsclr [<la>]
```

Parameters

Name	Description
la	Logical address of message-based Servant

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

Send the Word Serial *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>
```

Parameters

Name	Description
la	Logical address of message-based Servant
abortop	Option for aborting Word Serial operation

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

The next table lists the options for the `<abortop>` parameter.

Option	Description
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, 2, 3, and 4, where the operation is not aborted until the response is read.***

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>
```

Parameters

Name	Description
la	Logical address of message-based Servant
command	Longword Serial command
respflag	Response flag

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

The next table lists the options for the `<respflag>` parameter.

Option	Description
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 Serial query.

Command Syntax

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

Parameters

Name	Description
la	Logical address of message-based Servant

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

Get the response for the previously sent Longword Serial 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>
```

Parameters

Name	Description
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

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

The next table lists the options for the `<respflag>` parameter.

Option	Description
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>
```

Parameters

Name	Description
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
```

Parameters

None

Example

Get the timeout value.

```
wsgettmo
```

Servant Word Serial Protocol Commands

VIC supports the following Servant Word Serial Protocol commands.

wssenable

Purpose

Enables all Servant Word Serial Protocol commands.

Command Syntax

```
wssenable
```

Parameters

None

Example

Enable all Servant Word Serial Protocol commands.

```
wssenable
```

wssdisable

Purpose

Inhibits all Servant Word Serial Protocol commands from being received.

Command Syntax

```
wssdisable
```

Parameters

None

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>
```

Parameters

Name	Description
count	Maximum number of bytes to receive
mode	Transfer mode bit vector

The next table defines bit 0 in the <mode> parameter.

Bit 0 Value	Description
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>
```

Parameters

Name	Description
string	Buffer to send
mode	Transfer mode bit vector

The next table lists bits in the <mode> parameter.

Bit	Description	
0	Not DOR	
	0	Do not send DOR signal to Commander
	1	Send DOR signal to Commander
1	END bit termination	
	0	Do not send END with the last byte
	1	Set 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>
```

Parameters

Name	Description
abortop	Option for aborting Word Serial operation

The next table lists the meaning of bits in the <abortop> parameter.

Bit	Description
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 Access Commands

VIC 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>
```

Parameters

Name	Description
accessparms	Access parameters bit vector
address	VXI address
width	Data width

The next table lists bits in the <accessparms> parameter.

Bit	Description	
0 to 1	VXI address space	
	1	A16
	2	A24
	3	A32
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
5 to 6	0	Reserved
7	Byte order	
	0	Motorola
	1	Intel
8 to 15	0	Reserved

The next table lists the options for the <width> parameter.

Option	Description
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>
```

Parameters

Name	Description
accessparms	Access parameters bit vector
address	VXI address
width	Data width
value	Value to write

The next table lists bits in the <accessparms> parameter.

Bit	Description	
0 to 1	VXI address space	
	1	A16
	2	A24
	3	A32
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
5 to 6	0	Reserved
7	Byte order	
	0	Motorola
	1	Intel
8 to 15	0	Reserved

The next table lists the options for the <width> parameter.

Option	Description
1	Byte
2	Word
4	Longword

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>
```

Parameters

Name	Description
la	Logical address
reg	Offset within VXI Logical Address registers

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>
```

Parameters

Name	Description
la	Logical address
reg	Offset within VXI Logical Address registers
value	Value to write

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>
```

Parameters

Name	Description
`srcparms` and `destparms`	Source and destination access parameters bit vector
`srcaddr` and `destaddr`	Address (offsets within source and destination address spaces)
`length`	Number of elements to transfer
`width`	Data transfer width

The next table lists bits in the <srcparms> and <destparms> parameters.

Bit	Description	
0 to 1	Source and destination address space **Note:** ***You cannot use Local address space in*** VIC.	
	1	A16
	2	A24
	3	A32
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
	6	Nonprivileged VME64 block access
	7	Supervisory VME64 block access
5 to 6	0	Reserved
7	Byte order	
	0	Motorola
	1	Intel
8 to 10	0	Reserved
11	Use programmed I/O	
12	Do not increment address (FIFO mode)	
13	Disable block accesses	
14	Disable SYNC-MXI protocol (applies only to PCI-MXI-2 and AT-MXI-2)	
15	0	Reserved

The next table lists the options for the <width> parameter.

Option	Description
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

VIC supports the following local resource access commands.

getmyla

Purpose

Gets the logical address of the local VXI device.

Command Syntax

```
getmyla
```

Parameters

None

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>
```

Parameters

Name	Description
reg	Offset within VXI Logical Address registers
width	Data transfer width

The next table lists the options for the <width> parameter.

Option	Description
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>
```

Parameters

Name	Description
reg	Offset within VXI Logical Address registers
width	Data transfer width
value	Value to write

The next table lists the options for the <width> parameter.

Option	Description
1	Byte
2	Word
4	Longword

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>
```

Parameters

Name	Description
enable	Action to perform on MODID enable bit
modid	Bit vector for Bits 0 to 12, corresponding to Slots 0 to 12

The next table lists the options for the <enable> parameter.

Option	Description
0	Clear MODID enable bit.
1	Set MODID enable bit.

Example

Set all the MODID lines.

```
setmodid 1, 0xffff
```

readmodid

Purpose

Reads the MODID lines of the VXIbus backplane.

Command Syntax

```
readmodid
```

Parameters

None

Example

Read the MODID lines.

```
readmodid
```

VXI Signal Commands

VIC supports the following VXI signal commands.

routesignal

Purpose

Selects whether each type of signal is enqueued on a software queue or handled by the signal handler.

Command Syntax

```
routesignal [<la>, ]<modemask>
```

Parameters

Name	Description
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.

☞ **Note:** ***When using device-level commands, do not specify the*** la. ***The*** la ***of the active device will be used.***

The following two tables define bits in the <modemask> parameter 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)

The following table defines bits in the <modemask> parameter 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 `VIC` to receive all signals.

Command Syntax

```
enablesignalint
```

Parameters

None

disablesignalint

Purpose

Desensitizes `VIC` to not receive signals.

Command Syntax

```
disablesignalint
```

Parameters

None

signaldeq

Purpose

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

Command Syntax

```
signaldeq [<la>, ]<signalmask>
```

Parameters

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

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>
```

Parameters

Name	Description
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>
```

Parameters

Name	Description
signal	Signal to jam on the front of the queue

Example

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

```
signaljam 0x1111
```

waitforsignal

Purpose

Waits for one of the specified signals to be received.

Command Syntax

```
waitforsignal [<la>, ]<signalmask>, <timeout>
```

Parameters

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

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

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

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

VXI Interrupt Commands

VIC 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>
```

Parameters

Name	Description
controller	Controller for which to configure interrupt routing. -1 = Local controller or the first remote controller
sroute	Bit vector of VXI interrupt levels to route. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

The following table describes the options for the <sroute> bit vector.

Option	Description
0	Handle as a VXI interrupt for appropriate level.
1	Handle as a VXI signal 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 `VIC` to the specified VXI interrupt levels being processed as VXI signals for the specified controller.

Command Syntax

```
enablevxitosignalint <controller>, <levels>
```

Parameters

Name	Description
`controller`	Controller for which to enable VXI interrupt. -1 = Local controller or the first remote controller
`levels`	Bit vector of VXI interrupt levels to enable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

The following table describes the options for the `<levels>` bit vector.

Option	Description
0	Leave at current setting.
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 VIC to the specified VXI interrupt levels being processed as VXI signals for the specified controller.

Command Syntax

```
disablevxitosignalint <controller>, <levels>
```

Parameters

Name	Description
controller	Controller for which to disable VXI interrupt. -1 = Local controller or the first remote controller
levels	Bit vector of VXI interrupt levels to disable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

The following table describes the options for the <levels> bit vector.

Option	Description
0	Leave at current setting.
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 `VIC` 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>
```

Parameters

Name	Description
`controller`	Controller for which to enable VXI interrupt. -1 = Local controller or the first remote controller
`levels`	Bit vector of VXI interrupt levels to enable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

The following table describes the options for the `<levels>` bit vector.

Option	Description
0	Leave at current setting.
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 `VIC` 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>
```

Parameters

Name	Description
`controller`	Controller for which to disable VXI interrupt. -1 = Local controller or the first remote controller
`levels`	Bit vector of VXI interrupt levels to disable. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

The following table describes the options for the `<levels>` bit vector.

Option	Description
0	Leave at current setting.
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>
```

Parameters

Name	Description
controller	Controller to configure. -1 = Local controller or the first remote controller
modes	Bit vector of VXI interrupt levels to set to ROAK or RORA

The following table describes the options for the <modes> bit vector.

Option	Description
0	Set to ROAK VXI interrupt.
1	Set to RORA 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>
```

Parameters

Name	Description
controller	Controller for which to assert VXI interrupt. -1 = Local controller or the first remote 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>
```

Parameters

Name	Description
controller	Controller for which to deassert VXI interrupt. -1 = Local controller or the first remote 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>
```

Parameters

Name	Description
controller	Controller for which to acknowledge VXI interrupt. -1 = Local controller or the first remote controller
level	Interrupt level

Example

Acknowledge interrupt 4 in Controller 1.

```
acknowledgevxiint 1, 4
```

VXI Trigger Commands

The VIC 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>
```

Parameters

Name	Description
controller	Controller on which to source trigger line. -1 = Local controller or the first remote controller
line	Trigger line to source
protocol	Trigger protocol
timeout	Timeout value in milliseconds

The following table defines possible values for the <line> parameter.

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

The following table defines possible values for the <protocol> parameter.

Value	Protocol
0	ON
1	OFF
2	START
3	STOP
4	SYNC
5	SEMI-SYNC
6	ASYNC
7	SEMI-SYNC (and wait for acknowledge)
8	ASYNC (and wait for acknowledge)
ffffh	Abort previous acknowledge pending (5 and 6)

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>
```

Parameters

Name	Description
controller	Controller on which to enable sensing of trigger line. -1 = Local controller or the first remote controller
line	Trigger line to enable sensing
protocol	Trigger protocol

The following table defines possible values for the <line> parameter.

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

The following table defines possible values for the <protocol> parameter.

Value	Protocol
0	ON
1	OFF
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>
```

Parameters

Name	Description
`controller`	Controller on which to disable sensing of trigger line. -1 = Local controller or the first remote controller
`line`	Trigger line to enable sensing

The following table defines possible values for the `<line>` parameter.

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 trigger line to be encountered on the specified controller.

Command Syntax

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

Parameters

Name	Description
controller	Controller waiting for trigger line to be encountered. -1 = Local controller or the first remote controller
line	Trigger line to wait on
timeout	Timeout value in milliseconds

The following table defines possible values for the <line> parameter.

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

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>
```

Parameters

Name	Description
controller	Controller on which to acknowledge trigger interrupt. -1 = Local controller or the first remote controller
line	Trigger line to acknowledge

The following table defines possible values for the <line> parameter.

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>
```

Parameters

Name	Description
controller	Controller on which to map signal lines. -1 = Local controller or the first remote controller
srctrig	Source line to map to destination
desttrig	Destination line to map from source
mode	Signal conditioning mode. 0 = no conditioning

The following table defines possible values for the <srctrig> and <desttrig> parameters.

Value	Source/Destination Line
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

The following table defines bit values for the <mode> parameter.

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 are 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>
```

Parameters

Name	Description
`controller`	Controller on which to unmap signal lines. -1 = Local controller or the first remote controller
`srctrig`	Source line to unmap from destination
`desttrig`	Destination line mapped from source

The following table defines possible values for the <srctrig> and <desttrig> parameters.

Value	Source/Destination Line
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>
```

Parameters

Name	Description
controller	Controller on which to configure assertion mode. -1 = Local controller or the first remote controller
line	Trigger line to configure
mode	Configuration mode

The following table defines possible values for the <line> parameter.

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)

The following table defines possible bit values for the <mode> parameter.

Bit	Configuration Mode	
0	Specific Line Configuration Modes	
	0	Synchronize rising edge of CLK10.
	1	Synchronize falling edge of CLK10.
1	General Configuration Modes	
	0	Synchronize with next CLK10 edge.
	1	Pass trigger through asynchronously.
2	0	Do not participate in SEMI-SYNC with external trigger acknowledge protocol.
	1	Participate in SEMI-SYNC with external trigger acknowledge protocol.

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>
```

Parameters

Name	Description
controller	Controller on which to configure the TIC counter. -1 = Local controller or the first remote controller
mode	Configuration mode
source	Trigger line to configure as input to counter
count	Number of input pulses to count before terminating

The following table defines possible values for the <mode> parameter.

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

The following table defines possible values for the <source> parameter.

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

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>
```

Parameters

Name	Description
controller	Controller on which to configure the external connection. -1 = Local controller or the first remote controller
line	Trigger line to configure
mode	Configuration mode

The following table defines possible values for the <line> parameter.

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

The following table defines possible bit values for the <mode> parameter.

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

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 the TIC chip internal dual 5-bit tick timers.

Command Syntax

`trigtickconfig <controller>, <mode>, <source>, <tcount1>, <tcount2>`

Parameters

Name	Description
`controller`	Controller on which to configure the TIC chip internal dual 5-bit tick timers. -1 = Local controller or the first remote controller
`mode`	Configuration mode
`source`	Trigger line to configure as input to counter
`tcount1`	Number of input pulses (as a power of 2) 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 2) to count before asserting TICK2 output

The following table defines possible values for the `<mode>` parameter.

Value	Configuration Mode
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.

The following table defines possible values for the <source> parameter.

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

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 Commands

VIC supports the following system interrupt handler commands.

enablesysfail

Purpose

Sensitizes VIC to SYSFAIL interrupts from the specified controller.

Command Syntax

```
enablesysfail <controller>
```

Parameters

Name	Description
controller	Controller on which to enable interrupt. -1 = Local controller or the first remote controller

Example

Enable the SYSFAIL interrupt on the embedded CPU (or first remote controller).

```
enablesysfail -1
```

disablesysfail

Purpose

Desensitizes VIC to SYSFAIL interrupts from the specified controller.

Command Syntax

```
disablesysfail <controller>
```

Parameters

Name	Description
controller	Controller on which to disable interrupt. -1 = Local controller or the first remote controller

Example

Disable the SYSFAIL interrupt on the embedded CPU (or first remote controller).

```
disablesysfail -1
```

enableacfail

Purpose

Sensitizes VIC to ACFAIL interrupts from the specified controller.

Command Syntax

```
enableacfail <controller>
```

Parameters

Name	Description
controller	Controller on which to enable interrupt. -1 = Local controller or the first remote controller

Example

Enable the ACFAIL interrupt on the embedded CPU (or first remote controller).

```
enableacfail -1
```

disableacfail

Purpose

Desensitizes `VIC` to ACFAIL interrupts from the specified controller.

Command Syntax

```
disableacfail <controller>
```

Parameters

Name	Description
`controller`	Controller on which to disable interrupt. -1 = Local controller or the first remote controller

Example

Disable the ACFAIL interrupt on the embedded CPU (or first remote 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
```

Parameters

None

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
```

Parameters

None

Example

Disable the local Soft Reset interrupt.

```
disablesoftreset
```

assertsysreset

Purpose

Asserts SYSRESET* on the backplane of the specified controller.

Command Syntax

```
assertsysreset <controller>, <mode>
```

Parameters

Name	Description
controller	Controller on which to assert SYSRESET*
mode	Mode of execution

The following table defines possible values for the <controller> parameter.

Value	Controller
-1	From local controller or the first remote controller
-2	All extenders

The following table defines possible values for the <mode> parameter.

Value	Configuration Mode
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 remote controller) without changing the current configuration.

```
assertsysreset -1, 0
```

enablesysreset

Purpose

Sensitizes VIC to SYSRESET* interrupts from the specified controller.

Command Syntax

```
enablesysreset <controller>
```

Parameters

Name	Description
controller	Controller on which to enable interrupt. -1 = Local controller or the first remote controller

Example

Enable the SYSRESET* interrupt on the embedded CPU (or first remote controller).

```
enablesysreset -1
```

disablesysreset

Purpose

Desensitizes VIC to SYSRESET* interrupts from the specified controller.

Command Syntax

```
disablesysreset <controller>
```

Parameters

Name	Description
controller	Controller on which to disable interrupt. -1 = Local controller or the first remote controller

Example

Disable the SYSRESET* interrupt on the embedded CPU (or first remote controller).

```
disablesysreset -1
```

Bus Extender Commands

VIC 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>
```

Parameters

Name	Description
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.
directions	Bit vector of directions for ECL lines. Bits 5 to 0 correspond to ECL lines 5 to 0, respectively.

The following table describes the options for the <lines> bit vector.

Option	Description
0	Disable for appropriate line.
1	Enable for appropriate line.

The following table describes the options for the <directions> bit vector.

Option	Description
0	Out of the VXI extender
1	Into 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>
```

Parameters

Name	Description
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.
directions	Bit vector of directions for TTL lines. Bits 7 to 0 correspond to TTL lines 7 to 0, respectively.

The following table describes the options for the <lines> bit vector.

Option	Description
0	Disable for appropriate line.
1	Enable for appropriate line.

The following table describes the options for the <directions> bit vector.

Option	Description
0	Out of the VXI extender
1	Into 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>
```

Parameters

Name	Description
extender	Logical address of VXI extender
utilmode	Bit vector of utility bus signals corresponding to the utility bus signals.

The following table describes the options for the <utilmode> bit vector.

Option	Description
0	Disable for corresponding signal and direction.
1	Enable for corresponding signal and direction.

The following table describes the bit values for the <utilmode> bit vector.

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>
```

Parameters

Name	Description
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.
directions	Bit vector of directions for VXI interrupt levels. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

The following table describes the options for the <levels> bit vector.

Option	Description
0	Disable for appropriate level.
1	Enable for appropriate level.

The following table describes the options for the <directions> bit vector.

Option	Description
0	Out of the VXI extender
1	Into 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

VIC supports the following auxiliary commands.

help

Purpose

Provides help on the specified command. If the specified command is not unique, it lists all commands with that prefix. For example, help vxi will list all commands with the prefix vxi.

Command Syntax

```
help <command>
```

Parameters

Name	Description
command	A valid command or a keyword for group of commands. The keywords for help are: cwsc, swsc, hldbac, lra sigc, intr, trig, sih, scc, bec, aux, and all.

?

Purpose

Provides help on the specified command. If the specified command is not unique, it lists all commands with that prefix. For example, `? vxi` will list all commands with the prefix `vxi`.

Command Syntax

```
? <command>
```

Parameters

Name	Description
command	A valid command or a keyword for group of commands. The keywords for help are: `cwsc`, `swsc`, `hldbac`, `lra sigc`, `intr`, `trig`, `sih`, `scc`, `bec`, `aux`, and `all`.

disablemonitor

Purpose

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

Command Syntax

```
disablemonitor
```

Parameters

None

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>]
```

Parameters

Name	Description
`controller`	Logical address of controller to monitor

The following table defines possible values for the `<controller>` parameter.

Value	Description
-1	Local controller or the first remote controller
-2	OR of all controllers

Example

Monitor Controller 24.

```
enablemonitor 24
```

set

Purpose

Invokes device-level calls. The status bar displays the name of the active device. At this stage, you should not specify <la> in the VIC commands that require it as the first parameter, such as Word Serial commands.

To leave device-level calls mode, use set with no arguments, or with the device on which VIC is running, that is, with the <la> of the embedded or CPU-MXI card.

Command Syntax

set <la> or <devname>

Parameters

Name	Description
la	Logical address of device
devname	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` `VIC` commands. If no parameter is specified, it displays the current access parameters.

Command Syntax

```
pparms [<accessparms>]
```

Parameters

Name	Description
`accessparms`	Access parameters

The following table defines possible bit values for the <accessparms> parameter.

Bit	Access Parameters	
0 to 1	VXI address space	
	1	A16
	2	A24
	3	A32
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
5 to 6	0	Reserved
7	Byte order	
	0	Motorola
	1	Intel
8 to 15	0	Reserved

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` `VIC` commands. If no parameter is specified, it displays the current width.

Command Syntax

```
pwidth [<width>]
```

Parameters

Name	Description
`width`	Data transfer width

The following table defines possible values for the `<width>` parameter.

Value	Data Transfer Width
1	Byte
2	Word
4	Longword

Example

Set width as byte parameter.

```
pwidth 1
```

peek

Purpose

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

Command Syntax

```
peek <address>
```

Parameters

Name	Description
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 VIC` command) to the specified address (in the address space with access parameters set by the `pparms VIC` command).

Command Syntax

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

Parameters

Name	Description
address	VXI address
value	Value to write

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 VIC screen and writes it to the specified file.

- This command is applicable to DOS only.

Command Syntax

```
scripton <mode>, <filename>
```

Parameters

Name	Description
mode	Mode used for write
filename	File to which the information is to be written

The following table defines the options for the <mode> parameter.

Value	Data Transfer Width
0	Writes only the commands
1	Writes both the VIC commands entered and the responses

Example

Capture all the information from VIC and write it to the file "temp.dat."

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

scriptoff

Purpose

Closes the script session.

- This command is applicable to DOS only.

Command Syntax

```
scriptoff
```

Parameters

None

cfon

Purpose

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

- This command is applicable to DOS only.

Command Syntax

```
cfon
```

Parameters

None

cfoff

Purpose

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

- This command is applicable to DOS only.

Command Syntax

```
cfoff
```

Parameters

None

rmentry?

Purpose

Returns Resource Manager (RM) information about a device.

Command Syntax

```
rmentry? [<la>]
```

Parameters

Name	Description
la	VXI logical address

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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?
```

Parameters

None

numladdrs?

Purpose

Gives the number of known logical addresses.

Command Syntax

```
numladdrs?
```

Parameters

None

cmdrtable?

Purpose

Displays the system hierarchy information.

Command Syntax

```
cmdrtable?
```

Parameters

None

a16memmap?

Purpose

Gives A16 memory map of the system.

Command Syntax

```
a16memmap?
```

Parameters

None

a24memmap?

Purpose

Gives A24 memory map of the system.

Command Syntax

```
a24memmap?
```

Parameters

None

a32memmap?

Purpose

Gives A32 memory map of the system.

Command Syntax

```
a32memmap?
```

Parameters

None

readregister?

Purpose

Reads the register (word access) on the specified logical address.

Command Syntax

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

Parameters

Name	Description
la	VXI logical address
reg	Offset within VXI Logical Address registers

☞ **Note:** ***When using device-level commands, do not specify the*** la. ***The*** la ***of the active device will be used.***

☞ **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, ***and*** VNUM.

Example

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

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

writeregister

Purpose

Writes the register (word access) on the specified logical address with the specified value.

Command Syntax

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

Parameters

Name	Description
la	VXI logical address
reg	Offset within VXI Logical Address registers
value	Value to write

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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

Parameters

None

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?
```

Parameters

None

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>]
```

Parameters

Name	Description
la	VXI logical address

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

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>]
```

Parameters

Name	Description
la	VXI logical address

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

Get the device information for 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>
```

Parameters

Name	Description
la	VXI logical address

☞ **Note:** ***When using device-level commands, do not specify the*** `la`***. The*** `la` ***of the active device will be used.***

Example

Soft reset the device at Logical Address 4.

```
devicereset? 4
```

$

Purpose

Executes VIC commands from a file.

Command Syntax

```
$ [<filename>]
```

Parameters

Name	Description
filename	File from which to execute commands. If <filename> is not specified, $ brings up a file selection dialog so that you can choose the name interactively.

Example

Execute the VIC commands in the file "temp.dat."

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

history

Purpose

Displays the last 20 VIC commands.

- This command is applicable to DOS only.

Command Syntax

```
history
```

Parameters

None

Example

Display the last 20 VIC commands.

```
history
```

!

Purpose

Executes the previously executed VIC command. If no parameter is specified, it executes the last VIC command in the history of commands.

Command Syntax

```
! [<command string>]
```

Parameters

Name	Description
command string	Command string

*

Purpose

Executes the last VIC command for the specified number of times. If no parameter is specified, it executes the last VIC command once.

Command Syntax

```
* [<repeat number>]
```

Parameters

Name	Description
repeat number	Number of times to execute the last command

Example

Execute the last VIC command 10 times.

```
* 10
```

version

Purpose

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

Command Syntax

```
version
```

Parameters

None

system

Purpose

Exits the VIC utility temporarily to the operating system.

- This command is applicable to DOS only.

Command Syntax

```
system
```

Parameters

None

quit

Purpose

Quits the VIC utility.

Command Syntax

```
quit
```

Parameters

None

Appendix

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a Fax-on-Demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by application engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the /`support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	 Telephone	 Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 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 _____

Title _____

Company _____

Address _____

_____

Fax (____) _____ Phone (____) _____

Computer brand _____ Model _____ Processor _____

Operating system (include version number) _____

Clock 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™ Graphical Utilities Reference Manual*

Edition Date: June 1997

Part Number: 371696A-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 (____) _____ Fax (_____) _____

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

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

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
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 64 KB *short* address space. In VXI, the upper 16 KB of A16 space is allocated for use by VXI devices configuration registers. This 16 KB region is referred to as *VXI configuration space*.

A24 space — One of the VXIbus address spaces. Equivalent to the VME 16 MB *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.
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

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.

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.

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.
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
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

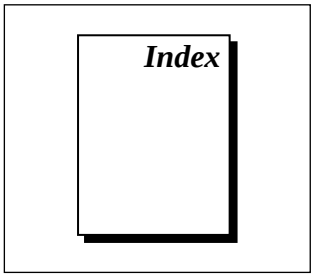
unasserted	A signal in its inactive false state.

V

VME	Versa Module Eurocard or IEEE 1014
VIC	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
VXIedit	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.



Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W